Automated Design Synthesis of Structures using Growth Enhanced Evolution

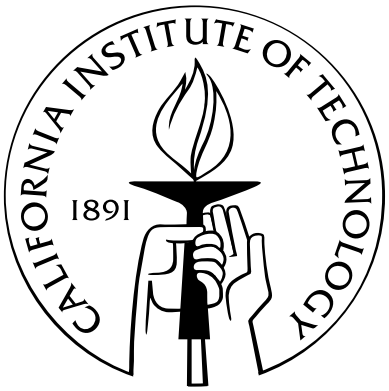
Thesis by

Fabien Nicaise

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2008

(Defended 19 September 2007)

Organic life beneath the shoreless waves

Was born and nurs'd in ocean's pearly caves;

First forms minute, unseen by spheric glass,

Move on the mud, or pierce the watery mass;

These, as successive generations bloom,

New powers acquire and larger limbs assume;

Whence countless groups of vegetation spring,

And breathing realms of fin and feet and wing.

- Erasmus Darwin,

Grandfather of Charles Darwin

The Temple of Nature 1802 [28]

To err is human, but to really foul things up you need a computer

- Paul Ehrlich

Why kick the man downstream who can't put the parts together because the parts really weren't designed properly?

- Philip Caldwell,

CEO of Ford Motor Co.

Acknowledgements

There are a many people that I need to thank for their help and support over the last few years. First and foremost is my advisor, Professor Erik K. Antonsson. He is the kind of advisors most graduate students dream about, letting you plot your own course and make your mistakes while making sure that you actually make progress towards graduation.

Second, I'd like to acknowledge and thank the rest of my committee, Chris Adami, Ken Pickar, and Joel Burdick, for their knowledge, insight, and guidance. I'd like to single out Chris Adami who was almost a co-advisor at times.

I'd like to thank my friends and colleagues for their support and for getting me out of the office once in a while. Last, though certainly not least, I'd like to thank my wife for support over the years.

Abstract

Engineering design is a complex problem on generating and evaluating a variety of options. In traditional methods, this typically involves evaluating up to a dozen different point designs. The limit on the process is the amount of time to generate, refine, and evaluate the various concepts. Using a computer helps to speed up the process, but human involvement still remains the weakest link.

The natural extension of this process is to continually and rapid generate, refine, and evaluate concepts entirely automatically. Evolutionary Algorithms provide such a method, by emulating natural evolution. The computer maintains a population point design, each of which is represented by a gene string that is allowed to change (mutate) and combine with other genes (crossover). At each generation, every individual is modified then evaluated and the improved solutions proceed to the next generation.

This thesis will extend the biological model by introducing a growth process to each individual. This is akin to the concept of a multi-cellular organism developing in the womb. An encoding for discrete truss structures is described that provides for such an extension. The truss grows from a few basic elements. After showing several examples demonstrating the growth process, the method is applied to a couple simple examples using evolutionary algorithms.

Contents

[1]This section is very similar to Adami et. al.[8] as many of the concepts work in the same way.

List of Tables

List of Figures

Listings

Chapter 1

Introduction

From the war of nature, from famine and death, the most exalted object which we are capable of conceiving, namely, the production of the higher animals, directly follows. There is grandeur in this view of life, with its several powers, having been breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.

– Charles Darwin (1809-1882),

Origin of Species. [27]

1.1 Background

The common image of an engineer's job to take a model of system and analyze it to predict its performance, with a little time thrown in for modelling. The design portion of the loop, as shown in Figure 1.1, is often ignored. In many cases it's left to one of two schools of

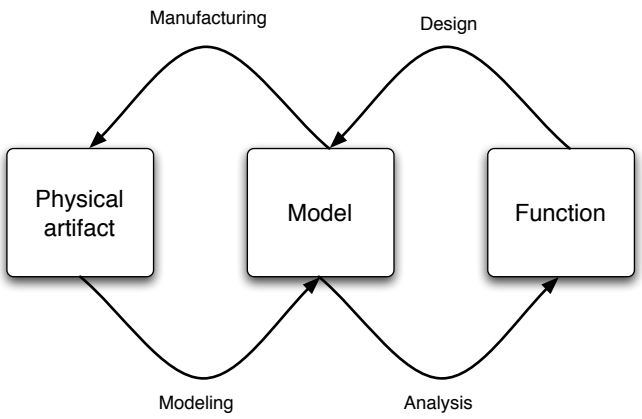


Figure 1.1: Overview of the engineer's job.

thought: "it's easy, you'll figure it out" and "either you have it or you don't." This is clearly an exageration, as there is a wide spectrum in between, but not unrealistic. Even Richard Feyman lays out his training in mechanical engineering design [40]:

> You look in the Boston Gear catalogue, and select those gears that are in the middle of the list. The ones at the high end have so many teeth they're hard to make. If they could make gears with even finer teeth, they'd have made the list go even higher. The gears athe low end of the list have so few teeth they break easy. So the best design uses gears from the middle of the list.

There must be a better way than this!

There is in fact a better way. There are many better ways. One of those is to formalize the process, doing proper trade studies, defining requirements, and doing all of those things that engineering students get taught in their design class. But this does not address the seemingly impossible first step of how do you generate the concepts that will be analyzed in a trade study? You are still left with that "magic" step of creativity.

For certain, a creative person has an advantage here, but they are also fraught with biases from past experiences. While those are good, as they develop engineering judgment, a bad experience for the wrong reasons can often color future decisions inappropriately. This is one of the places where automated design tools become useful. They can quickly and efficiently explore the design space, which is frequently much larger than a designer can properly explore, and identify viable candidate solutions. Those candidates in hand, the designer can then confidently do the analyses and trade studies that they have been trained for.

If they are so useful, where are these tools? The reality is that synthesis is a rather complicated task for a computer as well. In fact, almost all such tools aren't really sythesis tools at all but rather search tools, exploring the design space at a much faster rate than human designers. Where as a designer may have time for only five to ten candidates, some of the leading computerized search tools can consider millions of candidates in a similar time frame.

These tools are still quite complex. In the world of mechanical engineering design, and in structure synthesis specifically, there have been some attempts to create these tools. One of the popular avenues to accomplishing this task is to use Evolutionary Algorithms. Essentially applying Darwinian style evolution to engineering problems. To do this, designers must represent their design space in a genome that can be evolved. There are a few methods that have been tried (these will be discussed at length in the next chapter) all translate the design directly, using what is called *Direct Encoding*. This can take the form of a FEM input deck or a variety of other similar representations.

These direct encodings however have met with little success, as will be discussed in Chapter 2. They are too rigidly defined and do not respond well to the variation operators imposed by the algorithms. They also do not scale well when the scope of the problem is changed, for example if the structure requires thermal information, it can take a long rewrite of the genome and the details of the algorithm.

This thesis will propose an alternative. Instead of using Direct Encoding and representing the structure directly, this work uses concepts from nature to represent instructions for building the structure, a concept called *Indirect Encoding*.

1.2 Motivation

The conceptual design phase of any engineering task is an incredible time when ideas seem limitless and possible. Nothing has been done before on this new concept and engineers are free to guide it how they will. This is the 'clean slate' time on a project.

It is also a time during which decisions will be made that determine the future success (or failure) of designs. Coincidentally, it is also the time at which the least is known about the design and what things are likely to work. Mistakes, or more likely faulty assumptions, made during conceptual design may not be detected for weeks, months, or even years on larger projects. These errors can also be very costly to repair as they may require starting the process over at conceptual design again.

Because of this, it is critical to get a broad exploration of the design space. Humans, while very creative, are very poor at exploring the design space for a variety of reasons:

- Previous experiences may produce a bias against or towards particular concept.

- A tendency to like and defend 'your ideas', despite the training not to do so
- The size of the design space may be too large for one person to explore
- The design space may be too complex to efficiently sub-allocate to various persons
- Multiple competing factors, and the lack of knowledge about which factors to consider make comparison of alternatives difficult
- There is never enough time to look at all of the options

Formal methods [11] provide a mechanism for working around some of these issue. One popular method is the use of Evolutionary Algorithms: a technique based on Darwinian evolution and applied to engineering artifacts shows much promise and has been used in many fields. However, in the area of structural and mechanism design, it has met with only limited success.

If the success that has been obtained using Evolutionary Algorithms could be applied to problems of structural design or further to configuration design, there is a potential for saving much time, money, effort, and aggravation.

1.3 Thesis Contributions

This thesis is focused on the creation and implementation of a new encoding scheme for synthesizing discrete structures with Evolutionary Algorithms. There were two major steps in this undertaking:

- Develop an encoding scheme that can be used to grow a truss

- Develop an evolutionary framework to evolve

At the time this work got started, all Evolutionary Algorithms used a *Direct Encoding* scheme. This is especially true of mechanical design, where the long history of Finite Element Methods (FEM) provides a ready-made solution to encoding. But this does not lead to a very evolvable solution. Instead, we return to the natural inspirations that started Evolutionary Algorithms and use a more natural growth. Now the gene represents instructions on building an individual. To demonstrate the capablities of this technique, several examples were hand coded to show the growth capablity.

The second important contribution of this thesis is the generation of a framework to evolve the rule sets developed in the first step. This involves creating the mutation and crossover operators for the new genome definition. It also requires defining and implementing an evaluation technique. Initially, this was done using Nastran, a commercial analysis package, but that prooved too costly and took too much time to run so a custom analysis solution was implemented. Again, to demonstrate the capabilities, several example cases were run through the evolutionary process to generate candidate solutions.

1.4 Chapter Overview

Chapter 2 will begin by framing the discussion with a background of mechanical design and the methods in use in mechanical design. Special attention will be paid to Evolution Algorithms in their varied forms, as well as what are the preceived deficiencies of these methods as currently used.

Chapter 3 will go back to the basic biology and seek some understanding of the process

at play in the natural world, as they relate to evolution. Part of this will be examining how the nature of DNA and the fact that it does not directly represent organism but rather stores a set of blueprints on how to build the individual. This chapter will discuss how biological growth occurs and begin to lay out a method that will simulated in subsequent chapters.

Chapter 4 will utilize the concepts of Chapter 3 to create a new encoding scheme, *indirect encoding*, for use in the types of problems outlined in Chapter 2. This chapter will also discuss the specific implementation of Evolutionary Algorithms used in this work.

Chapter 5 will utilize the method outlined in Chapter 4 to present some examples using the method and compares it with traditional methods.

Chapter 6 is a summary chapter that will take a broad look at all the work described herein. The chapter will discuss some of the limitations of the work and suggest some fixes and future developments that could be made in this area.

Chapter 2

Engineering Design

We try to solve the problem by rushing through the design process so that enough time is left at the end of the project to uncover the errors that were made because we rushed throught the design process. [66]

– Glenford Myers (originally found in McConnell)

2.1 Introduction

Engineering Design, as opposed to design or industrial design, is focused an the iterative application of analysis and calculation in order to meet a prescribed set of requirements [76]. This does not necessarily imply any more rigor or formalism in the process, though it probably should.

This chapter will take a brief look at engineering design, especially in the conceptual design phase of the process. While much of the discussion is applicable to the broad field of engineering design in general, the discussion will be narrowed into a more manageable chunk by focusing on the design of mechanical systems, specifically truss structures. Auto-

mated methods in design optimization and synthesis will be examined as they relate to the work that will be presented in later chapters. Then, the perceived deficiencies with these methods will be discussed.

This chapter will also examine, briefly, some of the competing methods that were not used directly but do provide interesting background. These are other computational optimization and/or synthesis tools that could be useful in future endouvors. There will be brief justification, if not very rigorously, for why these were not used.

Finally, emerging methods in engineering design in general and in Evolutionary Algorithms will be examined. These provide some insights into ways to approach the solution we seek and will provide some context for the next chapters.

2.2 Design in Engineering

Design. It is such a simple word and such a simple concept: create something that currently isn't. And, yet ask ten people about design, and you are likely to get ten different answers. Maybe even eleven different answers.

In his book, Petroski [71] relates a story about an architect, a psychologist, and an engineer having a discussion about design (not the set-up for a joke). In the discussion, the architect argues for aesthetics, the psychologist argues for ease of use, and the engineer just wants it to work. The reality is that all three are correct, design must involve all three of these aspects. In fact it must involve many more aspects as well: cost, political considerations, legal aspects, and so much more.

Because of all of these competing constraints, design is often a difficult and challenging

task. The subtitle to Petroski's book is: *"Why there is no perfect design,"* to which he gives the short answer to just a few pages into the book:

> All design involves choice, and the choices often have to be made to satisfy competing constraints.

Before a choice can be made however, there needs to be choices from which to choose. It is these choices, and more specifically, where they come from that is of interest in this thesis. This is the art, science, and practice of synthesis.

2.2.1 Engineering Design

Scott [76] tells us that *"Engineering design distinguishes itself from other field of design by its use of calculation and analysis."* To return to the example from above, it is difficult to imagine a computational definition of aesthetics for architect to study or measure of intuitivenes for the psychologist, even if some quantification can be done through polls or such devices.

The distinction is, of course, not so clear. Ertas [37] ask in his first chapter if design is art or science. He states that intuition is an important aspect of engineering design. He proposes that the "intuition of a well qualified designer" is key to the process. But this is not very satisfying, even if true. According to Ertas, one has to design to be a good designer, which means that at some point, there are less good designers doing design work so they can become good designers, in an apprenticeship role; a less than desirable situation in the relatively rigorous world of engineering (though unavoidable).

Kicinger [55] divides the design process methodologies into three categories: Formal

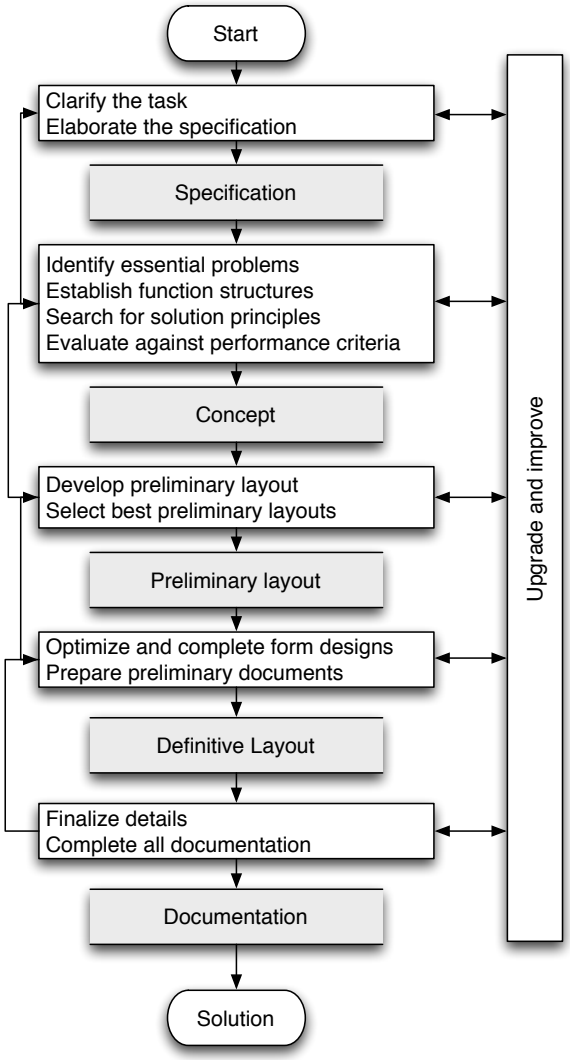


Figure 2.1: The engineering design process. Adapted from [69]

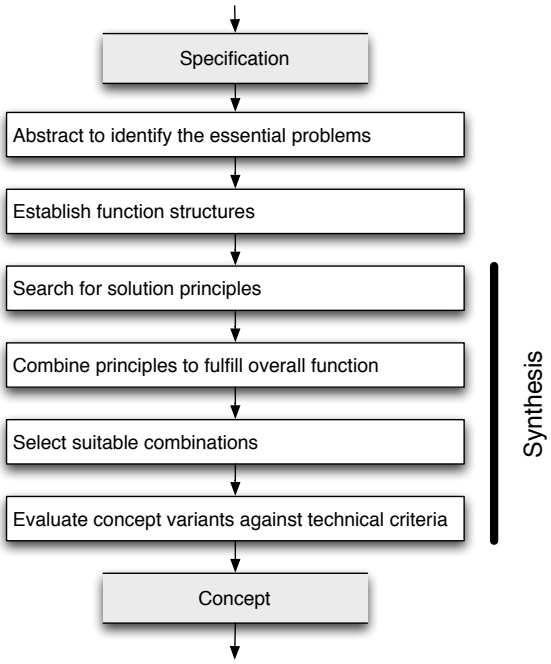


Figure 2.2: The conceptual design process. Adapted from [69]

models, heuristic models, and Agent-based models. Of these, it is the formal methods that are the most relevant to this thesis.

Many have proposed formal methods to assist in the process of design, to ensure that the best designs can be promulgated, even for less experienced designers [11, 81, 69]. As Scott [76] points out, the advantage of formal methods is that they allow for codification and computation. Which, in turn, means that they have the possibility of automation. Because of this key advatage, formal methods form the basis of this thesis.

2.2.2 Conceptual Design

Conceptual design is the earliest phase of design where candidate solutions are considered. In conceptual design, all of the problems of engineering design are concentrated. And they are multiplied by the fact that the concept is just begining to take shape and not much is

known about it. The process is so involved that some are even applying chaos theory to study it[12].

Pahl and Beitz [69] gives a a very concise definition for the process:

> Conceptual design is that part of the design process in which, by the identification of the essential problems through abstraction, by the establishment of function structures and by the search for appropriate solution principles and their combinaiton, the basic solution path is laid down through the elaboration of a solution concept.

That is quite a mouthful, but accurate. In other words, the conceptual design is that part of the design where the basic features of a candidate solution. This is a very iterative process where one concept will be generated, studied, and evaluated based on its merrits.

One of the key phrases that Pahl and Beitz use is "the basic solution path." At the conceptual design level, very little is known about the actual design that will emerge. For this reason, it is important that the concept that emerges at the end of the conceptual design be a concept that is flexible enough to accomodate downstream changes as knowledge increases. For this reason, it is inappropriate to talk about optimization in conceptual design. Optimization produces tall, sharp peaks in quality, but they tend to fall off fast an the parameters change. In conceptual design, the opposite is desired: the peak must be high enough to satisfy the requirements, but it must also be broad, as the design and performance parameters will change, sometimes quite a bit during later design phases.

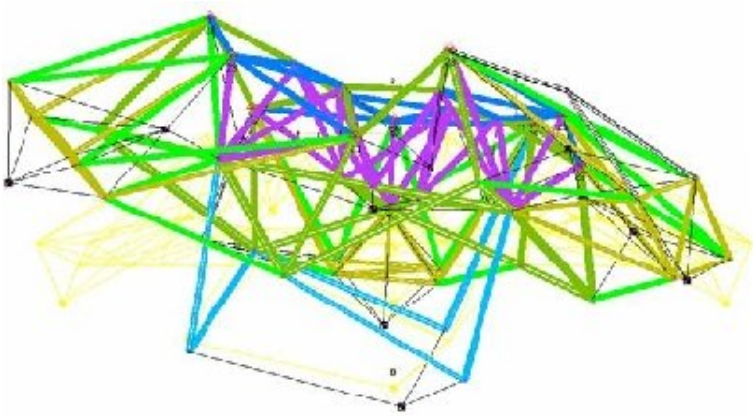


Figure 2.3: An early conceptual layout of the structure for the Mars Science Laboratory Descent Stage. Courtesy of JPL.

2.3 Problems of Interest

In order to make this a realistic thesis, some effort must be made to limit the scope of the task. The type of artifact of interest is the mechanical structure. More specifically, the discrete structure, also known as a truss structure. A sample of such a structure is shown in Figure 2.3

There are many reasons that such a choice was made. The first is that it has to be made. The breadth of design, even structural design is simply too large for the focus of one project. Trusses were selected because of their relative simplicity. They are easy to conceptualize, easy to imagine, and relatively easy to analyze; though they remain just as complicated to design. Trusses lend themselves to quick and robust analysis methods that are ideal for an evolutionary framework. On the other hand, as was learned from the project, the discrete nature of trusses presents some unique challenges to the growth process.

Having thus limited the scope of study it is imperative to make a point about applicabiity. The method that will be presented in the next few chapters is aimed at the scope described above. It is however applicable to a wide variety of design tasks. Some time will

be spent in the last chapter describing reasonably simple extensions that could be made with relatively small effort, such as continuous structures or mechanisms. It is in fact a goal of this work that it should be relatively simple to expand, since this is one of the deficiencies with exist Evolution Algorithms (as will be shown shortly).

Lastly, problems of interest here are ones that are too large for a single designer to handle, but too complex and interdependent to subdivide among several engineers. The goal is not to replace the designer but give them more tools with which to accomplish difficult design tasks put to them. That being said, it is worth pointing out that most of the examples in this thesis are small, simple structures so that the principles can be understood.

2.4 Evolutionary Methods

Evolutionary Algorithms (EA) are a computational search and optimization tool based on the natural process of evolution. It is a broad term that encompasses several other methods and algorithms. After discussing the general character of Evolutionary Algorithms, we will then turn to some of the major algorithms that are encompassed by the umbrella term.

In order to make any Evolutionary Algorithm work, there are three key areas that need to be defined: Selection; Mutation or Variation; and Reproduction or Transmission. These are, in fact, the same three elements that Charles Darwin observed over 150 years ago as the key elements in natural evolution [27]. Figure 2.4 shows how these three properties act in a cyclical and continuous fashion. A little later, we will describe these processes in detail.

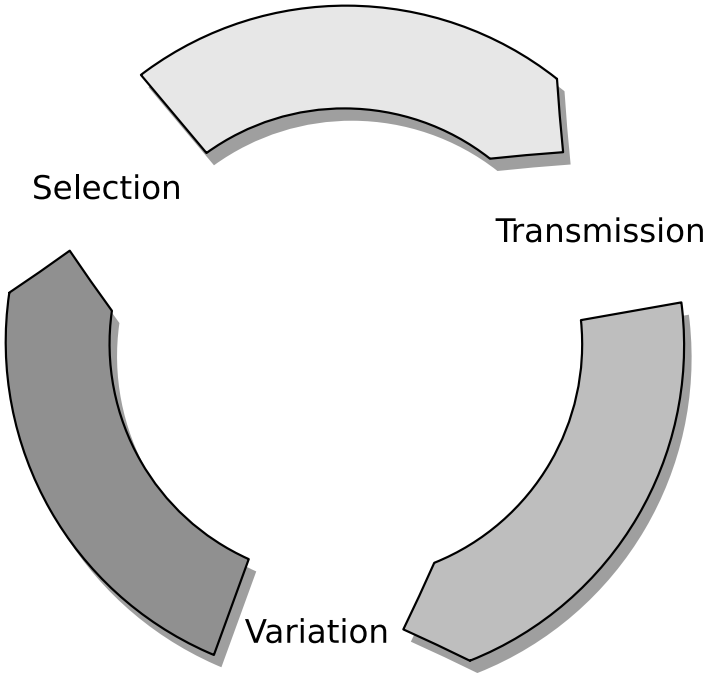


Figure 2.4: D
iagram of basic evolution as found in nature.]Diagram of basic evolution as found in nature.
Adapted from [60]

2.4.1 Key element of EAs

As mentioned previously, the three key aspects of Evolutionary Algorithms are Selection, Mutation, and Reproduction. This section will examine those three aspects in detail and then will briefly discuss the small modifications that are required to to transition from the natural world to the artificial silicon world (which has been termed *in silico* to contrast with *in vivo* for live nature and *in vitro* for a specimen in a controlled lab[1]).

2.4.1.1 Generational Selection

Selection, in short is the basic means by which an individual is selected from the population to pass its genetic material on to the next generation. The natural version will be discussed in more detail in Chapter 3. For now, we'll simply say that the fitest individual survives; this could be the fastest, tallest, strongest, or simply the most attractive individual.

In silico, there are many more options available, as well as many more pitfalls. First, we need to define what makes one individual "better" than another, using the proper language: what makes one organism *fitter* than another. This is typically done by what is called a fitness function. This is some performance metric on the individual. The nature of this function is as diverse as the applications. In structural mechanical engineering, we typically evaluate for stress level, natural frequency, and mass. Additionally, we could also evaluate based on raw material cost, manufacturing cost, reliablity, or a myriad of other domain specific factors. The final trick in defining these fitness functions is aggregating all of the performance metrics into a single metric to describe the individual. There are a few common aggregation functions:

[1] usually credited to Pedro Miramontes, 1989.

Maximum Exactly as it sounds, take the maximum individual score to be the aggregate score. This usually requires some non-dimensionallization of the various performance variables.

Minimum Like maximum, but using the minimum score.

Sum Add all of the scores together to form an aggregate score. Again, non-dimensionalizing the performance variables tends to improve results. Could be a weighted sum or an average, as they are essentially the same.

Rms The root sum of squares method

Product Multiply all of the factors together to get the aggregate score.

Method of Imprecision(M_OI) A fuzzy method of aggregating disparate engineering information using designer preferences [77].

These methods all have benefits and drawbacks. The advantage of the first is that it is fast, but tends to let one parameter compensate for the others too much. For example, even if the structure is perfectly stiff, if it weighs ten times more than the earth, it is not realistic. Likewise, the minimum has the problem that it does not reward an individual fast enough for making a little progress in one area.

The sum and RMS aggregation functions also has many drawbacks in that they fail to properly account for scaling behaviour of the various parameters. The product function does not offer much compensation between different parameters, as they are all treated equally. There are many other possible aggregation functions, but these are the major ones in use. See [22] for a relatively recent survey of aggregation methods. Finally, $M_OI$offers a

good choice for design work, that separates the parameter scaling and the relative weighting, as well as the degree of compensation. More on this method will be provided in Chapter 4, where the algorithm implemented will be discussed in detail.

Having calculated the fitness score, there needs to be a method for selecting which individuals go to the next generation. This is the heart of *selection*. Again, while there are many methods available, these are the most popular:

Steady State Every parent produces one offspring and that offspring replaces the parent in the population. This is probably the simplest concept and the easiest to understand, but can be difficult if sexual reproduction (crossover) is used.

Elitism The best individual is carried to the next population. This could also be the best n individuals, where n can range up to the population size. If n is less than the population size, then one of the other methods would need to be used to select the remaining slots.

Roulette A non-uniform random selection in which the odds of an individual being selected is proportional to its fitness. This method can occasionally run into problems if the population has very high or very low fitness individuals as they will skew the odds and will tend to lead towards pre-mature conversion.

Tournament Two (or more) individuals are selected at random. The better one goes to the next generation and the others do not.

Rank Selection Like Roulette selection, but the odds are determined by the individual's ranking, not the individual's fitness itself.

Again, there are benefits and drawbacks to all of these methods. The basic idea is to balance passing good individuals to the next generation with converging too fast to a local maximum by not including enough diversity in the next generation. Again, more details on this will be presented in Chapter 4 as the algorithm is implemented.

2.4.1.2 Mutation

Mutation is the basic element of change in Evolutionary Methods, as well as in nature. A mutation is said to occur when a gene, or a portion of a gene, spontaneously changes in quality. This may or may not acutally produce an effect on phenotype, depending on where the mutation occurs.

Specific implementations of genomes all require somewhat different mutation operators. However, they all do share a couple of traits. The first kind of mutation is generally applicable: duplication, where whole sections of the genome over. The second is point mutations. These operate on a single aspect of the genome randomly changing it to another value.

2.4.1.3 Reproduction

Crossover is the process by which two (or potentially more) individuals combine to form a new individual (offspring). The details of crossover are even more dependent on encoding scheme (genome) than mutation is.

For an example, consider Figure 2.4.1.3. Two generic parents are created using a floating point representation. Then, an example of blend crossover is used to generate the child. In this case, the blend is the mean of the two parents. For this to be meaningful, though,

Parent 1	—1.2	5.6—
Parent 2	—3.4	7.8—
(a) Blend	—2.3	6.7—
(b) Mix	—1.2	7.8—

Figure 2.5: Examples of cross over. (a) Blend crossover, where each allele in the child is the mean of the parents' allele. (b) Mix crossover, where whole alleles are taken from each parent and passed directly to the child.

the positions on the genome has to be meaningful. If the positions of alleles on the genome is random, then it is unlikely that the blend crossover scheme would work.

The other type of crossover is the mixing crossover. In this case, alleles are taken as whole statements and passed unchanged to the offspring. In this case, position on the genome is less important, as the allele is transfered directly.

Having had this discussion about parents coming together to produce offspring, brings to mind a question: how are the parents chosen to produce each offspring. As expected, there are a variety of options:

Random Generate a uniform random number for each parent. Though, not the best idea, because Generational Selection is used to decide what individual lives to the next generation, this may not be a bad thing.

Rank Using an exponential random number, select the parent based on their rank in a list sorted by fitness. Once an individual has been used, it can either be put back into the population, to mate again, or be removed to give more chance to the others.

Direct Parents are assigned in sequential order based on fitness. (The best and second best mate, the third and fourth best mate, etc.)

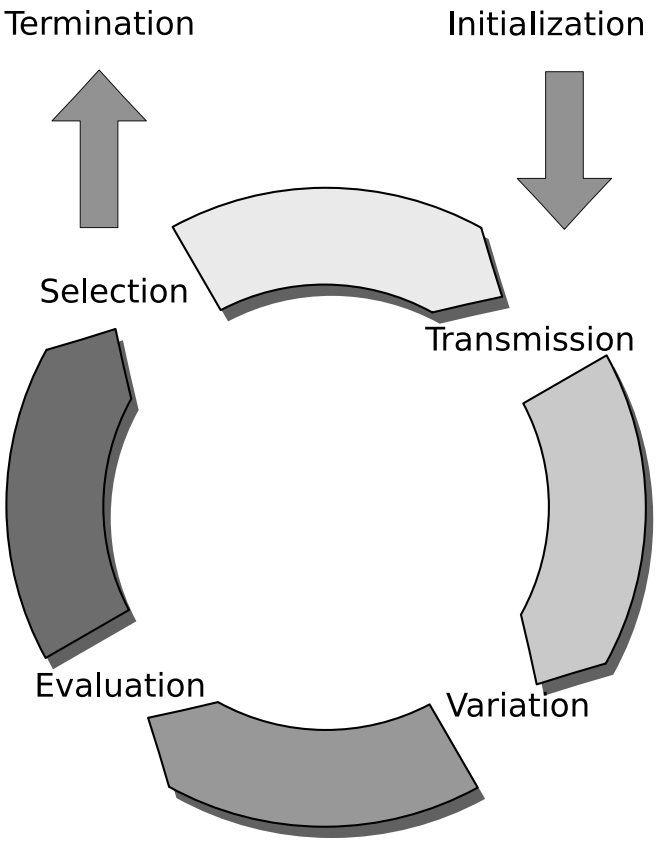


Figure 2.6: Diagram of basic evolution, modified with artificial constructs to make an evolutionary algorithm. [60]

2.4.1.4 Other aspects

As can be expected, when evolution is moved from the natural world to the computer, there are aspects that are special to the computational enviroment. These are initiallization, evaluation, and termination.

Initiallization is something that nature no longer has to worry about, even if scientists still struggle with explaining it. In the computational scene, there needs to be a way to

initiallize a population in order to proceed. One popular method of initiallizing a population is to seed it with a known valid individual. This gives the evolutionary process a solid start, but may put it too close to a local maximum that it cannot escape. On the other hand, the process could be started with a purely random set of genomes. This would give a broader start, but would cause many generations to be used simply to come up with a feasible solution.

Much mention has been made so far about the fitness of an individual, but so far no mention of where it comes from. In nature, it is clearly the ability to survive, but in-silico, it becomes a little more complicated. To get a fitness for an individual, it must be *evaluated* for desired performance. These fitness functions are as diverse as the applications and can be anything quantifiable. There are aspects to this function that can help or hinder evolution, but in principle there are no limits to what can be evaluated.

The last issue to be addressed is termination. When should the evolutionary process be stopped. This is one area where nature will be of no use. To the best availbale knowledge, nature never stops evolving, it simply continues to find smaller niches to take advantage of or changes the target as the environment changes. In the design world however, there must be a point where the process stops and the product is built. The simple answer is that the designer stops when all of their desires (preferences) have been satisfied.

If it were that simple, it would be really nice. However, in real design, it is rare that all parameters can be fully satisfied as many of them actually conflict with each other directly. Therefore, there is no perfect solution. In these cases, the designer must stop the process when the design is good enough or they run out of time. In practical consideration, for Evolution Algorithms, this usually means that the designer runs it in batches of generations

until satisfied.

2.4.2 Common Variations on EAs

As might be expected, there are many variations on this basic concept of using natural evolution as a basis for improving designs in engineering. Some are natural extensions of others, some where developed by two different groups in tandem, and some simply developed from a different point of view. In this section, we will examine a selection of the common algorithms that have been developed in the basic framework of Evolutionary Algorithms and how the fit in with the work under consideration.

2.4.2.1 Genetic Algorithms

Genetic Algorithms (GA) were an attempt to make a generic, application independent algorithm, in which the genome would be a fixed length binary genome. In this scheme, the mutation operator and crossover operator were trivial implementations of the random bit-flip and the random single point crossover, respectively. In reality, most implementation used real value encoding, and many even extended the method to variable length genome [61].

2.4.2.2 Genetic Programming

Although not the first to suggest it, Genetic Programming (GP) has been promoted by Koza [57]. It typically (though not required) uses a tree style genome to represent the data under consideration. This gives a recursively evaluated genome, as is often found in the Lisp language. Due to this nature, many times what ends up being evolved is a program

to be evaluated for fitness. This means that this technique is often found in attempts to design software. One interesting point that separates most GP from other Evolutionary Alogirithms is that GP tends to only use crossover, there is generally no point mutation operator.

2.4.2.3 Evolutionary Programming

Evolutionary Programming (EP) was first described by Fogel [42] consisted of a very basic algorithm of population size N, that was augmented to size $2N$ by adding the children of mutation then selecting the N best for the next generation. The encoding for EPs generally resembled GA encoding, in that it was a direct representation of parameters from the solution. Although capable of both sexual and asexual reproduction, most success with EPs was derived from purely asexual recombination, most likely due to the encoding scheme [52]. The population size and mutation strength varied widely accross a variety of test problems.

2.4.2.4 Evolution Strategy

Evolution Strategies (ES) are a class of Evolutionary Algorithms that maintain a population of 1 individual. The lone parent then creates m new offspring. Of the augmented population of $(1 + m)$, the single best individual is chosen to survive to the next generation. These were originally applied to a sequence of several parameters in an optimiziation application.

2.4.2.5 Classifier System

Classifier Systems (CS or CFS), also somtimes called Learning Classifier Systems (LCS), were initially proposed by [47]. The vision for Classifier Systems is to have a system

capable of sensing its environment and then reacting appropriately. THis is actually a strong precursor to the current work, except that it appears to have very strong difficulties in actually working.

2.4.2.6 Summary on EAs

As can be seen in the previous few sections there are many types of Evolutionary Algorithms. The reality is that they are all very similar in their methods. There is no sharp line to distinguish them from each other. In fact, as can be seen reading the above sections, there is a very blurry line between them, sometimes to the point that they are virtually indistingishable.

For this reason, this thesis will not try to fit into one of the other classes. As has been done by others [52], the methods used in this thesis will be refered to as an Evolutionary Algorithm, and not further refine the definiton. For the current purpose, these are the characteristics of an Evolutionary Algorithm:

Population A set of candidate solutions shall be maintained that consists of at least one solution

Encoding the candidate solutions shall be encoded in some method to allow the other elements to proceed. This can be a sequential or non-sequential set of values (binary, integer, floating point, or even character).

Variation the candidate solutions shall be changed or varied in an incremental fashion. This may take the form of asexual reproduction where one parent is used or sexual reproduction where more than one parent is used.

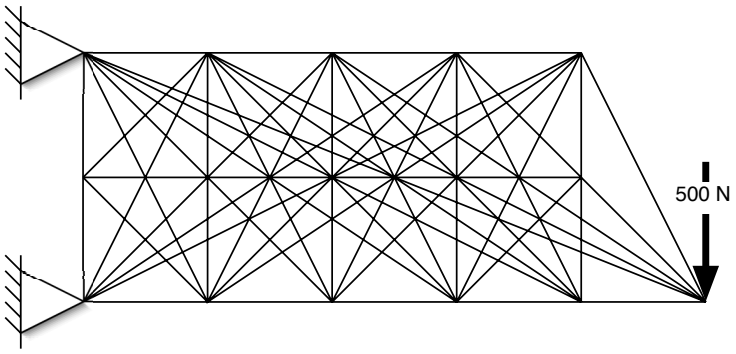


Figure 2.7: Sample of a base structure in two dimensions, where every possible solution is included in the structure and the genome encodes which links are turned on or off.

Evaluation The candidate solutions shall be challenged to meet a performance goal and ranked among other solutions as to how well it performed.

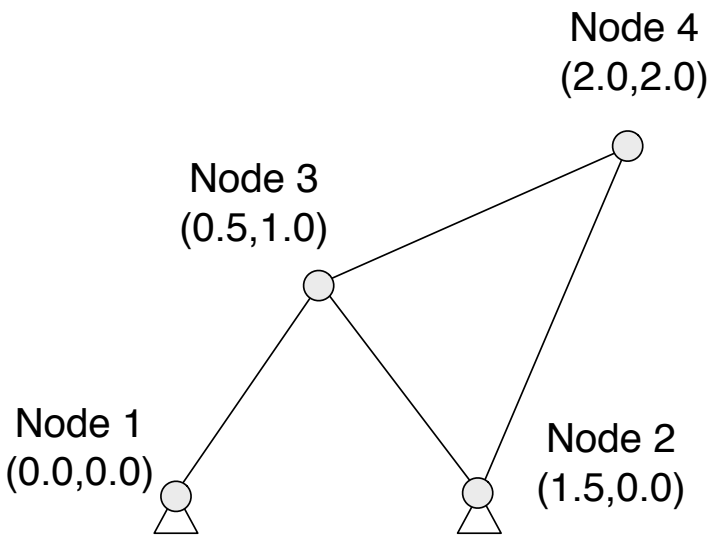
Selection the candidate solutions shall be challenged to remain in the population based on its fitness.

2.4.3 EAs in Structural Design

Evolutionary Algorithms have been a popular technique to apply to structural design synthesis problems because of their inherent lack of designer bias and potential for quality output.

The first major encoding scheme used in structural synthesis is the *base structure*. In this encoding scheme, the genome is a set of real values, indicating the cross-sectional area of elements in a pre-defined, over-constrained structure, such as in Figure 2.4.3. If the area drops below a certain threshhold, that element is removed for that individual. In that sense, every possible solution already exists, waiting for evolution to scape away the excess.

The other major encoding scheme used is structural synthesis is what will be called



Genome: [0.0 0.0][1.5 0.0][0.5 1.0][2.0 2.0]
[1 3 A_1][2 3 A_2][3 4 A_3][2 4 A_4]

Figure 2.8: Sample of FEM type encoding, in two dimensions (brackets added for clarity). The first gene of the genome represents the coordinates of the nodes and the second gene the connectivity between them.

FEM encoding. In this encoding, illustrated in Figure 2.4.3, uses the commonly used representation of a truss for Finite Element Modelling. In this scheme, the genome is divided into two parts. The first part is the node data and contains a listing of the position of the nodes. The second part is the element data and contains the two end points of the element along with section properties (in the example, only crossectional area is tracked.)

** discuss examples from the lit

2.5 Other Design Methods

While this chapter, and in fact this whole thesis, has focused on Evolutionary Algorithms as a tool for design, there are other methods in use. This section will briefly look at a few of them: simmulated annealing, swarm based design, and agent based design as alternatives.

In addition to briefly describing their functionality and uses, I will also compare them to the ideas suggested in this thesis and why they are not compelling or useful to the perceived needs.

For the purposes of this document, we ignore the more traditional optimization methods, such as newton-rhapson(some examples are [83, 70, 23]) or hill-climbing (examples are [51, 33]) algorithms. These methods all require a relatively smooth, continuous search space. In addition, they all need to have a fixed complexity to have any hope of success. Both of these requirements are contrary to conceptual design in mechanical or structural engineering. There is no *a-priori* knowledge on the scale of complexity of the system under consideration. By that very fact, the continuity is violated as adding a new set of parameters is an inherently discontinuous operation.

Instead, we will focus on the more potent techniques for searching large spaces. Most of these are also stochastic techniques, like Evolutionary Algorithms, but operate on a fundamentally different principal.

2.5.1 Simulated Annealing

Simulated Annealing (SA) is another class of stochastic search algorithms. Originally proposed by Kirkpatric et. al. [56] and Cerny [21] simmultaneously in the mid 1980's, is based on the the ideas of metallurgical annealing, where the atomic crystalline structure of materials is altered, and generally improved, by application of a heat treatment involving high temperatures and followed by a slow cooling to working temperatures.

This method has many similarities with evolutionary methods. The both rely on suc-

cessively evaluating candidate solutions and then disturbing, or mutating them. In the case of Simulated Annealing, only one individual is considered at any given time step. If this individual is better than the previously stored state, it is adopted as the the new stored state. If the candidate individual is not as good os the stored state, then it is usually thrown out, although there is a small probability that it will be kept any way. This last part is the critical step. It allows the algorithm to escape from a local minimum, which we have already discussed is a critical ability for any algorithm that traverses complex, discontinuous, multi-modal performance lanscapes. It is also from this last operation that the method get's its name: the probability is controlled by a *temperature* parameter that monotonically decreases to zero over a optimization run, as it would in mettalurgical annealing. In someways, Simulated annealing can be considered as a Genetic Algorithm with a population of one (though some careful choices in parameters will have to made in order for this to be true in the strictest sense).

Simmulated annealing has found a home in a variety of optimization problems. The most classic is the travelling salesman problem because of its simplicity and elegance in posing and evaluation yet complexity in solving. [62] is but one of many examples of using Simulated Anealing to tackle the travelling salesman problem. Others [74] have use Simmulated Anealing to solve fuzzy logic linear programming problems. There are more practical promlems that have been addressed as well. For instance [82] makes use of Simmulated Anealing to optimize a job shop flow. Simmulated anealing has also taken a strong root in electrical circuit design. [44] and [65] are two examples in this vast field. And even closer to our interests, [80] used Simmulated Anealing to optimize the configuration of an electrical trasmission tower.

Finally, to discuss the relative merrits of Genetic Algorithms versus Simulated Aneal-ing. This is a topic that has been broached before, mostly from a computer science point of view rather than an engineering point of view, however the results should still be applica-ble. Lahtinen et. al. [58] and Manikas et. al. [65] both tried to take some sample problems and run them through both algorithms. Both had mixed results. Manikas found that one of their three test problems were the same results and the other two were marginally better while Lahtinen found that the Simulated annealing performed marginally better. In short, there is no clear winner.

For our purposes here, we disregarded Simmulated Annealing for several reasons. First, for mostly historical reasons, we had experience with Evolutionary Algorithms and it did not make sense to switch at that time. Further, the Evolutionary Algorithms are implic-itly parallel, which was a very attractive feature. While Simulated Annealing can be made parallel, it is a more complex operation due to the serial nature of the algorithm. Finally, the idea that germinated in our musings on improving the design search process was bi-ologically based, and it appears as more creadible to have the biological basis applied to Evolutionary Algorithms. As a closing thought on this section, there is no reason that the work that will be presented here cannot be used to extend Simmulated Annealing, it is sim-ply a more tenous connection to use the biological extensions that will be demonstrated to extend a crystalline based method.

2.5.2 Ant Colony Optimization

Ant Colony Optimization(ACO) techniques were developed by Marco Dorigo in 1992 [30]

for optimizing systems; particularly, he used it to find shortest paths through graph structures. ACO, as it is commonly refered to, another biologically inspired method for searching through a large domain, in a similar fashion to the way ants forage for food. They start out randomly searching their domain, trying to find any source of food (valid solutions) in their area. As they find useful sources of food, they lay down pheromone tracks on their trip back to the colony. Other ants are then more likely to find that pheromone trail and follow it to the food and back to the colony.

There are two keys to making ACO work (as in real ants). First, the ants are governed by probability. If they find a pheromone trail, they may or may not follow it. Even when on the trail, they may or may not stay on it. This allows for further exploration of the space and helps keep ACO out of local optimums. The second key is that pheromone trails evaporate. This allows old, useless tracks to disappear and prevents ants from following useless, lower optimality tracks. The other benefit to evaporation, is that it provides a selection pressure (the shamelessly borrow a term from Evolutionary Algorithms). Because the tracks will evaporate, the ones that last the longest will be the ones that get traversed the fastest, i.e. the shortest paths. After searching in this way for some time, all the ants will eventually follow the same path, giving a termination condition, as the optimum has been found.

As mentioned above, this technique was developed for graph traversal. As such it has been applied to a variety of graph based problems. The most classic of which is the travelling salesman problem [32]. It does show up in a few limited structural engineering problems where they have been applied to creating an FEM mesh for analysis [59]. Another very popular area of application is telecommunication networks [20, 7, 24]. As might be expected, they also show up in neural networks [15, 39]. There are a few more areas

that have been examined with ACOs: circuit design [4], Web usage and data mining [5], evolutionary tree reconstruction [10], DNA sequencing [14], and many, many more [31].

So finally, the question is why did we not use this method to further our design synthesis work. The short answer is that ACO is very good at optimizing graph traversing whereas we are trying to generate the graph. The distinction is not trivial. ACOs are good, as already mentioned, at finding the shortest path through a graph of cities (travelling salesman). There is no analog to this in structural design. It might be akin to finding the most stressed load path in a given structure. While this might be a useful measure to use in the design process, it does not directly generate a better structure. The few examples in the literature that use ACOs to design structures [78, 16] only optimize the section properties of an existing structure. Applying this model to synthesis, is, again, not trivial. The simple method is to allow the optimization to make a null section, i.e. no connection. This essentially brings us back to the base structure method that was discussed above and clearly not a desirable condition.

2.5.3 Agent Based Design

Agent Based Design, often called Intelligent Agent Design (IA), is technique for providing one or more agents, bits of software with some intelligence, with a method for creating more complex systems. The basis is that they form the basis of complexity theory, where each little bit is simple and easy, but the whole creates complex interactions that were not originally expected.

These have been used in a variety of areas. In replication and understanding of nature,

agent based design has been used to create wasp nests from simple rules of the type that real wasps might use. It robot control, they have found widespread use [41, 79, 45]. One attractive aspect using agent based design in control is the ability to assign certain agents to be 'advocates' for certain behaviors or task while a master agent acts as the arbiter. They have also been tried on problems of interest to mechanical engineering as well [18, 17].

And finally to discuss why these were not used. The principal reason is that they require intelligence to be programmed into the agents. In practice, these are typicaly heuristic engineering rules. These are the same types of rules that engineers themselves follow. While this is one level of abstraction and can be useful in day to day work for reducing work load, especially on repetitive tasks, it is not the purpose of this project. We seek to use the computer to search for and generate non-obvious solutions. Ideally, solutions that an engineer would not come up with on their own. This, by definition, requires working in a purer space than that already defined by the engineering community.

Finally, it is worth noting that in many ways, as the work will show, that we have made use of many of the techniques in IA to make our work function. The agents are purely artificial and very basic, but each one acts on its own, independent of the others around it. It will be shown to be almost a corruption of IA.

2.6 Perceived Deficiencies

The various Evolutionary Algorithms discussed above, take individually or as a whole, all have some problems. The first is that they have only been mildly successful in structural design synthesis. The encodings required tend to limit the solutions, as in the case of the

base structure ** add example. More than this though, the strict direct encoding leads to four fundamental problems: lack of modularity, lack of emergence, inability to affect significant topological changes, and complex extension. Each of these will now be looked at.

2.6.1 Lack of Modularity

The standard encoding for Evolutionary Algorithms uses a direct encoding, the genome to represent the phenotype. It may be in code, such as binary or some other abstraction, but it is still an exact representation of the genome. Much the way that the word 'genome' could be protected by spies by coding it into 'fdmnld' (by using the letter immediatly before the one intended), it is still a direct representation of the word.

The poses a problem for enabling modularity. Exactly repeating a block of structure, does not help the structure in any way, and in fact violates the laws of nature (one piece of matter cannot occupy the same space as another). In the case of the base structure, the idea of a module becomes even more arcane. There is no concept of a module when you are turning on and off a given set of links.

2.6.2 Emergence

The concept of emergence is the idea that a concept or idea that was not specifically programmed will appear in the solution. This clearly cannot happen when a base structure is used, as every solution is already in the base solution, waiting to be uncovered. In the slightly less restrictive, direct FEM encoding, emergence still has issues, as the encoding

scheme reduces exploration of the search space.

2.6.3 Topology Changes

A further problem with the direct encoding method of Evolutionary Algorithms is the difficulty placed on the mutation parameter to affect a topological change. The actions required to make a topological change are:

1. create a new node.

2. create three new links to that node.

3. make sure the node is in a feasible location.

4. make sure the other three end point of the link connect to different nodes.

Assuming a 0.001 mutation rate (each parameter has a 0.001:1 chance of being mutated in each generation), the odds of just adding the four necessary items (steps 1 and 2 above) is 10^{-12}. These are staggering odds, even by evolutionary standards. And the last two steps have not even been accounted for yet.

Some work has been done to reduce this step by 'applying a bandaid' to the mutation function. For example, when a mutation indicates the creation of a node, three links are automatically added [48] But this is not satisfying as it is just putting the designer's knowledge and experience back into the simulation, which was a strong goal of using EAs to begin with.

2.6.4 Expansion

Another issue, is that any time an EA wants to be expanded in scope, it essentially needs a complete, or nearly complete re-write of the genome encoding. The reason for this is that the genome was conceived specifically for the purpose, adding new elements changes the way mutation is applied as, for example, thermal parameters may not mutate the same way that structural parameters. Likewise, crossover will need to be redifined to account for the fact that it cannot add structural data where thermal data is expected, or vice-versa. This problem would be even worse if one tried to incorporate complete non-mechanical properties, such as electronics or fluidics.

2.7 Summary

This chapter has taken a broad look at the field of engineering design and the various methods available to assist the engineer in doing their job. Many of the problems with those tools were identified and examined for potential remedy with the new method proposed later in this thesis.

In order to make the task more feasible, the scope of study was also restricted to discrete mechanical structures (Trusses). They are conceptually a simple idea and the analysis of them is fairly straightforward, even though the task of designing them properly remains as difficult as any in engineering design.

The next chapter will take a look at the original inspiration for Evolutionary Algorithms, Nature, for some guidance on how to improve them beyond current capability.

Chapter 3

Biological Inspirations

We do not arbitrarily give laws to the intellect or to other things, but as faithful scribes we receive and copy them from the revealed voice of Nature.

– Sir Francis Bacon [53]

3.1 Introduction

Having looked at the deficiencies with artificial evolution in the last chapter, attention will now turn to finding a way to improve the situation. In order to do this, the natural world will be examined in further details. Some effort will be expended on understanding how and why evolution works in the natural world.

This will lead us into a discussion of the encoding scheme used by nature (DNA). This will involve a discussion of development in biology, as a single cell grows and develops into a large, multi-cellular creature.

This will lead into a brief overview of artificial development models that have been created by others. Finally, this will be translated into a set of guidelines on what an encoding

scheme for artificial structural design synthesis should look like.

3.2 Evolution

Evolution is the improvement of a genetic line by the accumulation of beneficial changes, *i.e.*, mutations when one base (A, C, G, or T) is replaced by another. There are many reasons that such a change may occur. It is important to realize that the fitness evaluation in nature is always done at the fully grown individual level. The actual code does not play an important role in the evaluation of fitness other than creating the individual to be evaluated.

The fact that the code does not directly represent a distinct structure creates some interesting properties. A point mutation of a single base could cause no change at all in the case where it simply encodes the same amino acid, but in a different way. Similarly a single point mutation can also cause drastic changes: when a mutation causes the protein that activates the 'eye' gene gets mutated away, the eye may never grow. While a seemingly inconsequential detail, this could actually be a critical mechanism. In most cases, we want mutations to change little, in order to keep individuals alive. However, to achieve significant exploration of the design space, a single point mutation has to have the capacity to create large change.

The differences in the DNA code among various animals is minute. Mice, apes, and humans all share more than 99% of their genetic material. [46] Most of the differences among different animals are in the regulatory mechanisms of the DNA. Furthermore, most changes in evolution in the recent genetic past have been almost exclusively in the regulatory mechanism. This means that between humans and their closest ancestors, there has

been little change in what the DNA code contains, but rather the changes have been made to how cell development is regulated.

3.3 The Code

The first step to understanding how evolution works in nature is to understand the code. This overview begins with the most basic structures and proceeds to more complex structures.

Deoxyribonucleic Acid (DNA) contains a backbone, made essentially of sugar, that supports an assortment of bases, which provide the information encoded in the molecule. The four bases that make up DNA provide all the information for the growth, development, and regulation of the organism whose growth processes it encodes. These four bases: Adenine (A), Cytosine (C), Guanine (G), Thiamine (T) are grouped in sets of three in an ordered list. Each of these groups translates into one of 20 different amino acids. However, simple combinatorics indicates that there are 48 possible sets of 3 bases. Clearly, these are not unique translations. As described in [46], some different code combinations actually are interpreted the same amino acid. One major reason for this is to protect the code. Some mutations, while changing the actual code, do not change the interpretation.

A group of these amino acids that appear in a row combine to form a protein molecule. The variety of proteins is almost limitless. It is the proteins that are produced by the encoding in the DNA that do all the real work in the cell.

3.4 Development

The primary purpose of the proteins is to ‘act’ in the cell. These ’actions’ can take many forms. The proteins play roles in allowing other chemicals to pass into and out of the cell, and also serve as catalysts in many cell functions.

Additionally, the proteins perform a signaling function, both inside and outside of the cell. Outside the cell, they communicate their existence and needs to surrounding cells. Within the cell, they cause the cell to have an identity.

The final purpose of the proteins is cell regulation. Some proteins can cause certain portions of the the DNA to not be read. This represses certain functions or activates others. This causes cells to act in different fashions. Cells with the ‘eye’ protein, activate certain portions of the DNA to be read, causing structures like irises and retinas to be made. Similarly, those areas are turned off in cells that contain the ‘liver’ protein while those areas that cause those cells to produce the enzymes to break down alcohol are activated. [19]

3.5 Artificial Biology

This is not the first work to try to emulate natural processes. This section wil briefly look at several examples that were drawn from, at least in part.

3.6 Artificial Structural Growth

One of the keys that was identified above is the ability for a single mutation to cause both small and large changes. In classic EC’s, achieving both of these is challenging. Small

changes occur quite readily: for example, the in the evolution of structures, the location of a node might be mutated from $x = 3.231m$ to $x = 3.522m$. However, large changes in a single point mutation are more difficult, as discussed in Section **??**. The mutation of adding a node is a much more difficult task, as not only does the mutation have to add the node, it must also determine the correct connections to make a valid structure.

The approach to this problem for truss-like structures is to develop a set of rules that can 'act' on the truss. These could be rules such as *grow a new connection towards node X* or *strengthen the connection to node Y*. The details of implementing this approach are not clear at this time and certainly not proven.

The development would also need the ability to sense it's environment using conditional statements. This would create some differentiated cells that react to certain stimuli differently from other cells. One cell may be a 'growth' cell that attempts to connect across gaps in the structure, while other cells may be 'receptor' cells that signal that a connection is needed.

There may also be certain additional benefits to an indirect, rule based encoding over a traditional encoding. One possibility is that this approach may produce greater robustness. Many biological systems can overcome many otherwise crippling defects to still produce viable individuals, such as the starfish, which can regenerate entire arms that are cut off during the growth process. Another possibility is that good code sequences will be applicable to multiple tasks. Because of the code itself would be responsive to the environment in which it is placed, a well evolved yet generic code could provide a good seed solution to most problems of interest.

This approach may appear to be similar to GP, since it evolves programs to create

structures. For example a GA might use statements like 'There is a link between nodes i and j'. GP would instead use a statement like 'Connect nodes i and j by a link' However, there are two major differences between GP and the approach outlined here:

1. *Use local rules, not global rules.* This would be akin to creating multi-cellular organisms, rather than a large single cell. Each portion of the structure (or generalized artifact) should act independently of distant portions of the structure. It should, however, respond to its local environment, both of the structure and the external stimuli, using the signaling properties of the simulated proteins.

2. *Move away from psuedo-assembly instructions* and instead use growth-based encoding that responds to the local environment, including sensing the conditions in the environment and taking appropriate actions.

3.6.1 Basic Necessities of an Encoding Scheme

Any encoding scheme has to adhere to a few basic rules to work properly. Kicinger [55] summarizes perfectly a few basic rules originally laid out by Gen and Cheng [3].

Non-redundancy Each genotype should represent only one phenotype to ensure there is no ambiguity in phenotype.

Legality Each possible genome should represent some phenotype. Note that *illegality* is not the same as *infeasibilyt*, where a genome represents a phenotype that lies beyond the constraints on the design space.

Completeness Any point in the phenotypic design space must be representable by some genome. If not, portions of the candidate solutions may be excluded.

Lamarckian property The alleles of a genome must have the same meaning/action, regardless of where it appears. Gen and Cheng argue that this is a requirement for inheritability.

Causality The scale of the effect on the phenotype, should be proportional to the scale of the change on the genotype. Gen and Cheng refer to this as continuity.

3.6.2 Examples Generative Encodings

While the work presented here is novel, there have been a few similar types of work presented. Some of them were used an inspiration on the direction to take the work and several of them have shown up in the literature more recently, as this work was wrapping up, and need to be mentioned for completeness.

There are also a few interesting examples of using Lego^{TM}bricks to represent the design [43, 72]. These two methods use an assembly sequence of Lego^{TM}bricks to represent the final phenotype. While not strictly a generative method of the style desired, this is a great leap forward from the standard direct encoding.

Aside from the biologically motivated examples described above, the earliest known related to this field are based on L-systems, originally developed by Aristid Lindenmayer [63], a cellular automata based method. One of these methods is an L-Systems representation for making voxel tables [49, 50]. In a similar class of problems, cellular automata have also been used to represent designs of buildings [54].

There are also a few examples that evolve generative representations, but apply them to design types of problems. The first is an example of using the ideas of cell division to mesh a part for FEM analysis [75]. In this example demonstrates the ability of a small set of instructions to create a quality mesh after some evolution. In this case, the mesh is applied to an existing artifact and is not used to design the part. Another example is the use of genetic regulatory networks for controlling robots. Kumar [**?**] shows one such example, where the genome is a list of reactions that occur based on input stimulus proteins generated by the sensors. The results in case end up a genome that represents a coding scheme to control their robot, but the genome still represents the controller directly, it does not create a new contoller topology for use in the robot.

Finally, there are some genetic network systems that have not been applied to evolution. The best example of this is work by Nagpal et. al. [68]. In this work, the authors use 2-D shapes (circles) to act as cells that can only communicate with their immediate neighbors. While this work did not evolve their genomes, they did observe some of the key goals of using representative systems: the ability to represent complex structures with simple rules and robustness to a variety of external stimuli.

3.7 Conclusions

This chapter has examined the biological background in evolution and how it works in nature. It has also examined how nature encodes it organisms and the basics of the genotype to phenotype conversion. This process, also known as development, is one of the keys to a successful encoding scheme.

Finally, with this knowledge in mind, some basic rules and constraints were placed on what would make a good encoding scheme for use in structural design synthesis. This knowledge will lead into the next chapter which will discuss the encoding scheme that was created for this thesis.

Chapter 4

Computing Environment

The bad news is that, in our opinion, we will never find the philosopher's stone. We will never find a process that allows us to design software in a perfectly rational way. The good news is that we can fake it.

– David Parnas and Paul Clements [66]

4.1 Introduction

The Chapter 2 described the percieved deficiencies with a broad range of Evolutionary Algorithms as used in structural design synthesis. Most of these stem from the fact that a traditional, directly encoded genome does not provide a suitable mechanism for tolerating mutation and crossover operations.

After that, Chapter 3 described what is observed in the original, natural evolution, as observed first by Charles Darwin and subsequently by thousands of biologist. This lead us to a better understanding of the mechanisms involved evolution and gave us a glimpse of why it works.

This chapter will take ideas from biology and apply them to the perceived deficiencies of artificial Evolutionary Algorithms in an effort to improve their performance and usefulness. As an introduction, the chapter will start with a general discussion of what is expected in a growth model and how it should look and behave.

After that, we will discuss the artificial environment that is created *in-silico*, a virtual incubator of sorts for the structures to develop in. Then the genome that makes the structure actually grow will be described. We'll provide a few simple examples of the growth to demonstrate the feasibility and potential of the model. We will also present a brief proof to demonstrate that the genome model has the ability to span the entire design space. Finally, we will discuss the details of the Evolutionary Algorithm used to evolve these genomes.

4.2 Goal of a Growth Model

What we seek is a model similar to the one outlined in [8]. The simulation grows a final phenotype that cannot be divined simply by inspection of the genotype. One of the keys to generating is to realize that the model will create a multi-cellular organism.

Astor and Adami [8] outlined four principals of molecular and evolutionary biology. They are listed here with the explanation from Adami and Astor along with comments on why they are considered important for furthering EC tasks:

Coding: The model should encode structures in such a way that evolutionary principals can be applied. Clearly, this is the heart of the matter and the motivation for this task.

Development: The model should be capable of growing a nework by a completely decentralized growth process, based exclusively on the cell and its interactions. There is no master

control the review the actions and decide that something looks ‘wrong’ and should be fixed.

Locality: each cell must act autonomously and be determined only by its genetic code and the state of its local environment. This removes the idea of cells being able to query the ‘master control’ during growth. Cells are allowed only to look at what they are doing in their local neighborhood in which to decide their actions.

Heterogeneity: The model must have the ability to describe different, heterogeneous cells in the same network. This is akin to having bone cells in the body and muscles cells. They serve different purposes and without either one, the body does not function.

The goal of this model goal of such a system to remove the designers influence over the design process. ECs that have implemented ‘fixes’ to improve exploration or convergence rate tend to simply transfer designer bias at a different level. This method makes the structure independent of the designer. Also, by making the cells develop their own structures, they will be able to re-use the same coding to create similar structures, providing some level of modularity.

4.3 Artificial Growth

4.3.1 Environment and Chemistry

In order to grow an individual, there needs to be an environment in which to grow, an agar of sorts. In this case, a very simple version of reality will be used. This is a three dimensional, continuous world with a gravity imposed. To encourage growth, the elements

are able to sense gravity, but are not affected by it. i.e. the do not fall while unsupported. (Note: This is not the case during evaluation, only during the growth portion.)

In this world, proteins secreted by the cells (Nodes) provide for all the intercellular communication. These proteins are allowed to diffuse through the artificial world using a simplified diffusion model. Rather than trying to solve the partial differential equation directly, either analytically or numerically, this one is done by a simple model. Each 'injection' of proteins is modeled by it's own decaying gaussian pulse:

$$p_{ij} = C_0 \frac{\exp\left(\frac{K_1((x-x_0)^2+(y-y_0)^2+(z-z_0)^2)}{t-t_0}\right)}{K_2(t-t_0)} \tag{4.1}$$

for every source j and every protein i. (x, y, z) is the location in question and (x_0, y_0, z_0) is the location where the protein was added with original concentration C_0. K_1 and K_2 are parameters used to define the spacial and temporal decay of the concentration of proteins. These should be tweaked to improve performance, depending on the scale of the task.

Then the global effect is found by summing all the individual contributions:

$$p_i(x, y, z)|_t = \sum_{j=0}^{N} p_{ij}(x, y, z)\Big|_t \tag{4.2}$$

for every protein i.

Using this simple model, the gradient of the protein is easily calculated. The individual

contribution of one source is:

$$\nabla p_{ij} = 2K_1 \frac{\exp\left(\frac{K_1((x-x_0)^2+(y-y_0)^2+(z-z_0)^2)}{t-t_0}\right)}{K_2(t-t_0)} \begin{bmatrix} (x-x_o) \\ (y-y_o) \\ (z-z_o) \end{bmatrix} \tag{4.3}$$

$$= 2K_1 p_{ij} \begin{bmatrix} (x-x_o) \\ (y-y_o) \\ (z-z_o) \end{bmatrix} \tag{4.4}$$

Then, using the linearity of the gradient operation, the global effect is found by summing all of the individual components:

$$\nabla p_i(x,y,z)|_t = \sum_{j=0}^{N} \nabla p_{ij}(x,y,z)\Big|_t \tag{4.5}$$

where all variables are as defined above.

At each time step of the growth process, all of the concentrations and gradients are updated based on what happened during the last iteration of the processing. All of the cells are constantly producing their own proteins and creating a dynamic environment.

As more complicated designs are considered (such as with obstacles) it would be worthwhile to consider a more accurate model that actually tracked real diffusion.

There are two classes of proteins used: *External Proteins* are free to diffuse across the entire space. External proteins serve to communicate between cells. External proteins secreted by one cell can cause the behavior of another cell to active, terminate or intensify. External proteins also provide the direction of many actions, such as growth, by way of the

gradient.

Internal Proteins are constrained to remain inside the cell. While they do not diffuse across the space, they do decay, so if the cell does not produce more, it will eventually disappear. Since internal proteins do not diffuse across space, they cannot be inter-cellular communicators; however, they do further specialize certain cell functions as well as serve as intra-cellular communicators, providing information from one gene to the next.

4.3.2 Nodes (Cells)

The heart of the system is the node. These are the synthetic analog of the biological cell. Each node acts independent of the others and without *direct* knowledge of the actions of others. It does however remain able to sense another cell's actions using the diffusabe proteins described above. At each time step, the cell executes it's genetic code and acts based on it's local environment. This is analogous to how a cell in your body works. There are a variety of different types of nodes, which will be described separately. Nodes come in three basic varieties.

Base nodes serve as the grounding point for any truss structure. In order to have a variety of behaviors, base nodes are further subdivided in categories by an integer. These are often fixed in space by the designer, and therefore, gene rules that instruct the node to move are often disregarded.

Load nodes are the points (loads) to be supported. Like base nodes, load nodes are also subdivided by an integer. Depending on designer needs, these nodes may or may not be constrained.

Field nodes make up anything that is left. Any new node that is created in the course of running the code is automatically a field node. Field nodes, like the other nodes, can also be differentiated using an integer identifier. These nodes are free to move around the design space freely.

Connections between nodes are established by *links*. These have all the basic properties expected from a standard mechanical truss: cross-sectional area, material, and moments of area.

While they are unconnected, the only exist as *proto-links*. While growing, the proto-links are actually property of the node that is growing them. There are two ways to convert a proto-link into a full link:

1. The tip of the proto-link gets close to another node.

2. The code causes the proto-link to convert into a full link and a new node is added at the end.

At this point, if a proto-link encounters another protolink, nothing happens. This might be an interesting area to explore in future development.

4.3.3 Links

Links are mechanical elements that connect the nodes together to form a truss. These are non-active elements in the growth simulation. Nodes do all of the work and when they connect, they form a link. Therefore, links are very simple devices and only carry own information: cross-sectional area and material properties specifier.

While still growing, a link is called a proto-link. At this stage, it is a dependent of the node that is controlling it. A proto-link will remain a dependent property of the node until it finds a place to tie off its free end point and become a full link. While in this state, the parent link is able to control all aspects of the proto-link, including, the cross-sectional area, material properties, and growth size.

4.4 Genetic Code[1]

Having set the stage with the physics and chemistry of this artificial world and having met the participants, nodes and links, it's time to describe the rules of the game.

Every cell operates independently from all the others, but they use the same genome. The functions encoded on this genome is a custom creation for the purpose of designing these the structures of interest. While they are loosely based on what is observed in bi-ololical growth, no attempt is made to reproduce with fidelity the actual process found in biological systems; the goal is only to be inspired by the observed mechanisms. This language is now described.

The genome found in each cell consists of a series of genes. These genes act like mini-programs that sense the environment and take an appropropriate action. Each gene consists of two stings of statements. The first string contains all the statements that control regulation functions while the second sting contains the development functions.

[1]This section is very similar to Adami et. al.[8] as many of the concepts work in the same way.

Table 4.1: Conditional statements with veto power. $CTPx$ stands for any cell type, as described in Section 4.3.2. Similarly, PTx stands for any protein type, internal or external, as outlined in Section 4.3.1.

Rule	Represses Gene if
$SUP\{CTPx\}$	cell is not of type CPTx
$NSU\{CTPx\}$	cell is of type CPTx
$ANY\{PxX\}$	there is any protein of type PTx
$NNY\{PTx\}$	there is no protein of type PTx
$NLL\{x\}$	number of attached links less than x
$NLM\{$x$\}$	number of attached links more than x
$LGT\{$x$\}$	length of proto-link longer than x
$LLT\{$x$\}$	length of proto-link shorter than x

4.4.1 Conditional Statements and Regulation

The first step in growing a truss is to evaluate the conditional statements to figure out which genes are active and what their effect is.

There are two basic classes of conditional statements: repressive and evaluative.

Repressive condition statements statements that have the power to completely shut off or repress the entire gene. Mostly for expediency in computing, the evaluation takes a first pass through all the conditional statements to identify any repressive genes. If any is found that causes a repression, evalution of that gene is immidiately stopped and the evalution continues with the next gene in the genome. Examples of these genes are listed in Table 4.1.

Evaluative condition statements are conditional statements that can take on any real value. The only real limit to the value is the machine precision. In actuality, most of the evaluations of note are on the same order of scale as the domain of problems under consideration. In order to evaluate a conditional statement, it must take in an initial condition

Table 4.2: Evaluative conditional statements. $CTPx$ stands for any cell type, as described in Section 4.3.2. Similarly, PTx stands for any protein type, internal or external, as outlined in Section 4.3.1 and {PTx} is the local concentration of that particular protein.

Rule	Evaluation value Φ	
NOC	$\Phi(\Theta, NOC)$	$= \Theta$; no change/neutrality
ADD{PTx}	$(\Theta, ADD[PTx])$	$= \Theta - \{PTx\}$
SUB{PTx}	$(\Theta, SUB[PTx])$	$= \Theta + \{PTx\}$
MUL{PTx}	$(\Theta, MUL[PTx])$	$= \Theta * \{PTx\}$
ANY{PTx}	$(\Theta, ANY[PTx])$	$= \Theta$, if $\{PTx\} \neq 0$
NNY{PTx}	$(\Theta, NNY[PTx])$	$= \Theta$, if $\{PTx\} = 0$
AND{PTx}	$(\Theta, AND[PTx])$	$= \min(\Theta, \{PTx\})$
NND{PTx}	$(\Theta, NND[PTx])$	$= -\min(\Theta, \{PTx\})$
ORR{PTx}	$(\Theta, ORR[PTx])$	$= \max(\Theta, \{PTx\})$
NOR{PTx}	$(\Theta, NOR[PTx])$	$= -\max(\Theta, \{PTx\})$

(based on what previous statements have done, and the concentration of the protein it is tied to. Examples of evaluative conditional statements are listed in Table 4.2.

To better understand how the evaluation proceeds, the same formalism is introduced as in [8]. Let C be a vector of condition statements of length n with components a_i, with $1 < i \leq n$:

$$C = (a_1, a_2, \ldots, a_n) \tag{4.6}$$

For easier processing, this vector is broken down into two vectors, one of repressive statements of length m and one of evaluative statements of length l:

$$C_1 = (b_1, b_2, \ldots, b_m) \tag{4.7}$$

$$C_2 = (c_1, c_2, \ldots, c_l) \tag{4.8}$$

where C_1 contains all repressive statements and C_2 contains all evaluative statements. Like

wise, b_i and c_i are the repressive and evaluative statements, respectively. Note that $l + m \geq n$ since some statements may be both repressive and evaluative, as shown in Tables 4.1 and 4.2.

Now the evaluation proceeds.

1. Check through all the statements in C_1 for any condition that is not met. If any item exercises it's veto power, evaluation stops.

2. Evaluate C_1. The result is the condition value for this gene and is used by the expressive statements.

Now, assume that the first step has passed without any veto being exercised. Step two is then to evaluate C_2. Define $\Phi(\$Theta, c_i)$ as the evaluation function of evaluative statement c_i with some initial condition Θ. The return value of each evaluative statement becomes the initial value of the next statement. Then the overall condition value Φ_C of the gene C is

$$\Phi_C = \Phi_{C_2} = \Theta(\ldots\Theta(\Theta(\Phi_0, c_1), c_2), \ldots, c_l). \tag{4.9}$$

Clearly shown however is that the first statement requires some initial value. This is simply set to $\Theta_0 = 0$.

In closing out this section, several notes are made. The evaluative statements, C_2 are not commutative, therefore order in the vector is important and must be maintained. On the other hand, repressive statements, C_1, are clearly commutative, as any one of them can trip the veto power and cause all execution to stop.

Table 4.3: Expressive statements. Growth of proto-links follows the local protein gradient. PTx stands for any protein type and $CPTx$ stands for any cell type.

Rule	Statement Description	Use of Θ, the condition value
$GRD\{PTx\}$	Grow proto-link following gradient of PTx	Distance to grow
MAT	Change proto-link material	Index of the material to use
MOM	Increase (decrease) the cross sectional area of the proto-link	Amount to increase (decrease)
RLX	Scale the cross sectional area of the proto-link	Scale factor
$MOV\{PTx\}$	Move the node following the gradient of PTx	Distance to move
$PRD\{PTx\}$	Produce more protein PTx	Quantity to produce
$SPL\{CTPx\}$	Split off the proto-link into a full link and create a new node of type $CTPx$	none
DIE	Remove the node and all connecting links	
NOP	No action, neutrality	none

4.4.2 Expressive Statements and Truss Development

Once the gene regulation has been evaluated, it's time to apply some actual growth rules and do something constructive. It is important to note that each expressive statement is an independent action. These statements act based on the condition value calculated above and a few more specific environmental constraints, such as protein concentration gradient. Examples these rules are sumarized in Table 4.3 and each rule is described in detail below.

4.4.2.1 Proto-link Development

As discussed in the Model section above, connections between nodes (cells) are represented as links. During their growth however, they are termed proto-links. Expressive statements control growth of the structure by the growing proto-links; once they become links, they are beyond the touch of the evaluation.

The simplest command of interest is *GRD*. *GRD* causes the node to start growing, or continue if one was started earlier, a proto-link. *GRD* is linked to one of the diffusible proteins. The local gradient of this linked protein determines the direction of growth of the protolink. The conditional value calculated above determines how far the proto-link grows.

MAT selects the material that the link will be made of. The calculated conditional value provides the index of the desired material. This corresponds to a user supplied table of approved materials. Since the calculated value can be any real value, it must be treated to give an integer that can be used in the look-up table. First, the negative is dropped if needed. Then it is truncated to give an integer (this gives a better distribution over valid indices than rounding). Finally, if the interger is greater than the number of available materials, the modulo is taken and gives the final material index.

The other commands of interest to growing proto-links are *MOM*, and *RLX*. These two commands change change the cross-sectional area of the proto-link. *MOM* is an addition function and adds the conditional value to the existing area. *RLX* is an addition function and scales the the existing area. Note that in order to get a valid area, the *RLX* command drops any negative signs that may appear in the condition value.

4.4.2.2 Link Development

As proto-links grow and develop, they will eventually need to become defined links. There are two ways that this can happen. Using either the *GRD* statement or the *SPL* statement.

The first one, *GRD* has already been explained above. There is one detail left to capture. If, in the process of growing, the tip of the proto-link should come near an existing node, the proto-link will connect to that node and convert to a full link. ‘Near’ is a user parameter

that is generally chosen based as a percentage of the physical domain of the design task. Typically about 1% works well, but it depends on user intent.

The other method of spawing a link is the *SPL* command. This command explicitly causes the proto-link to convert to full link. The end of the proto-link is converted into a new node and a link is established between the spawning node and the newly created one. The type of the node created is a parameter of the *SPL* command. The new node will begin acting like any other node, starting in the next round of evaluation.

4.4.2.3 Other Expressive Statements

The *MOV* command allows the node itself to move about in its environment. The conditional value determines how far it moves and the local concentration of protein determines the direction. When a node moves, it takes with all the internal protein that it has acquired as well as any links and/or proto-links that maybe connected to it.

Finally, there are the housekeeping statements. Under the *PRD* statement are several related activities. The first is to produce internal proteins, which help to further specialize the cell activities. The second task of *PRD* is to secrete external proteins which communicate to the other nodes in the area. These are similar tasks, but it is important to separate the two, as they serve two very different purposes. In both cases, the conditional value determines the amount of protein to produce.

4.4.3 Simulation in Action

As is often the case in Genetic Algorithms or other Evolutionary Algorithms, the collection of nodes and links produced by gene evaluation is called an individual or an organism.

The purpose is to create a self-sustaining, viable structure that can then be evaluated for performance.

Each simulation is started in a similar fashion. The designer specifies a number of base nodes, where the truss will be grounded, a number of load nodes where objects need to be supported, and, if desired, some field nodes to seed the simulation. Each node has the same genome as every other node in the simulation from which to derive its behavior.

At each discrete time step, each node senses its environment, evaluates each genes, and produces the appropriate action. To prevent any problems with degenerative cases, this is done synchronously. The simulation stops when no new actions have occured. Additionally, to prevent wasting time and resources, the designer specifies a maximum number of iterations that the simulation may run.

4.5 Examples

4.5.1 Self-limiting Growth

The most trivial example of the growth rules is one base node and one load node. A genome is then written that will connect the two nodes with a link. The simulation should then stop. This situation and its genome is shown in Figure 4.1.

4.5.2 Small Truss

The next simplest case to actually build a simple truss structure. The scenario is very similar. One load node is placed in the field and three base nodes are added. Then the

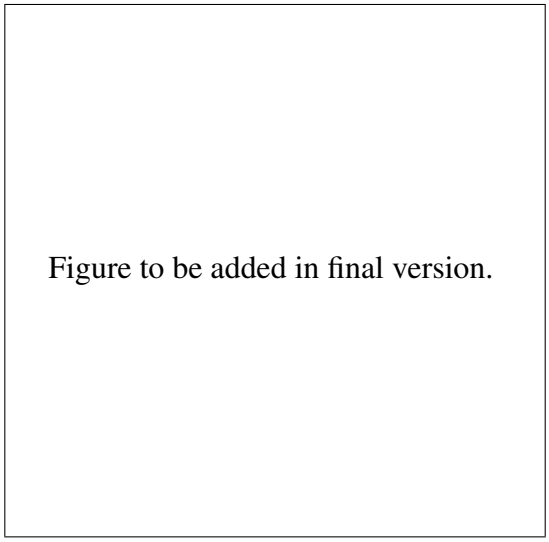


Figure 4.1: Simple example of self-stopping growth.

simulation is run to allow the base nodes to support the load. The genome and the results are shown in Figure 4.2.

4.5.3 Large Truss

The next step is to create a more complex truss. In this case, there are six base nodes and three load nodes. The genome and the results are shown in Figure 4.3.

4.5.4 Diversity

Finally, to show the diversity possible in such a scheme, a non-engineering example is created. The results of this simulation is shown in Figure 4.4.

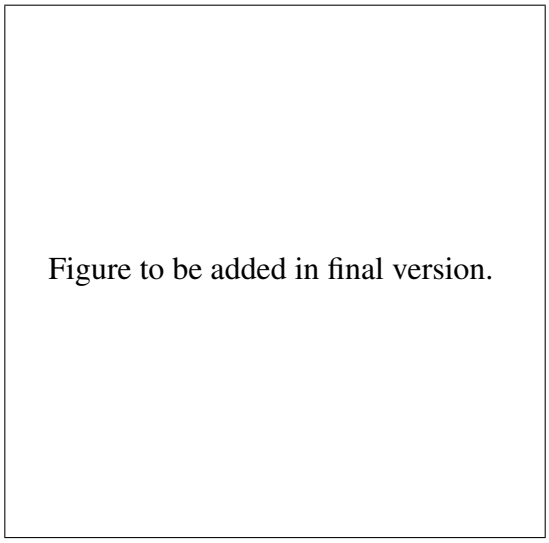


Figure 4.2: Small Truss.

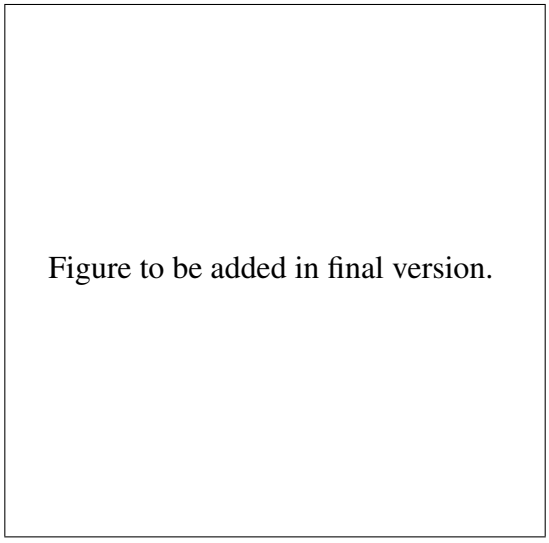


Figure 4.3: Large Truss example.

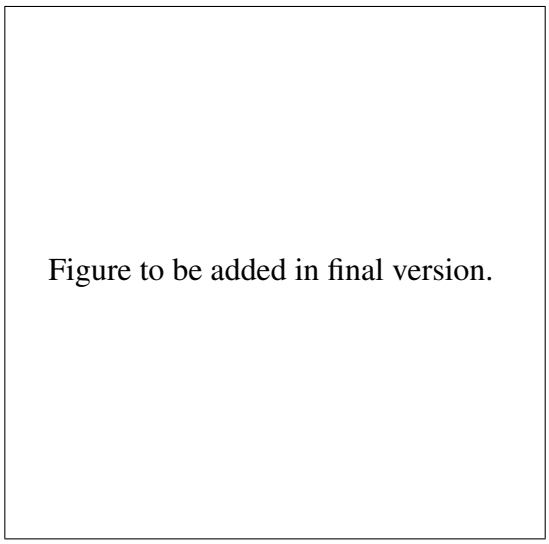


Figure 4.4: Example of Diversity.

4.6 Completeness

Having described the code used and seen some examples, one very naturally asks, can it be used to represent any truss, no matter how arcane or unwieldly. In other words, we would like to show that the mapping represented by the development of the organism, the transformation from genotype to phenotype, is onto.

In this task, we must consider both sides of the gene statement: the conditional and the expressive.

4.6.1 Conditional

We wish to show that any real value can be obtained using the conditional statements outlined above. The conditional statements serve two purposes. First, they serve as a regulatory mechanism to turn genes on or off and as such they play a role in enabling the code, but not in link definition. Second, the conditional statements compute a Θ value that is used

to determine the strength of experessive gene actions. As will be shown shortly, one of the key requirements on the Θ value is that it encompass the entire real line, $\Theta \in (-\infty, \infty)$.

The value of Θ depends on operations based on the concentrations of proteins present at the node location. By definition, the concentration of a protein must be $\in \Re$. So the positive real line of Θ can be reached simply by addopting the value of the concentration directly. To get the other half of the real line, the either the SUB or the NOT atom must be used. The SUB function subtracts the value of the concentration from the current value of Θ. If we assume that SUB is the only atom in a given expression, then the effect is to add the negative of the concentration. This gives the other half of the real number line, giving us the desired $\Theta \in \Re$ to take to the next section.

4.6.2 Expressive

We now turn our attention the Expressive portion of the gene representation. In this case, we wish to show that the growth rules are capable of producing any truss. We take this in multiple steps.

First, we restrict our space. In this case, we will set it to be any truss that has potential to be realized. The simplest requirement is to there be at minimum four nodes and three links. Anything less is not a real truss. For instance, three nodes with either two or three links creates a a mechanism that can rotate around two of the nodes.

Then, we say that to proove that any truss is possible if any link is possible. This part is rather trivial, as a truss is simply o collection of links put together in a coherent fashion.

Finally, we show that any any link is possible based on the Conditional discussion above

and the available rules. Consider four point, $p_1, p_2, p_3, p_4 \in \Re^3$ that are not coplanar. Then the goal is to connect p_1 to some arbitrary $t \in \Re$. The choice of using p_1 can be done without loss of generality, since all p_n are chosen and labelled arbitrarily. Since the point t is any point in $\Re$, creating any link to it is the same as creating any link in space. We begin by connecting it.

Consider the vectors $v_i, i \in 1, 2, 3$ from p_1 to each of the other p_n and normalize to define three directions. Since the original four points were non-coplanar, the three vectors all have different components and form a basis (though a not necessarily an orthogonal one). If we now iteratively apply the GRD rule which grows by Θ in the direction of the gradient of some protein. In the simple case, only one node produces each protein, so growing in the direction of the gradient is equivalent to growing in the direction of one of the other nodes or the direction v_i. One more note, since $\Theta \in (-\infty, \infty)$, the negative portion is corresponds to movement away from a node. Since the three v_i form a basis and GRD can be applied once (or more) to each, the point t can clearly be placed arbitrarily in $\Re^3$.

The final step is to ensure that any type of link can be created. This is fairly trivial to do, as the MAT atom selects any material (from the list of availbale materials) and the MOP, MOM, and RLX function grow the cross-sectional areas of the link can be set to anything.

Thus we have shown that any arbitrary link can be created. By repeatedly applying such a procedure, any arbitrary truss can be created, meeting the goal we started out with.

4.7 Performance Criteria

Before turning our attention to Evolutionary Algorithms and evolving the best rule sets, and in turn the best structure, we need to discuss what makes a good structure. The obvious answer is "one that won't fail in use." While this is clearly true, it is also too simplistic, as there are some important subtleties that are well know to any engineer. This section will discuss the aspects of performance that we are interested in as well as outline how they are evaluated in the simulation.

4.7.1 Mass

Since one of key areas of interest in the design of space structures, we care very much for the mass of our artifacts. Even if this were not a space based application, the mass still serves a useful measure: it tells you how much material you need, and therefore is a surrogate to cost.

The mass is defined quite simply as:

$$M = \sum_{i=0}^{N} M_i \tag{4.10}$$

where M is the mass of the system, M_i is the mass of link i and is defined by:

$$M_i = \sum_{i=0}^{N} A_i \rho_i L_i \tag{4.11}$$

where A_i is the cross-sectional area of the link, ρ_i is the density of the link, and L_i is the length of the link.

4.7.2 Strength and Stiffness

The structural evaluation of a truss is relatively simple, but computationally intensive. The evaluation of strength and stiffness are relatively similar and will be treated together as long as feasible. Here, we will present only the basics of the method. The details on the method and the validation of the results are presented in Appendix A.

For the sake of this text, strength of the structure is defined as the relative closeness to failure by yielding, commonly refered to as factor of safety:

$$\text{Strength} = \text{FOS} = \min_i \frac{\sigma_i}{\sigma_{i,yield}} \tag{4.12}$$

Likewise, the stiffness is defined as the lowest free-vibration frequency of the structure.

In order to make these two evaluations, we seek construct a model of the type:

$$\mathbb{M}\underline{\ddot{X}} + \mathbb{K}\underline{X} = \mathbf{F} \tag{4.13}$$

where $\mathbb{M}$ is the mass matrix, $\mathbb{K}$ is the stiffness matrix, $\underline{X}$ is the nodal displacement vector, and $\mathbf{F}$ is the applied force vector.

In order to do this, each of the n nodes in the truss is taken as the point of reference. The mass of each node is given as the sum of half of the mass of each link connected to it:

$$m_i = \sum_{j=0}^{k} m_j/2, i = 1..n \tag{4.14}$$

where m_i is the mass of each node i, m_j is the mass of each of the k links that connect to

node i. Then, the dynamics of each node can be written as:

$$m_i\underline{\ddot{x}_i} + \sum_{j=0}^{k} k_{ij}\left(\underline{x_i} - \underline{x_j}\right) = \mathbf{f_i}, i = 1..n \tag{4.15}$$

where k_ij is the spring constant between nodes i and j. This forms a set of three equations for each node. The set of $3n$ equations for all nodes form a set of linear equations and can be assembled into the model desired. This presents just a brief overview of assembly of the equations. The complete details are discussed in Appendix A.

From this simple model, we can find equations for both the strength and the stiffness. The strength can be found from solving the static portion ($\underline{\ddot{X}} \rightarrow \underline{0}$) giving the simple equation:

$$\mathbb{K}\underline{X} = \mathbf{F} \tag{4.16}$$

that can be solved by any method desired. In the case of this work, the solution is derived using the LaPack function `dgesv()`. (Details in Appendix A).

The frequency analysis can be done by solving the homogeneous part of Equation 4.13 ($\mathbf{F} \rightarrow \mathbf{0}$) giving the second order differential equation:

$$\mathbb{M}\underline{\ddot{X}} + \mathbb{K}\underline{X} = \mathbf{0} \tag{4.17}$$

which can also be solved by any popular method. For this work, the LaPack `dgeev()` function does the job quite well. Details on both of these solution methods are in Appendix A.

4.7.2.1 Strength specific details

For generating the strength data, we choose to do a static analysis, and therefore ignore the derivatives in Equation 4.13:

$$\mathbb{K}\underline{X} = \mathbf{F} \tag{4.18}$$

which leads to a simple equation in $3N$ variables (is in the number of nodes) that can be solved for the displacements. But in order to this, we must first apply the external constraints.

In order to ground a node, we set the entire row of $\mathbb{K}$ that corresponds to that node to 0 except for the the diagonal term, which is set to 1. The corresponding force is also set to 0. Other constraints are possible and follow a similar method.

From the displacement, we can easily calculate the strains $\underline{\epsilon}$ of each link, which gives in the usual fashion the stress in each link $\underline{\sigma}$ that we were looking for.

4.7.2.2 Stiffness specific details

To find the free vibration frequencies of the structure, we set the applied forces to zero yielding the ordinary differential equation:

$$\mathbb{M}\ddot{\underline{X}} + \mathbb{K}\underline{X} = \mathbf{0} \tag{4.19}$$

From here one can pre-multiply by $\mathbb{M}$ to generate a simpler version:

$$\ddot{\underline{X}} = \mathbb{W}\underline{X} = \mathbf{0} \tag{4.20}$$

where: $\mathbb{W} = \mathbb{M}^{-1}\mathbb{K}$. Again, we apply the boundary conditions to the matrix $\mathbb{W}$ by setting each constrained row to 0 as before.

Finally, calculating the square root of the eigenvalues of $\mathbb{W}$ gives the natural frequencies of the structure. For this task, we keep the lowest frequency as our measure of performace, but one could imagine further processing at this point.

4.7.3 Geometric Constraints

There are a variety of possible measures that fit into this category, but we will focus on only a few of them that are most critical to the applications that we care about.

The first is to ensure that we have actually grown a structure. That is, there are sufficient links that the structure is statically determinant. This is a reasonably trivial calculation, as you simply count all the connections at each node. Then, any node that has less than three connections causes this test to fail. Despite its inherent simplicity, this test is vital. The structural evaluators discussed above all require a well conditioned structure; therefore, this test ensures that not only do we get what we want, but also that we don't crash the structural evaluation. Note that nodes with no connections are simply ignored, as they would not contribute anything of use.

The second criterium is to ensure that all the objects (nodes) that need to be supported are in fact supported. Again, this is a fairly simple, but important test. In this case, you look at all of the load nodes and ensure that at least one link connects to it; one link is all that is required since we have already performed test 1 above.

The final criterium of interest here is to keep links and nodes out of keep-out areas.

That is, if we have alocated a certain space to some other purpose, then we don't want a link crossing through it.

4.7.4 Robustness

Robustness can mean many things to many designers. See Pahl and Beitz [69] for one discussion of robust design. For the present work, it means that the design has a tolerance to small variations from the actual design point. These may be manufacturing tolerances, where one link is made smaller (or larger) that was planned. It could also show up as an environmental change where the structure is attached to the side of a hill instead of flat ground, thereby changing the gravity vector and altering performance. In short, there are many reasons why variations could be introduced. By saying that the design is robust, we are saying that it will still perform adequately.

Evaluating robustness can be quite a challenge. To demonstrate full robustness, one would have to test a large assortment of candidate errors and then aggregate the various results into a single robustness score (the MIN, or worst case operator, seems like the right aggregation scheme here). However, this can be computationally expensive, as many cases need to be run and we would like our program to run in a reasonable time frame.

However, there is one trick we can use. Knowing that we are using Evolutionary Algorithms, we will already be running many cases and we can use that to our advantge. At each generation, each individual will be evaluated only once, but the parameters will all be randomly varied according to user specifications. The basic application looks like this:

$$P_{\text{eff},i} = P_i + X\xi_i \tag{4.21}$$

where $P_{\text{eff},i}$ is the effective value of parameter i, P_i is the nominal value of the parameter, X is a gaussian random number, and ξ_i is the standard deviation assigned to the parameter. In this method, the designer needs only specify the variability of the various parameters by specifying a set of ξ_i variables to use. These would likely be assigned in bulk based on the type of parameter in use, *e.g.* all forces would be varied according to a single ξ_i.

The advantage to this method is that each individual is evaluated only once every generation. However, for a genome to survive multiple generations, and therefore be considered 'good,' it will have been evaluated in multiple variations, thus leading to robustness with a low computational cost.

4.7.5 Other measures

While we have highlighted several key parameters of interest to our designs, they do not form a unique set. There are many other aspects that could be of interest. For instance, if we were designing a bridge, we might care about the corrosive tolerance in a salt-spray environment or we may care about the dielectric potential created by joining two dissimilar metals at a single interface. In short, there are nearly as many performance criteria as there are designs.

Evolutionary Algorithms are remarkable that they can handle any of these. Essentially, if we can create a quantifiable measure, then artificial evolution will design to it. There are a few subtleties, but for the most part this is true.

4.8 Genetic Algorithms

Having discussed the growth and development aspects of new method, we now turn to the rather mundane Evolutionary Algorithm. Figure 4.8 shows the algorithm in its utter simplicity. Having discussed the basics of Evolution Algorithms in Chapter 2, this section will not revisit all of those aspects here. This section will focus on the details of the algorithm as needed for this application. The first topic of discussion will be the variational operators, mutation and crossover. Second, we'll discuss the fitness evaluation. Then we'll discuss the selection scheme, both for which parents create the offspring and which individuals progress to the next generation. Lastly, we'll discuss some of the practical implications of running this taxing code efficiently on a distributed memory computational platform.

4.8.1 Variation Operators

Variation in this scheme takes the usual forms: asexual mutation and sexual recombination (crossover). While based on the standard forms, in this particual application, they are applied a different way. The next two sections will each detail one of the variation methods.

4.8.1.1 Mutation

The mutation function serves to introduce random changes to the genome, and the population at large, to explore new areas of the design space. The basics of the mutation operator is the same as any other, but does introduce some unusual aspects. The mutation process is shown in Figure 4.8.1.1 and detailed below.

The first step in mutation is genome duplication. With a very small probability, the

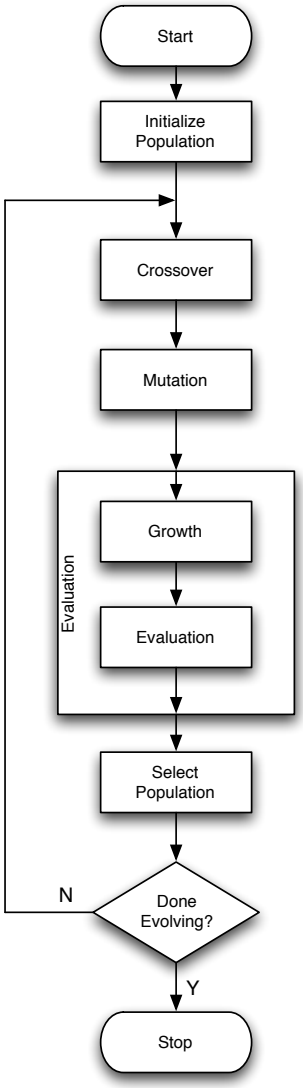


Figure 4.5: Modified Evolutionary Algorithm with growth and development

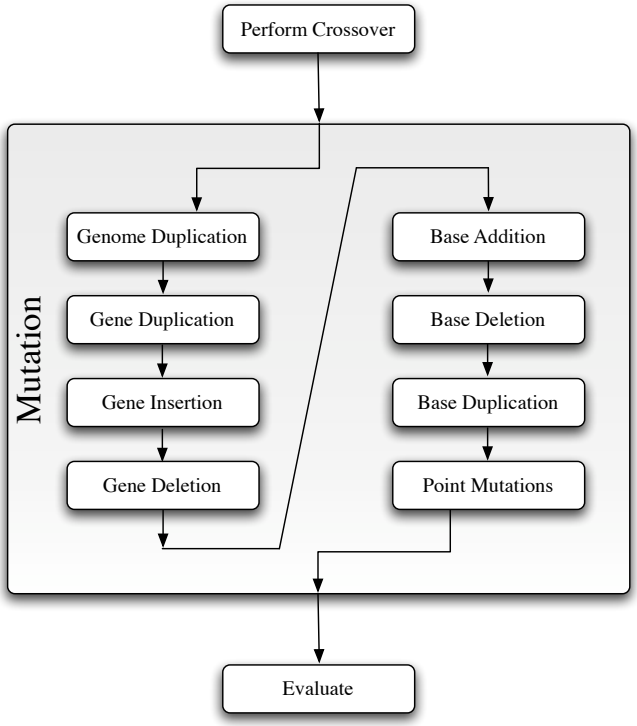


Figure 4.6: Flowchart of mutation options.

entire genome could be duplicated. There is some evidence for this happening in the biological world. [85] This provides a way for a good block of genetic code to be re-used somewhere else. For example, looking at the structures of the hand could be an old copy of the foot that has continued to evolve, selecting for slightly different aspects. [26]

The second step in mutation is gene duplication. This works just like genome duplication, but only one gene at time. There are no restrictions on the number of genes that can be duplicated in one step, but the probability of a single gene duplication event is usually very low and the odds that more than one happens per individual is miniscule.

The next steps are gene insertion and gene deletion, in order. Gene insertion helps to grow the gene, although it does not appear nearly as useful as gene duplication. The reason seems to be that with gene insertion, the gene starts empty and needs to grow into

```
Parent 1:  [P1G1C1][P1G1C2]::::::[P1G1E1][P1G1E2]
           [P1G2C1][P1G2C2]::::::[P1G2E1][P1G2E2]

Parent 2:  [P2G1C1][P2G1C2]::::::[P2G1E1][P2G1E2]
           [P2G2C1][P2G2C2]::::::[P2G2E1][P2G2E2]


Child:     [P2G1C1][P2G1C2]::::::[P2G1E1][P2G1E2]  (from Parent 2)
           [P1G2C1][P1G2C2]::::::[P1G2E1][P1G2E2]  (from Parent 1)
```

Figure 4.7: An example of crossover. The atoms are generalize so that Pn indicates it is from Parent n, Gm indicates it is from Gene m, and Cl and El indicates it is either conditional or expressive rule l.

something useful, whereas in gene duplication the gene comes ready made with something useful. Gene deletion is not too critical, but can serve to make the genome more compact and efficient, especially in later evolution.

Then, the same operations are repeated at the base or allele level. The mutation operator can add a new, random allele at any point in the gene string. Likewise, any allele can be duplicated or deleted.

Lastly, the mutation operator works through all remaining point mutations. This includes an allele converting from one type to another, for instance an `[NNYe1]` could become an `[ADDe1]` block. Finally, the internal aspects of each rule is allowed to vary, such as which protein it is triggered from (`e` vs. `i`) or which type of node to create (in the case of `[GRD]`).

4.8.1.2 Crossover

Crossover in the scheme described becomes very easy. Crossover between two parents can only happen at logical break points. The most logical break point is between genes, so

that a parent can only share a complete gene with its offspring. In addition to being easy, this is logical, a complete gene should represent some behavior, sense the environment and take an appropriate action. This very much fits with the goals stated above to develop modularity (in the genotype, for this case).

Because there is no selection based on species (most artificial genomes don't have the same number of genes or similar patterns in their genes) we don't impose a requirement on the child genome length. For each gene in the child, either parent has an equal chance of providing the gene. This can produce some strange degenerate cases. First, the child could be an exact copy of one parent or the other. Second, the child could contain an exact copy of both parents. And lastly, the child could contain an empty genome. None of these are very serious problems. The first case does not harm you, but it doesn't help either. The second case could be interesting, though unlikely, especially if the parent genomes are very similar. The last case is useless, but easily weeded out and ignored.

There are other options as well, though they seem less attractive. One could imagine breaking a gene and taking the conditional portion from one parent and the expressive portion from the other parent. This has some merit, as it does keep some of the gene module together. Another potential generalization, would be split the gene in between various atoms on the gene string, futher violating the modularity concept. Breaking the gene at any smaller resolution would not be feasible as an alternative.

One final possibility that was considered, but never implemented would be to use more that two parents in cross-over. Since the child can take any of its genes directly from one parent or another, it would be an easy matter to work in as many parents as desired. The benefit is not clear, but could prove an interesting experiment. This is an area that has been

looked at briefly in the literature [36, 35, 34, 38, 6, 13], but not very well studied.

4.8.2 Fitness Function

Above, we discussed all of the performance measures used to determine the performance metrics of a truss and provide a measure of by which to compare trusses. In order to arrive at a proper fitness function, one that can be used in choosing parents as well as in selection, there remains one last step to be performed; that step is to combine or aggregate all of the properties into a single performance metric, or fitness, that can be easily compared.

To the casual observer, this could be easily accomplished by classic aggregation methods, such as sum, product, or even root-sum-squared (RSS). However all of these methods suffer from severe deficiencies. One of the biggest is that they have no sense of scale: vibrational frequencies, which are typically in the 10-50 Hz range for large structures would get swamped by masses that could be in the thousands. Another problem is compensation: a drop of one hertz can be offset by a gain of a kg of mass, which is clearly not appropriate.

The Method of Imprecission (M_0I) [77] presents a solution to this task. The method works by asking the user to assign goal value, or preferences, to each of the performance parameters. This is a value on $[0, 1]$ where 1 is the best a parametr could reach and 0 is the worst. In all cases considered in this work, the preference functions for all parameters had the form:

$$\mu_X(X) = \begin{cases} 0 & \text{for } X < \min(X) \\ F(X) \in [0,1] & \text{for } X \in [\min(X), \max(X)] \\ 1 & \text{for } X > \max(X) \end{cases} \tag{4.22}$$

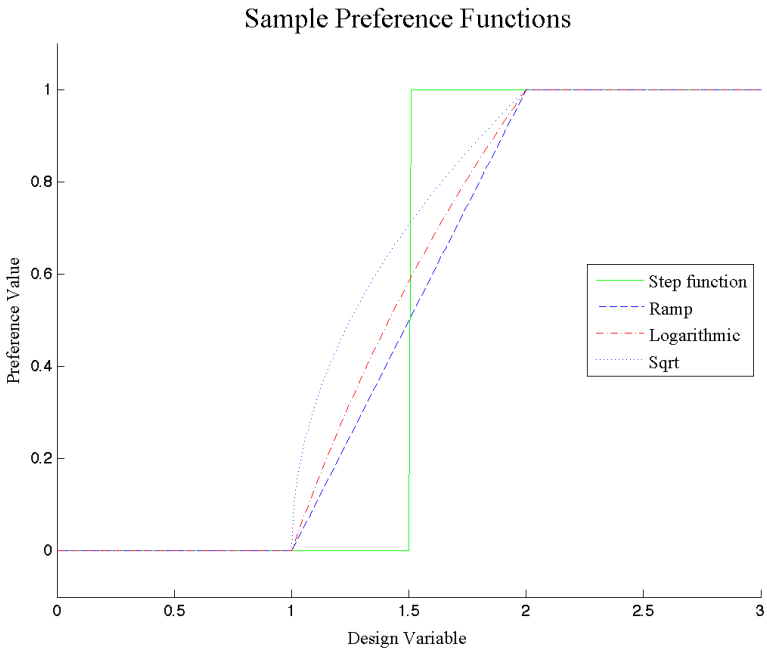


Figure 4.8: Example of a preference function

where X is any of the performance variables and $F(X)$ is some function that transitions between 0 and 1. The terms *max* and *min* in this case refer to the max and min allowables. The resulting preference function is shown in Figure 4.8. In some cases, like mass, the function is actually flipped around, so that low mass earns more preference than high mass. This can be applied to each of the parameters of interest.

Using the the mass calculated as described above, the preference for the mass is easily calculated:

$$\mu_M(M) = \begin{cases} 1 & \text{for } M < \min(M) \\ F(M) \in [0,1] & \text{for } M \in [\min(M), \max(M)] \\ 0 & \text{for } M > \max(M) \end{cases} \tag{4.23}$$

Essentially, this says that above a certain cutoff mass, the structure is considered com-

pletely invalid. Below another cutoff, it is considered so good that any lowering of the mass does not provide any useful increase in quality of the system. Anything in between gets a proportionate score between the two extremes.

Likewise, the method can be applied to the stiffness. Of the frequencies calculated above, the lowest frequency is used. For convenience, this lowest frequency will be represented by $\overline{f}$. This value can then be used to determine the preference of a particular design:

$$\mu_{\overline{f}}(\overline{f}) = \begin{cases} 0 & \text{for } \overline{f} < \min(\overline{f}) \\ F(S) \in [0,1] & \text{for } \overline{f} \in [\min(\overline{f}), \max(\overline{f})] \\ 1 & \text{for } \overline{f} > \max(\overline{f}) \end{cases} \tag{4.24}$$

Like with the mass score, the frequency (stiffness) score works in the sameway, below a critical freqency, the design is considered infeasible and gets a 0 preference. Above this frequency, it increases in prefence until it reaches a point where extra stiffness is of no additional use.

Finally, the strength preference can be calculated. Again, only the worst factor of safety is used. As defined above, this means that the lower the better. This is represented as $\overline{S}$. A factor of safety of 0 would be no stress and 1 would represent an element that is stressed to the yield point. Anything higher than that is an element that is critically stressed. The preference can be applied as follows:

$$\mu_{\overline{S}}(\overline{S}) = \begin{cases} 1 & \text{for } \overline{S} < \min(\overline{S}) \\ F(X) \in [0,1] & \text{for } \overline{S} \in [\min(\overline{S}), \max(\overline{S})] \\ 0 & \text{for } \overline{S} > \max(\overline{S}) \end{cases} \tag{4.25}$$

As before, if the factor of safety is below a minimum value, it is considered as good as it gets and receives a preference of 1. Above this value it begins to get progressively worse.

We also created one additional parameter to encourage early growth in a difficult environment. This parameter is consists of applying increasingly difficult tests for growths that do not qualify as a structure and takes the form:

$$\mu_t = \begin{cases} 0 & \text{for failing all tests.} \\ 0.25 & \text{for connecting enough base nodes} \\ 0.5 & \text{for connecting load nodes} \\ 0.75 & \text{for being a structure} \\ 1 & \text{for having a proper structure} \end{cases} \tag{4.26}$$

The first test checks that there are at least four nodes and three elements in the truss, as anything less has no possibility of being useful. The next test counts the number of constraints imposed on the constraint nodes. There must be at least six degrees of freedom constrained for the structural analysis to make sense. If there are enough constraints, the test parameter is increased by 0.25. If there are not enough constraints, the 0.25 is scaled in proportion to the number of constraints that actually exist. The evaluation also ends at this point, as there is no point in continuing.

If that test passes, the next step is to ensure that all load nodes are contained in the truss. This ensures that nodes of interest to the designer are included in the analysis. Once again, this test contributes 0.25 to fitness, failing the test conrtibutes an amount proportional to the the number of loads that are connected.

Finally, there is a test for properness of the structure. The first part tests that all nodes except constraint nodes have at least three elements connecting to it. While not perfect, this test is quick and weeds out most bad structures. The second half of the test verifies that the elements are valid, for instance that they have a positive cross-sectional area. Each step of this test contributes a 0.25 increase to the test preference.

Each of these preference functions can then be aggregated into a final, aggregated preference for the individual using the aggregation function defined by [77]. However, in this case, the aggregation is split into two functions. The reasons will be explained after the function is discussed.

$$\mu_{agg1} = \left(\frac{w_m \mu_m^{s_1} + w_\sigma \mu_\sigma^{s_1} + w_s \mu_s^{s_1}}{w_m + w_\sigma + w_s} \right)^{1/s_1} \tag{4.27}$$

$$\mu_{agg} = \left(\frac{\mu_{agg1} + w_t \mu_t^{s_2}}{1 + w_t} \right)^{1/s_2} \tag{4.28}$$

where μ_agg is the aggregated fitness, w_i are the relative weights for each parameter, and s is the degree of compensation [77]. Variations in these parameters lead to different Pareto Optimal solutions. The parameters s_1 and s_2 determines how much each parameter can compensate for another. For example: if $s \to -\infty$ corresponds to the minimum function (no compensation) and $s = 2$ corresponds to a weighted RMS aggregation.

When aggregating design parameters, one of the important considerations is what is commonly called *annihilation*. This means that if one of the parameters is completely unsatisfactory, then the aggregate design also has to be completely unsatisfactory. This makes good intuitive sense: consider a structure that is so stressed that it is predicted to fail upon construction. In such a case it makes no difference how light weight the structure is,

it will not work. When $s < 0$, this principal of annihilation is met.

However, for use in Evolutionary Algorithms this presents a problem. In EAs, one of the guiding principals is that there should be a selective pressure to guide the algorithm to a good solution. This means rewarding even a bad solution with some credit, even if it is still not enough to make a feasible structure. This is the reason that the fitness evaluation was split. The first part represents purely design parameters that should obey the principal of annihilation and in for this s_1 should be set to a value that is less than zero. The second half has to do with the quality of the truss, essentially topology search. For this, difficult aspect, every benefit should be allowed and for this s_2 should be greater than zero. In addition, the user should make sure to choose maximum and minimum allowable values in the preference functions that cover a wide base, so that even marginally valid designs can be considered and allowed to evolve further.

By converting all of the performance metrics into so-called user preferences, the issue of scaling is removed. By using the separate weighting factors, the relative importance is improved. This meets all of our desires.

4.8.3 Selection

Selection, as used in this work, can mean two similar but different things: the selection of which individuals go on to the next generation (Generational Selection) and the selection of which individuals to use as parents(Parent Selection). There are many options for both of these available in the literature and some of them have been implemented in the growth evolution.

For Generational Selection, two algorithms were incorporated:

Steady State For a population size P, only P offspring are created and only the offspring proceed to the next generation.

Elitist For a population size$P$, nP offspring can be created (n) is an integer. The best P individuals proceed to the next generation.

The most commonly used version for this work is the Elitist selection. Although these are the only ones implemented at this time, there is nothing that precludes any of the other algorithms mention in Section 2.4.1.1 and indeed, implementing some of those could be advantageous. Of particular interest is Roulette Selection and Tournament Selection.

Parent Selection, likewise was implemented with couple of different algorithms:

Random Generate a uniform random number for each parent. Though, not the best idea, because Generational Selection is used to decide what individual lives to the next generation, this may not be a bad thing.

Rank Using an exponential random number, select the parent based on their rank in a list sorted by fitness.

Direct Parents are assigned in sequential order based on fitness. (The best and second best mate, the third and fourth best mate, etc.)

Again, Section 2.4.1.3 details some of the other methods available and which could be implemented. The most commonly used version for this work is the Rank selection.

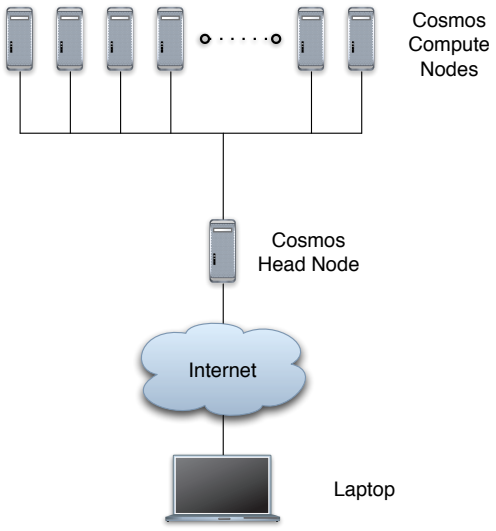


Figure 4.9: The client server architecture of the parallel EA.

4.8.4 Parallel Algorithm

In order to expedite the evolutionary runs, a parallel architecture was employed. This section will provide a brief overview of the architecture with psuedo-code. Detailed snippets of code can be found in Appendix C. Figure 4.8.4 shows the overall architecture of the system, with a *master* node to manage all of the population and several *clients* to do the detailed computations.

The server is responsible for all of the management tasks, this includes input and output functions, as well as maintaining the population, performing mutation and crossover, and instructing the clients on what task to work on. The clients on the other hand only evaluate each individual that the server hands them. This includes growing the structure from the base truss and the genome, as well as doing the performance evaluation.

Because it is simpler, we'll dicuss the client code first, shown in Listing 4.1 The command notDone is a parameter passed from the server indicating that the run is over and

Listing 4.1: Client pseudo-code.

```
while (notDone) {
  Receive_Genome( )
  Grow_Structure( )
  Evaluate_Structure( )
  Send_Fitness( )
}
```

that the clients should shut down. The functions Grow_Structure and Evaluate_Structure really are the meat of program and cause a genome to be complete evaluated, as described above. The function Receive_Genome uses MPI [1, 2] to receive a genome from the server. Likewise, function Send_Fitness uses MPI to send the fitness back to the server. Both of these functions will be discussed after the server code.

The server side code, is a match for the client, but contains more complications in order to help balance the work load accross the multiple compute nodes. Listing 4.2 lists the pseudo-code for the server. Again, the detailed code is in Appendix C. The server code is

Listing 4.2: Server pseudo-code.

```
while (notDone) {
  for (each clients) // Part 1
    Send_Genome( )
  while (jobs_remain)     // Part 2
    Receive_Fitness( )
    Send_Genome( )
  while (jobs_still_active) // Part 3
    Recive_Fitness( )
}
```

separated into three parts that are very similar. Part 1 simply says to send one evaluation job to each client. In Part 2, the server waits for a client to finish an evaluation, receives that result, and sends a new evaluation to the client. Part 2 ends when there are no more jobs left

in the queue. Part 3 waits and collects the remaining results. Afterwards, it returns to the rest of the program to do all of the EA operations. Once again, the functions Send_Genome and Receive_Fitness do exactly what the sound like and will be discussed shortly.

The reason this seeming complex is necessary is balance the load on the nodes. In case there is one evaluation that takes significantly longer, that node can only that job while the others will take up the slack. However, this system is not perfect. If the last job sent out is the long one, all of the compute nodes, as well as the server, will have to wait until the long job is done. In addition, there needs to be significantly more individuals than nodes for this system to work well. Although no effort was made to quantify it, or explore it in detail, experience with running the system indicates that the ratio individuals to nodes needs to be at least 5 to get a descent load balancing.

Before moving on, we present the pseudo-code for the four MPI functions that make up the heart of the above routines. Listing 4.3 shows the details of the Send_Genome function and its companion Receive_Genome is shown in Listing 4.4. Likewise, Listing 4.5 and Listing 4.6 show the details of Send_Fitness and Receive_Fitness , respectively.

Listing 4.3: Pseudo-code for Send_Genome.

```
Function Send_Genome {
 Convert genome to transfer buffer
 Send the individual number
 Send buffer size
 Send the buffer
}
```

These are fairly simple functions in pseudo-code. In fact, MPI proves to be so powerful that the actual code is barely more complicated. In the send and receive genome functions, one of the critical steps is to convert the genome into a transfer buffer. This is a quirk of

Listing 4.4: Pseudo-code for Receive_Genome.

```
Function Receive_Genome {
 Receive the individual number
 Receive the buffer size
 Receive the buffer
 Convert the buffer to useful format
}
```

Listing 4.5: Pseudo-code for Send_Fitness.

```
Function Send_Fitness {
 Send the individual number
 Send the fitness
}
```

MPI that the buffer that gets sent must be a contiguous block of memory. To resolve this issue, the genome is converted into its string representation into a standard **char**$*$ array. Also, because the genome, and therefore the buffer, could vary widely in size, that size must be communicated to the client so that it can allocate the proper memory block. The last catch to these functions is that the identity of the individual must be maintained so that when the results are returned, they can be placed in the correct spot.

4.9 Summary

This chapter has described all of the critical aspects of the evolution of indirectly encoded discrete structures. It began by discussing the virtual environment in which the structures a placed to do their growth. Then, it went on to discuss the basic building blocks of the genome that represent the structures. Finally, we discussed how that genome is integrated into an Evolutionary Algorithm and the parameters that control that evolution.

Listing 4.6: Pseudo-code for Receive_Fitness.

```
Function Receive_Fitness {
 Receive the individual number
 Receive the fitness
}
```

Chapter 5

Results

For ever complex problem there's a simple solution, and it's wrong.

– Umberto Eco, Foucault's Pendulum

5.1 Introduction

Having described the reasons that the current Evolutionary Algorithms do not work well, the natural inspirations to a new method, and the details of the proposed new method, indirect enconding, attention now falls on applying the technique to some examples. These will begin with some hand coded examples demonstrating the power of growth. Then, some evolved examples will be shown.

Figure 5.1: Starting position for simple growth example.

5.2 Hand Coded Examples

5.2.1 The simplest structure possible

The simplest conceivable, non-trivial growth simulation consists of one load node and one constraint node. The resulting structure, will quite clearly be infeasible, but it is a valid growth. The step up is shown in Figure 5.2.1. The nodes are spaced one meter apart.

The genome written for growing an element between these two nodes is shown in Listing 5.1. The matching growth sequence is shown in Figure 5.2.1. At this point, the genome is simple enough to understand, so we will take a look at exactly what this genome is saying:

Line 1: says that if the active node is a *Load2* type of node, produce external protein 1. This is essentially the load node announcing its position so that the other nodes can grow towards it.

Line 2: says that if the active node is a *Constraint1* type of node, and there are less than one link connected to the node, and there is some external protein 1, grow a proto-link in the direction of of the gradient of external protein 1. This is the actual growth of the proto-link from node 1 to node 2.

Line 3: says that if the active node is a *Constraint 1* type of node, then increase the cross-sectional area. In this case, the amount to grow is 1 mm^2, just to keep it simple.

Line 4: says to set the material properties of the proto-link to the material indicated by

Listing 5.1: Genome writen for the simple growth case.

```
[SUPl2]:::::::[PRDe1]
[SUPc1][NLL1][ANYe1][ADDe1]:::::::[GRDe1]
[SUPc1]:::::::[MOM]
[ANDe4]:::::::[MAT]
```

Figure 5.2: Movie sequence of simple growth example.

rounding he concentration of external protein 4 to the nearest integer. This line is essentially a band-aid to ensure that there is a real material set to the element and make a real truss. For this simple example, it is probably not needed.

As can be seen in Figure 5.2.1, the simulation starts pumping protein into the environment at $t = 1$. After a few time steps, the protein diffuses enough to trigger node 2 to begin growing towards node 1. Once, the proto-link gets near enough (in this case ¡ than 10 mm), the proto-link 'snaps' to node 2, creating an element (in the pictures, this is indicated by the proto-link changing color when the connection is made). At this point, node 1 stops doing anything useful, since the `[NLL1]` block shuts down that node after the connection is made.

5.2.2 A more complicated example

We now take a look at a more complicated example. In this case, there are four constraint nodes, with one load node. The set up is shown in Figure 5.2.2. In this case, the four base

Figure 5.3: Starting position for the pyramid growth example.

Listing 5.2: Genome writen for the pyramid growth case.

```
  [SUPl2 ]:::::: [ PRDe1 ]
2 [SUPc1 ][NLL1 ][ANYe1 ][ADDe1 ]:::::: [ GRDe1 ]
  [SUPc1 ]:::::: [MOM]
4 [ANDe4 ]:::::: [MAT]
```

Figure 5.4: Movie sequence of pyramid growth example.

nodes form a square one meter on a side and the load node is centered on the square, one meter above it.

The genome for this demonstration is shown in Listing 5.2. As it turns out, this is the same genome as was used above for the simple example. This is to show that even without evolutionary assistance, the idea of genome re-use seems fairly potent. The functionality of this genome, shown in Figure 5.2.2, works essentialy the same way as the simple example above, except that now there are four nodes growing towards the load node and they have to go 20% further.

5.2.3 Topology change

Finally, in the hand coded examples, we present an example that contains a topology change. This is a node that spawns where there was none, then proceeds to grow a new truss. The set up, shown in Figure 5.2.3, is essentially the same as in the pyramid example above. However, now each of the load nodes is given its own identity (C0-C3).

Figure 5.5: Starting position for the node spawning growth example.

Figure 5.6: Movie sequence of node spawning growth example.

The genome for this is shown in Listing 5.3. Because of the much more complex phenotype, the genotype is also much longer. Because it is so long, we won't go through it line by like above, but just comment on it. Much of the length comes from the fact that each of the nodes acts on its own. The long strings of `[MOM]` are to make the cross-sectional area of the links to grow faster and more realistically.

The are probably much better ways to accomplish some of these tasks, but this is a proof of concept before moving on to evolution, and not much work was done to optimize this hand coded genome.

5.3 Evolved Examples

After using hand coded examples to provide confidence in the method and, more importantly, the basic building blocks (rules) in the language, attention focused on evolving designs. The set up is the same as for the hand-coded examples: There is a load to be supported and there is a plane below to be mounted to.

The load to be supported is set to 500 N located at coordinate (0.0, 0.0, 1000.0).

5.4 Summary

Listing 5.3: Genome writen for the spawning growth case.

```
[ SUPL2 ] : : : : : : [ PRDE1 ]
[ SUPC0 ] [ NLL1 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPC1 ] [ NLL1 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPC2 ] [ NLL1 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPC3 ] [ NLL1 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPC0 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [
    MOM] [MOM] [MOM]
[ SUPC1 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [
    MOM] [MOM] [MOM]
[ SUPC2 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [
    MOM] [MOM] [MOM]
[ SUPC3 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [
    MOM] [MOM] [MOM]
[ ANDE4 ] : : : : : : [ MAT]
[ SUPC0 ] [ NLL1 ] [ LLT750 ] : : : : : : [ SPL0 ]
[ SUPC1 ] [ NLL1 ] [ LLT750 ] : : : : : : [ SPL1 ]
[ SUPC2 ] [ NLL1 ] [ LLT750 ] : : : : : : [ SPL2 ]
[ SUPC3 ] [ NLL1 ] [ LLT750 ] : : : : : : [ SPL3 ]
[ SUPf0 ] [ NLL2 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPf1 ] [ NLL2 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPf2 ] [ NLL2 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPf3 ] [ NLL2 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE1 ]
[ SUPf0 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM]
[ SUPf1 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM]
[ SUPf2 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM]
[ SUPf3 ] : : : : : : [ MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM] [MOM]
[ SUPf0 ] : : : : : : [ PRDE2 ]
[ SUPf1 ] : : : : : : [ PRDE3 ]
[ SUPf2 ] : : : : : : [ PRDE4 ]
[ SUPf3 ] : : : : : : [ PRDE5 ]
[ SUPf0 ] [ NLM1 ] [ NLL4 ] [ ANYE3 ] [ ADDE3 ] [ ADDE3 ] : : : : : : [ GRDE3 ]
[ SUPf1 ] [ NLM1 ] [ NLL4 ] [ ANYE4 ] [ ADDE4 ] [ ADDE4 ] : : : : : : [ GRDE4 ]
[ SUPf2 ] [ NLM1 ] [ NLL4 ] [ ANYE5 ] [ ADDE5 ] [ ADDE5 ] : : : : : : [ GRDE5 ]
[ SUPf3 ] [ NLM1 ] [ NLL4 ] [ ANYE2 ] [ ADDE2 ] [ ADDE2 ] : : : : : : [ GRDE2 ]
[ SUPf3 ] [ NLM3 ] [ NLL6 ] [ ANYE3 ] [ ADDE3 ] [ ADDE3 ] : : : : : : [ GRDE3 ]
[ SUPC0 ] [ NLL2 ] [ NLM0 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE3
    ]
[ SUPC1 ] [ NLL2 ] [ NLM0 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE4
    ]
[ SUPC2 ] [ NLL2 ] [ NLM0 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE5
    ]
[ SUPC3 ] [ NLL2 ] [ NLM0 ] [ ANYE1 ] [ ADDE1 ] [ ADDE1 ] [ ADDE1 ] : : : : : : [ GRDE2
    ]
```

Chapter 6

Conclusions

Evolution is a fact, even if it is not a complete theory.

– Daniel S. Fischer

Lecturing at Caltech 30 November 2006

6.1 Summary

Evolutionary Algorithms started with a simple concept: apply the ideas of natural evolution to the engineering design process. Several key parameters were established that were believed to be the fundamental keys to evolution. These are *transmission*, *selection*, and *variation*. Using these three principals a variety of related algorithms were developed to assist the design process.

These methods met with limited success, especially as the complexity of the design challenge increased. One explation for this limited capability, as proposed in this work, is the vast differences in the type of encoding used. The artificial Evolutionary Algorithms use a *Direct Encoding* in which the organims or individual is represented in its gene string by an exact description of the phenotype. Natural encoding (DNA) does not do this. Instead

it uses *Indirect Encoding* in which the the genotype is merely a set of instructions on how to build a phenotype.

This work has presented such an encoding for use in the design of discrete structures. The method of indirectly encoding a structure was detailed as were some demonstration examples to show feasibility of the technique. Much work yet remains to be done on applying the method to interesting and useful design tasks, but the method is sound and shows great promise to expand the capability of Evolutionary Algorithms.

6.2 Future Work

The work described in this thesis is very new and not found anywhere else. For this reason, this section on future work, could be longer than the entire thesis. But, for the sake of practicality, only brief look will be taken at a variety of different concepts for future work.

Usability There are many parameters that have cropped up in creating this process. The diffusion constants control how fast the growth proceeds. From experimentation, these seem to be determined by the size of the design domain. Some basic rules on how to select the proper constants would be useful. Likewise, in the evaluation there are many weights to be selected. It would be useful to provide some basic rules on how to select these to tune the simulation to yield proper results.

More interesting and realistic examples One of the reasons to use Evolutionary Algorithms is to tackle difficult design challenges. In developing this work, simple problems were used so that the processes at work could be understood. However, while

some benefit was visible on these simple problems, to really see the benefit of this method or Evolutionary Algorithms in general, requires a challenging problem that does not have a complicated solution. On a similar thought, a real world problem would be good, to show the practicality of the method.

Wild cards Currently, the encoding scheme requires exact matching on variable. For instance, in the block `[ADDe4]`, only the particular protein mentioned can be used. The ability to have a block like `[ADDe*]` where the concentration of any or every protein is added. Along this line, there could also be an 'internal register' in the cells so that a certain protein choice could be re-used, rather than having to evolve the same connection. For instance, in the current scheme a gene might look like: `[ANYe4]::::::[PRDe4]`, which produces protein `e4` if there isn't any currently there. A more useful form might be something of the form: `[REGe4][ANY(reg)]::::::[PRD` where a memory buffer (REGister) remembers a certain protein which can then be used by any other rule later in the gene. This is just one way such a mechanism could be implemented and might prove useful.

More evaluations In this work, the structure was evaluated solely on stress, stiffness, and mass. But there are many more options that could be done as well. This could be a thermal stability analysis or a thermal gradient analysis where different thermal environments are evaluated. The frequency analyses could be modified to consider primary modes in various axes. Various stress cases could be evaluated based on different loading situations. From a usability point of view, a mechanism for rapidly adding different evaluations would also be another important aspect.

Better diffusion The diffusion model used in this work is very crude. For the problems considered, it was not limiting, but in real design challenges, it could become an issue. One potential area is the inclusion of keep-out zones that the truss must avoid. Proteins diffusing around such objects might be one way that the growth process could 'sense' the object's presence and take evasive actions. This sort of behavior is not possible in the current diffusion model.

Links as cells Currently, when a proto-link converts into a link, it is a dead entity. That is, it can no longer be modified or changed. But if the links were considered a cell of its own, that could be remedied. Now, when a certain link has too much stress in it, it could respond, via a rule in the genome, by increasing the cross-sectional area.

Stopping condition When to step the growth and development process is a complicated choice. For this first stab at using development, the stopping condition was a time limit. After 200 time steps, the individual was considered fully mature and went on to the evaluation portion. However, this seems like too simple a solution. Some other method needs to be developed to create a meaningful maturity. One option is to monitor the individual and when it stops making meaningful changes, halt the process. Unfortunately, the odds of such a condition being reached are pretty small and may not be reliable. Another option might be to have some sort of resource utilization during the growth process and when all of the resources are used up, the growth is done (a resource other than time). Perhaps there is yet another way that has not been conceived yet.

Continuous evaluation In the current work, structural evaluation is done only at the end

of the growth and development process. This was done for the efficiency of the process and quick turnaround in testing. While this was a good goal, there may be some benefit to evaluating structural performance during the growth process. By evaluating during the growth, the cells could respond to stimuli, such as the stress in a link or the deflection of a node. This would be a significant increase in computational requirements, but could have a strong payoff.

Module identification One of the original goals of this project was to create an encoding to promote modularity, both in the genotype and phenotype. As this task concludes, the encoding has been created and demonstrated, but the modularity aspects have not been explored. It would be interesting to apply some module identification techniques, such as the work by Wang [84].

Extensions Finally, the method could be extended to other domains. The first possibility is to extend the domain to continuous structures. Some of this work has already started, although using a somewhat different technique [87, 89, 88].

Another possibility is to extend the domain to include mechanisms. This could be accomplished, relatively easily, by replacing a link with conceptual piston that changes the configuration of the truss. A rotary joint could also be implemented at certain joints.

Finally, a very interesting area of extension is that of configuration design. Instead of just designing the structure, the technique could also evolve the positions of the various items to be supported. This would require quite a bit of work, especially in defining the new rules to place these elements and in the fitness evaluation to make

sure that the evolved positions for elements made sense.

These are but a few of the myriad possibilities for extending and improving the method described in this thesis. It is hoped that these may provide some concrete ideas for someone to take some action on.

Appendix A

Structural Evaluation Validation

A.1 Method

This section will provide an outline of the the structural evaluation method used in this thesis. This is a more detailed examination of the method than was described in the main body of the text. While this will be a complete look at the method, the excruciating, gory details will can be found in the text by Connor[25].

The method described here is usually described as the "Displacement Method," although it does occassionally have other names. The process begins by developing the governing equations of a truss. As this is fairly well know process, it will go fast.

The first step is to define a truss. For the purposes of this document, a truss is defined as a set of prismatic members (called bars) that are connected to each other using frictionless spherical joints, placed only at their ends. The bars are assumed to have no mass and the combination of this fact and the frictionless pins, implies that the bar can only have a force along the bar itself, greatly simplifying the analysis. Due to this, the force can be represented as a scalar, assumed to be along the lines connecting the joint centers. Second, the domain is restricted to space trusses, i.e. only three dimensional structures will be

considered thereby setting $i = 3$.

Note that if there are j joints (nodes) and r imposed displacements (fixed or otherwise), there are then $3i - r$ unknown displacements to solve for. Likewise, there are m unknown member forces, and r reaction forces to solve for. In the work described in the main body of the text, the only important variables are the m member forces, but the others could be used as well if the analysis became more involved.

Following the convention established by Connor, the initial coordinates, displacement components, and components of the resultant external force at joint k are represented by $\overrightarrow{x_k}$, $\overrightarrow{u_k}$, and $\overrightarrow{p_k}$. Each is a three component vector. Then, position vector($\overrightarrow{\rho_k}$) for joint k in the deformed state can be represented by:

$$\overrightarrow{\rho_k} = \overrightarrow{r_k} + \overrightarrow{u_k} \tag{A.1}$$

Now, consider bar n, which connects node s to node k. Note that the assignment of the two ends is arbitrary, but must be consistent throughout the process. The direction from k to s is taken to be the positive direction of the bar. The length of the bar n is given by:

$$L_n^2 = \Delta\overrightarrow{r} \cdot \Delta\overrightarrow{r} = \sum_{j=1}^{3} (x_{sj} - x_{kj}) \tag{A.2}$$

and the direction cosine of the undeformed bar is:

$$\alpha_{nj} = \frac{1}{L_n} (\mathbf{x}_s - \mathbf{x}_k)^T \tag{A.3}$$

Considering the deformed state, $\Delta\overrightarrow{\rho} = \overrightarrow{\rho_s} - \overrightarrow{\rho_k}$ defines the length and direction cosines.

Then the deformed cosine matrix can be shown to have the coefficients:

$$\beta_{nj} = \frac{1}{L_n + e_n}\left(\Delta\overrightarrow{\rho}\cdot\overrightarrow{i}_j\right) \tag{A.4}$$

With some simplification, the row matrix β can be written as:

$$\beta_n = \frac{1}{1+\frac{e_n}{L_n}}\left[\alpha_n + \frac{1}{L_n}\left(\mathbf{u}_s - \mathbf{u}_k\right)^T\right] \tag{A.5}$$

To make life simpler, assume that for real materials $\frac{e_n}{L_n} \ll 0$, giving:

$$\beta_n \approx \alpha_n \tag{A.6}$$

With the preliminary work done, substitute (as Connor did) n_+ for s and n_- for k. Then, the equations developed so far are:

$$\begin{aligned}
L_n^2 &= (\mathbf{x}_{\mathbf{n}_+} - \mathbf{x}_{\mathbf{n}_-})^T(\mathbf{x}_{\mathbf{n}_+} - \mathbf{x}_{\mathbf{n}_-}) \\
\alpha_{\mathbf{n}} &= \frac{1}{L_n}((\mathbf{x}_{\mathbf{n}_+} - \mathbf{x}_{\mathbf{n}_-})^T) \\
e_n &= \gamma_n(\mathbf{u}_{\mathbf{n}_+} - \mathbf{u}_{\mathbf{n}_-}) \\
\gamma_n &= \alpha_{\mathbf{n}} + \frac{1}{2L_n}(\mathbf{u}_{\mathbf{n}_+} - \mathbf{u}_{\mathbf{n}_-})^T \\
\beta_n &= \alpha_{\mathbf{n}} + \frac{1}{L_n}(\mathbf{u}_{\mathbf{n}_+} - \mathbf{u}_{\mathbf{n}_-})^T
\end{aligned} \tag{A.7}$$

Now, to combine all of these bar equations using the connectivity of the truss. The connectivity of the truss is defined by a list of links giving the the starting node and ending

node for each link. For convenience of reference, assume that the bars in the list are number from 1 to m. In this fashion, construct the two matrices:

$$\begin{aligned} \mathbf{e} &= \{e_1, e_2, \ldots, e_m\} \\ \mathcal{U} &= \{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_j}, \} \end{aligned} \tag{A.8}$$

where m is the number of links and j is the number of nodes. Then the elongation of the links ($\mathbf{e}$) and the displacements of the nodes ($\mathcal{U}$) are related by:

$$\mathbf{e} = \mathcal{A}\mathcal{U} \tag{A.9}$$

where $\mathcal{A}$ is a mx$3j$ matrix made up only of the γ_n such that:

$$\begin{aligned} \mathcal{A}_{nn_+} &= +\gamma_n \\ \mathcal{A}_{nn_-} &= -\gamma_n \\ \mathcal{A}_{nl} &= \mathbf{0} \text{ when } l \neq n_+ \text{ or } n_- \end{aligned} \tag{A.10}$$

Next, define the matrix γ as a quasi-diagonal matrix:

$$\gamma = \begin{bmatrix} \gamma_1 & & & \\ & \gamma_2 & & \\ & & \ddots & \\ & & & \gamma_m \end{bmatrix} \tag{A.11}$$

Which helps in constructing $\mathcal{A}$ using the connectivity matrix $\mathbf{C}$ defined as:

$$\mathbf{C} = [\mathbf{C}_{kl}] \begin{cases} k = 1, 2, \ldots, m \\ \\ l = 1, 2, \ldots, j \end{cases}$$

$$\mathbf{C}_{nn_+} = +\mathbf{I}_i \tag{A.12}$$

$$\mathbf{C}_{nn_-} = -\mathbf{I}_i$$

$$\mathbf{C}_{nl} = \mathbf{0} \text{ when } l \neq n_+ \text{ or } n_-$$

and $\mathcal{A}$ takes the rather simple form:

$$\mathcal{A} = \gamma \mathbf{C} \tag{A.13}$$

Now it is time to apply some constitutive relationships. Since the bar are assumed to be two force members that only have pure tension or compression along the line between their two joints, leading to constant stress throughout the bar, and assuming linear materials (Hookean), the relationships are:

$$F = \sigma A$$

$$e = L\epsilon \tag{A.14}$$

$$e_0 = L\epsilon_0$$

and more specifically:

$$\begin{aligned} \sigma &= E(\epsilon - \epsilon_0) \\ F &= \frac{AE}{L}(e - e_0) = k(e - e_0) \\ e &= \frac{L}{AE}F + e_0 = fF + e_0 \end{aligned} \tag{A.15}$$

where f and k represent the usual spring flexibility and stiffness factors. These parameters can also be made into matrix equations:

$$\mathbf{F} = \{F_1 F_2 \cdots F_m\} \tag{A.16}$$

$$\mathbf{k} = \begin{bmatrix} k_1 & & & \\ & k_2 & & \\ & & \ddots & \\ & & & k_m \end{bmatrix} = \mathbf{f}^{-1} \tag{A.17}$$

and the equations become:

$$\mathbf{F} = \mathbf{k}(\mathbf{e} - \mathbf{e}_0) = \mathbf{F}_0 + \mathbf{k}\mathcal{AU} \tag{A.18}$$

and

$$\mathcal{AU} = \mathbf{e}_0 + \mathbf{fF} \tag{A.19}$$

Then, consider the equilibrium enquations at each node. The force in each bar was defined as $\overrightarrow{F}_n$. Since the bar is a two force member and using the definitions of positive,

the bar's contributions to each joint are:

$$\overrightarrow{F}_{nn_+} = -\overrightarrow{F}_n$$

$$\overrightarrow{F}_{nn_-} = +\overrightarrow{F}_n \tag{A.20}$$

$$\tag{A.21}$$

Then the resultant force, $\overrightarrow{p}_k$ at each node is:

$$\overrightarrow{p}_k = -\sum_{j_+=k} \overrightarrow{F}_{jj_+} - \sum_{l_+=k} \overrightarrow{F}_{ll_-} \tag{A.22}$$

which is the sum of the positive contributions and the sum of the negative contributions. As before, these $\overrightarrow{p}_k$ forces are assembled into a matrix:

$$\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_j\} \tag{A.23}$$

and the equations become:

$$\mathcal{P} = \mathcal{B}\mathbf{F} \tag{A.24}$$

where $\mathcal{B}$ is defined by:

$$\mathcal{B} = [\mathcal{B}_{lk}]$$

$$l = 1, 2, \ldots, j \text{ and } k = 1, 2, \ldots, m$$

$$\mathcal{B}_{n_+ n} = +\beta_n^T \tag{A.25}$$

$$\mathcal{B}_{n_- n} = -\beta_n^T$$

$$\mathcal{B}_{ln} = \mathbf{0} \text{ when } l \neq n_+ \text{ or } n_-$$

which can conveniently be defined by matrices developed above:

$$\mathcal{B} = (\beta \mathbf{C})^T \tag{A.26}$$

Now, displacement restraints are imposed. To do this, matices $\mathcal{U}$, $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{B}$ are rearranged such that the unknown joint displacents and prescribed joint displacements, $\mathbf{U}_1$ and $\mathbf{U}_2$, are separated and the prescribed and unknown node loads, $\mathbf{P}_1$ and $\mathbf{P}_2$, are also separated. All other matrices are rearranged to follow suit. Note that this works since the node and member assignments are essentially arbitrary. This looks like:

$$\mathbf{U} = \left\{ \frac{\mathbf{U}_1}{\mathbf{U}_2} \right\}$$

$$\mathbf{P} = \left\{ \frac{\mathbf{P}_1}{\mathbf{P}_2} \right\}$$

$$\mathbf{A} = \{\mathbf{A}_1 | \mathbf{A}_2\} \tag{A.27}$$

$$\mathbf{B} = \left\{ \frac{\mathbf{B}_1}{\mathbf{B}_2} \right\}$$

Then the imposed displacements and imposed forces are easy to apply. Note that there cannot simultaneously be an imposed constraint *and* an imposed force on any given node.

At this point, the basic equations for a truss have been developed. In their cleanest form, they are:

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{B}_1\mathbf{F} \\ \mathbf{P}_2 &= \mathbf{B}_2\mathbf{F} \\ \mathbf{F} &= \mathbf{F}_i + \mathbf{k}\mathbf{A}_1\mathbf{U}_1 \\ \mathbf{F}_i &= \mathbf{k}\left(-\mathbf{e}_0 + \mathbf{A}_2\mathbf{U}_2\right) \end{aligned} \tag{A.28}$$

Now, attention focuses on reducing these into a simple form ready for solving by a computer.

Substituting for $\mathbf{F}$,

$$\left(\mathbf{B_1}\mathbf{k}\mathbf{A_1}\right)\mathbf{U_1} = \mathbf{P_1} - \mathbf{B_1}\mathbf{F_i} \tag{A.29}$$

in the simplified, linear model, $\mathbf{k}$ is a constant and $\mathbf{A}_j = \mathbf{B}_j^T$, giving

$$\mathbf{K}_{11}\mathbf{U}_1 = \mathbf{P_1} - \mathbf{B_1}\mathbf{F_i} \tag{A.30}$$

where $\mathbf{K}_{11} = \mathbf{B_1}\mathbf{k}\mathbf{A_1} = \mathbf{A_1^T}\mathbf{k}\mathbf{A_1}$, which can be shown to be positive definite. Finally, these are the forces on the nodes, which can be solved on their own for a statics problem, but the goal is to have a dynamic problem. For this, the forces found, can be added to the

basic equation $f = ma$, where f is the forces found above and

$$\mathbf{a} = \frac{\partial^2}{\partial t^2}\mathbf{U}$$

This gives a system that is easily solved, but from a computational point of view, there is still more to do. $\mathbf{K}_{11}$ is generally a sparse matrix, and it would be a waste to do all of the matrix multiplications required. So the last step is a process for assembling the matrix without matrix math.

For this, let the bar stiffness be found from definitions above:

$$\mathbf{k}_n = k_n \beta_n^T \gamma_n \tag{A.31}$$

Using the the linear assumptions, $\beta_n = \gamma_n = \alpha_n$, Equation A.31 becomes:

$$\mathbf{k}_n = k_n \alpha_n^T \alpha_n \tag{A.32}$$

which is easily computed.

The node-displacement relationship then becomes:

$$\mathbf{p}_{n_+} = m_{n_+}\frac{\partial^2}{\partial t^2}\mathbf{u}_{n_+} - m_{n_-}\frac{\partial^2}{\partial t^2}\mathbf{u}_{n_-} + \mathbf{k}_n\mathbf{u}_{n_+} - \mathbf{k}_n\mathbf{u}_{n_-} + \beta_n^T F_{0,n} \tag{A.33}$$

If the entire set of node-displacement matrix equations are written out, they are:

$$\mathcal{P} = \mathcal{M}\frac{\partial^2}{\partial t^2}\mathcal{U} + \mathcal{K}\mathcal{U} + \mathbf{P}_o \tag{A.34}$$

Working iteratively, for each bar, the contribution to the matrices matrices are:

$\mathcal{K}$

$$\begin{aligned} &+\mathbf{k}_n \text{ in row } n_+, \text{ column } n_+ \\ &-\mathbf{k}_n \text{ in row } n_-, \text{ column } n_+ \\ &-\mathbf{k}_n \text{ in row } n_+, \text{ column } n_- \\ &+\mathbf{k}_n \text{ in row } n_-, \text{ column } n_- \end{aligned} \tag{A.35}$$

$\mathcal{P}_0$

$$\begin{aligned} &+F_{0,n}\beta_n^T \text{ in row } n_+ \\ &-F_{0,n}\beta_n^T \text{ in row } n_- \end{aligned} \tag{A.36}$$

$\mathcal{M}$

$$\begin{aligned} &\frac{m_n}{2}\mathbf{I}_{(3x3)} \text{ in row } n_+, \text{ column } n_+ \\ &\frac{m_n}{2}\mathbf{I}_{(3x3)} \text{ in row } n_-, \text{ column } n_- \end{aligned} \tag{A.37}$$

Finally, in this iterative scheme, restraints need to be applied since all bars and nodes were included in the creation of the matrices. For this work, only full restraint was used (i.e. a node is fixed in space). This can be relaxed using the technique described by Connor.

To apply the constraints there are two steps. The first is operations on the $\mathcal{K}$ matrix. For any node q that is fixed, set the off-diagonal elements of column q and row q to $\mathbf{0}$ and the

Table A.1: Selected material properties used in the validation analysis.

Name	ρ (kg/mm^3)	E (mN/mm^2)
Aluminum	2.711e-6	6.898e7
Steel	7.829e-6	2.0694e8

diagonal element to $\mathbf{I_{3x3}}$:

$$\begin{aligned} \mathcal{K}_{qt} &= \mathbf{0} \text{ for } l \neq q \\ \mathcal{K}_{qt} &= \mathbf{0} \text{ for } l \neq q \\ \mathcal{K}_{qq} &= \mathbf{I}_{(3x3)} \end{aligned} \tag{A.38}$$

and likewise for $\mathcal{P}_N$

$$\begin{aligned} \mathcal{P}_{N,l} &= \mathcal{P}_{N,l} - \mathcal{K}_{lq}\bar{\mathbf{u}}_q \\ \mathcal{P}_{N,q} &= \bar{\mathbf{u}}_q \\ l &\neq q \end{aligned} \tag{A.39}$$

where $\bar{\mathbf{u}}_q$ is the imposed displacement (often it is $\mathbf{0}$).

A.2 Validation

In order to ensure that valid designs were created by the method, the analysis method was tested with a several simple cases. The comparison values were generated using NASTRAN. The material properties used each test case were selected from the generic library provided with NASTRAN and are given is Table A.2.

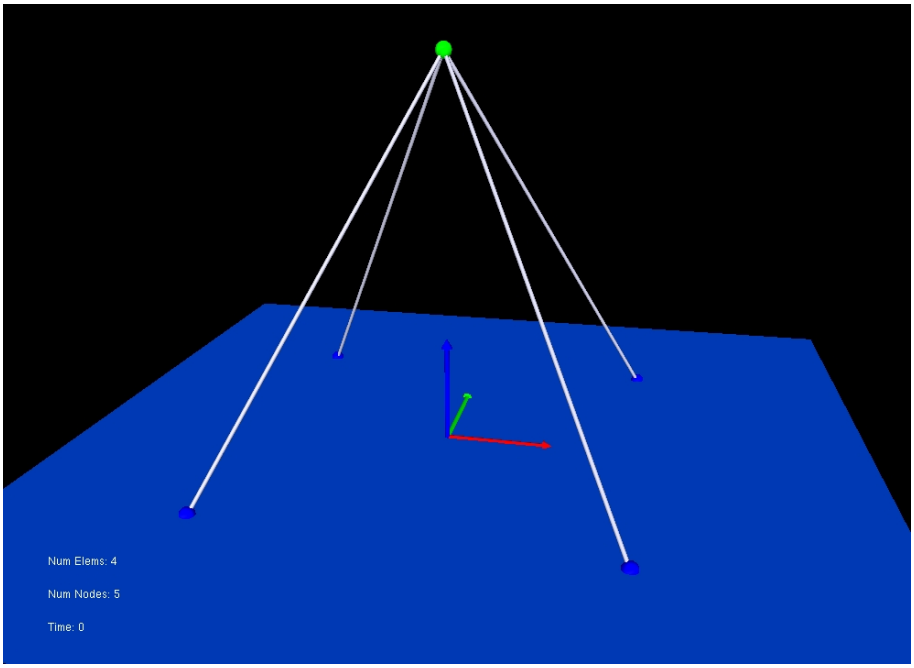


Figure A.1: Configuration of the pyramid test case.

A.2.1 Test Problem 1

The first test case, shown in Figure A.2.1, presents a simple trial of the evaluation routine, labeled the 'pyramid test case.' Table A.2.1 summarizes the geometry of the test case. In addition to this, nodes 1, 3, 4, and 5 are fixed constraints.

For the static load analysis, a load of 500 N is added at node 2 in the $+X$ direction.

Table A.2: Definition of the pyramid test case. (a) the list of nodes and (b) the list of elements.(units are mm or mm^2, as appropriate.)

Node ID	X	Y	Z
1	-500.0	-500.0	0.0
2	0.0	0.0	1000.0
3	500.0	500.0	0.0
4	500.0	-500.0	0.0
5	-500.0	500.0	0.0

(a)

Elem ID	End 1	End 2	Area	Material
1	2	4	75.0	Aluminum
2	2	3	75.0	Aluminum
3	2	5	75.0	Aluminum
4	2	1	75.0	Aluminum

(b)

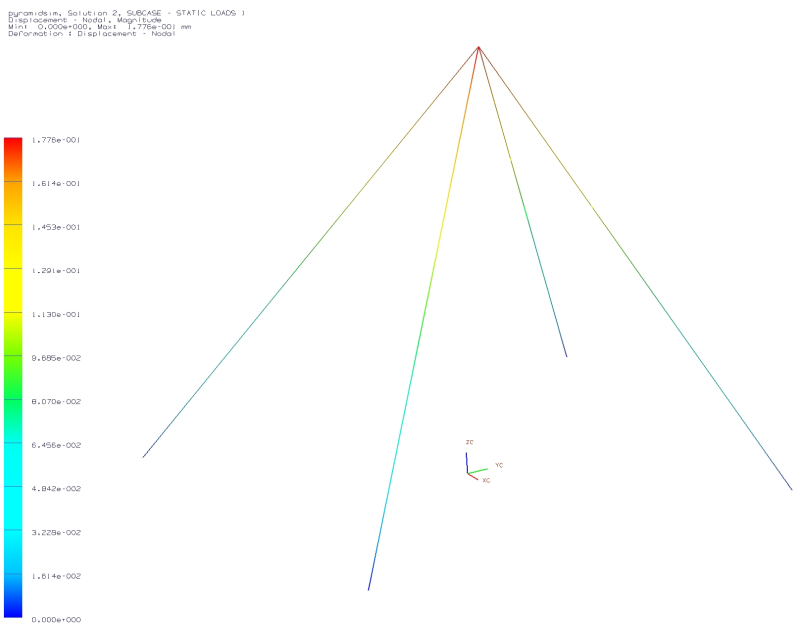


Figure A.2: NASTRAN calculated displacements to the pyramid test problem.

Table A.3: List of nodal displacement comparisons for the pyramid test problem. (Constraint nodes not listed.)(All units are in millimeters.)

Node	Nastran			Simulation		
ID	X	Y	Z	X	Y	Z
2	0.1776	0.0	0.0	0.1776	0	0

The results of the static analysis (as generated by NASTRAN) are shown in Figures A.2.1 and A.2.1. The results of the the output of the static analysis written for the truss evolution is shown in Tables A.2.1 and A.2.1. This table also demonstrates the close corrolation between the custom solver and the standard NASTRAN solution.

Table A.2.1 shows the results of the frequency analysis on the pyramid test case for both the custom solver and the NASTRAN tool. The table shows the first (and only) three modes of vibration of the pyramid test case and how closely the two solution methods match. It is likely that NASTRAN uses a very similar mechanism for its calculations and that on such a simple example, differences are not noticeable.

Figure A.3: NASTRAN calculated stress to the pyramid test problem.

Table A.4: List of element comparisons for the pyramid test problem. All units are in N/mm^2.

Elem ID	Nastran	Simulation
1	-4082.5	-4082
2	-4082.5	-4082
3	4082.5	4082
4	4082.5	4082

Table A.5: List of natural frequency comparisons for pyramid test problem.

Mode ID	Nastran	Simulation
1	378.5	378.5
2	378.5	378.5
3	756.9	756.9

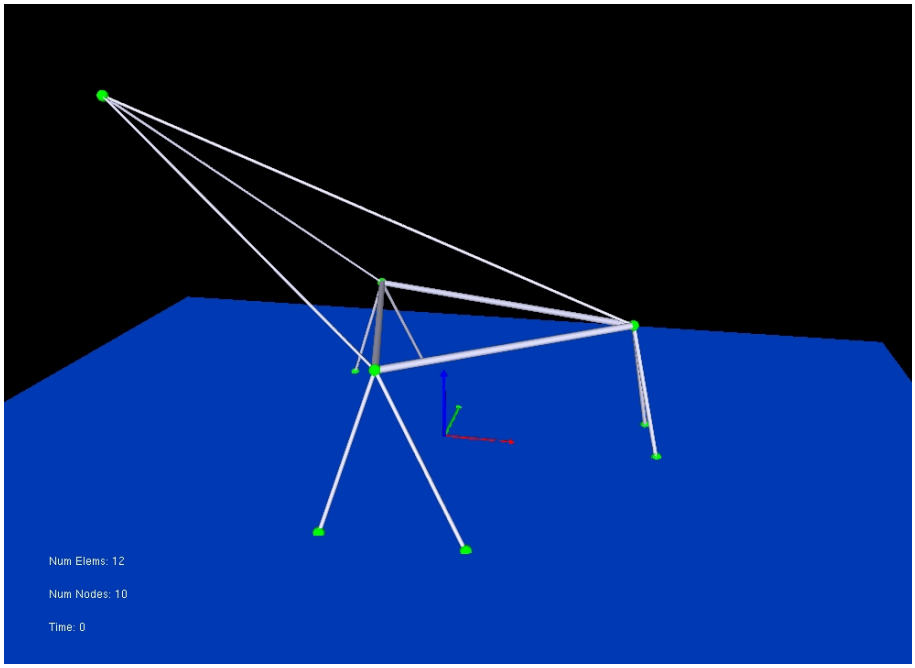


Figure A.4: Configuration of the crane test case.

A.2.2 Test Problem 2

To be certain that the solver was working properly, a second test case, labeled ‘crane,’ is run through both the custom solver and NASTRAN. The basic layout of the test case is shown in Figure A.2.2 and Table A.2.2 summarized the geometry of the case. In addition, nodes 12, 13, 14, 15, 16, and 17 are fixed. Lastly, node 18 must bear a 20,000 N force in the -Z direction.

The results of the static analysis on the crane problem (as generated by NASTRAN) are shown in Figures A.2.2 and A.2.2. The results of the the output of the static analysis written for the truss evolution is shown in Tables A.2.2 and A.2.2. This table also demonstrates the still close corrolation between the custom solver and the standard NASTRAN solution, even though there are now some differences.

Table A.2.2 shows the results of the frequency analysis on the crane test case for both

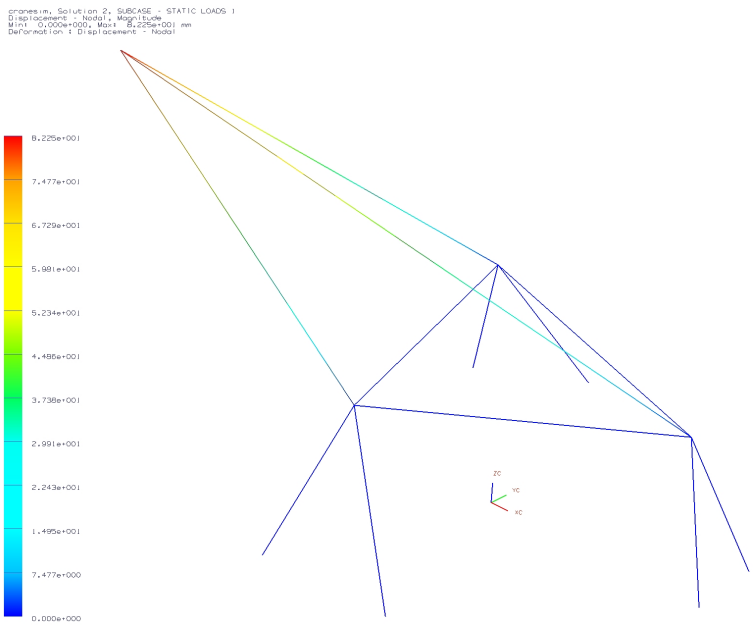


Figure A.5: NASTRAN calculated displacements to the crane test problem.

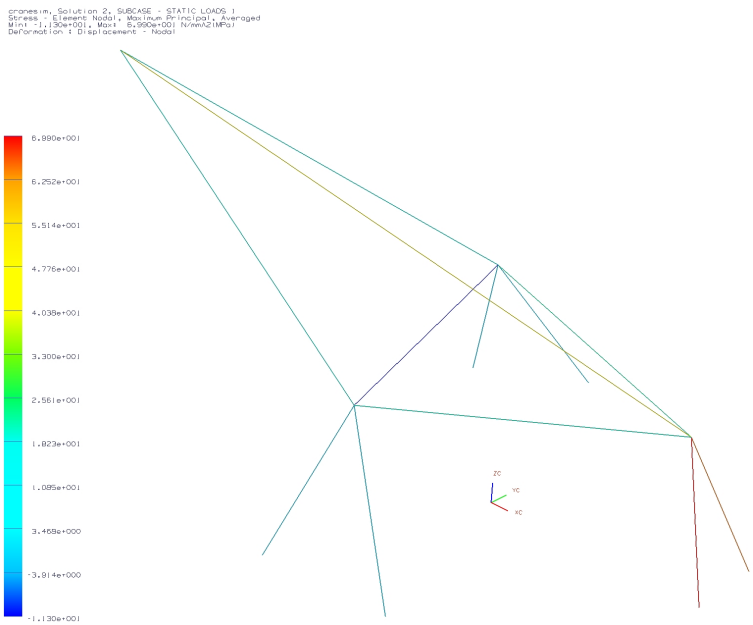


Figure A.6: NASTRAN calculated stress to the crane test problem.

Table A.6: Definition of the crane test case. (a) the list of nodes and (b) the list of elements.(units are mm or mm^2, as appropriate.)(IDs are assigned by Nastran, and maintained throughout analysis for simplicity.)

Node ID	X	Y	Z
2	698.3	39.4	500.0
3	-90.0	-550.0	500.0
4	-357.4	382.4	500.0
12	240.0	-732.9	0.0
13	815.1	-7.5	0.0
14	-561.9	553.2	0.0
15	772.9	259.0	0.0
16	-240.0	-732.9	0.0
17	-276.8	738.4	0.0
18	-1135.0	-231.4	1300.0

(a)

Elem ID	End 1	End 2	Area	Material
10	2	3	500.0	STEEL
11	4	3	500.0	STEEL
12	2	4	500.0	STEEL
22	2	13	200.0	Aluminum
23	2	15	200.0	Aluminum
24	4	17	150.0	Aluminum
25	4	14	150.0	Aluminum
26	3	12	150.0	Aluminum
27	3	16	150.0	Aluminum
29	4	18	80.0	Aluminum
30	3	18	80.0	Aluminum
33	2	18	95.0	Aluminum

(b)

Table A.7: List of nodal displacement comparisons for the crane test problem. All units are in millimeters.

Node	Nastran			Simulation		
ID	X	Y	Z	X	Y	Z
2	0.2601	0.1358	0.5061	0.2605	0.1359	0.5067
3	0.8143	-0.2783	-0.6178	0.8153	-0.2790	-0.6183
4	0.4319	-0.2561	-0.9716	0.4320	-0.2571	-0.9730
18	-44.14	-15.79	-67.57	-44.19	-15.81	-67.65

the custom solver and the NASTRAN tool. The table shows the first (and only) three modes of vibration of the pyramid test case and how closely the two solution methods match. Although, no documentation for NASTRAN has been found to support this, it appears that there are large deformations in the structure, giving a solution that would violate the small displacement assumptions made in the derivation and that NASTRAN compensates for this in someway.

Table A.8: List of element comparisons for the crane test problem. All units are in N/mm^2.

Elem ID	Nastran	Simulation
10	-41,184	-41,000
11	27,049	27,067
12	-53,033	-52,840
22	59,425	59,443
23	46,283	46,307
24	-77,310	-77,207
25	-76,008	-75,954
26	-110,496	-110,490
27	-53,577	-53,467
29	-386,549	-245,058
30	-433,617	-307,287
33	529,135	584,785

Table A.9: List of natural frequency comparisons for crane test problem.

Mode ID	Nastran	Simulation
1	75.3	75.3
2	94.5	94.4
3	111.0	110.9
4	141.7	141.6
5	233.7	233.6
6	382.4	382.2
7	443.3	443.1
8	504.3	504.0
9	708.1	707.7
10	882.0	882.0

Appendix B

Configuration Files

Listing B.1: Configuration file for controlling growth.

```
## Growth Configuration file
#

## Set the maximum time an individual is allowed to grow
maxIterations : 100

## file to capture the growth sequence for post-processing
captureFile : test.out

## distance that is considerd 'close enough'
## note that technically, units are non-dimensional
eBallSize : 1.0

## Diffusion parameters
#   K == K1 in text
#   T2 = K2 in text
T2 : 0.5
K : 0.000005
```

Listing B.2: Configuration file for controlling evolution.

```
# default configuration file for evolution
#

# select usage of MPI.  1=yes, 0=No
MPIstatus : 1

# random seed.In order to assist debugging, we can pick a specific seed
#   -1 = generate from clock
ranSeed : 124345

# define the method of selecting parents for crossover
#    1 = random parents
#    2 = exponential random parents
#    3 = rank order parents
parentSelectionScheme : 1

# define the method for downselecting the population
#    1 = random
#    2 = exponential random
#    3 = direct (elitist)
generationSelectionScheme : 3

# Enable various mutations
#
# in each case, 1=on, 0=off
enableGenomeDuplication : 1
enableGeneDuplication : 1
enableGeneDeletion : 1
enableGeneInsertion : 1
enablePointMutations : 1
enableAddBase : 1
enableDeleteBase : 1
enableAtomConvert : 1
enableDuplicateAtom : 1
enableAtomMutate : 1


# Probabilities of various events
#
geneDeletionProbability : 0.001
geneInsertionProbability : 0.01
geneDuplicationProbability : 0.001
genomeDuplicationProbability : 0.0001
baseInsertionProbability : .04
baseDeletionProbability : .002
baseDuplicationProbability : .01
baseConversionProbability : .01
```

Listing B.3: Configuration file for controlling growth.(cont.)

```
# limits
#
maxeProteins : 10
maxiProteins : 10
maxNodeTypes : 10
maxConnections : 10
maxGenerations : 1000
numIslands : 1
numIndividuals : 400

# Helper files
#     baseIndividual specifies a starting genome
#     matFile contains the valid material specifications
baseIndividual : test.gen
matFile : Debug.material

# file capture names
#
fitFileName : all.fitness
bestFileName : best.fitness

# fitness calculation variables
#
mass_weight : 1.0
mass_knee : 2.0
mass_shoulder : 1.0
stress_weight : 1.0
stress_knee : 1.0
stress_shoulder : 2.0
stiff_weight : 1.0
stiff_knee : 5.0
stiff_shoulder : 30.0
struct_weight : 1.0
aggeFactor : 2.0
```

Listing B.4: Configuration file for display post-processor.

```
## Set colors for various elements
#
fieldNodeColor1 : (1.0,0.0,0.0)
constNodeColor1 : (0.0,0.0,1.0)
loadNodeColor1 : (0.0,1.0,0.0)
protoLinkColor1 : (0.3,0.7,0.6)
linkColor1 : (0.68,0.68,0.78)
floorColor : (0.0,0.3,0.95)
backgroundColor : (0.0,0.0,0.0)

## Set display quality and parameters
#
nodeSlices : 10
nodeStacks : 10
CylSlices : 10
CylStacks : 10
ProtoStacks : 10
ProtoSlices : 10
windowSize : (1201x800)
fogDensity : 0.1
fogStart : 10.0
fogEnd : 3500.0

## Set lighting data
#
ambientLight : (1,1,1,1)
light0_ambient : (0.1,0.1,0.1,1)
light0_diffuse : (0.5,0.5,0.5,1)
light0_specular : (1,1,1,1)
light1_ambient : (0.1,0.1,0.1,1)
light1_diffuse : (0.5,0.5,0.5,1)
light0_position : (1,1,10000,0)
light1_position : (1,1,10000,0)
Light_Rot1 : (1,0,0,0)
Light_Rot2 : (0,1,0,0)
Light_Rot3 : (0,0,1,0)
Light_Rot4 : (0,0,0,1)

## Set view orientation
#
startPos : (-25.405,-281.255,2249.71,0)
startEuler1 : (0.976675,-0.0686528,0.203483,0)
startEuler2 : (0.214673,0.337542,-0.916507,0)
startEuler3 : (-0.00576338,0.938808,0.344402,0)
startEuler4 : (0,0,0,1)
```

Appendix C

Source Code

C.1 Acknowledgements

The author wishes the acknowledege the following provides of code and packages:

- *Anyoption*: A utility for parsing command line and option file commands with portability to various (non-POSIX) environments. Code and details may be found at `http://www.hackorama.com/anyoption`.

- *GLUI*: A Gui library based on OpenGL and GLUT. Provides platform independence and simple button and dialog box creation. Source and information available at `http://glui.sorceforge.net`.

- *GLUT*: An interface for OpenGL programing that hides most of the issues related to platform dependency. See [86] for details of implementation.

- *Numerical Recipies*: These form indispensible guide to a variety of numerical methods and were used for many of the algorithms and data structures used. See [73] for details.

- *OpenGL, STL, and C++*: The basic code was written is C++ [29] and greatly utilizing the STL [64, 67] library. All graphics and visualization was achieved with OpenGL [86].

- *CPPLapack*: C++ wrapper to the standard Lapack library, available at `http://cpplapack.sourceforge.net/`.

- *LAPACK*: Linear Algebra PACKage, provider of many easy, pre-made routines for solving systems of equations, available at `http://www.netlib.org/lapack/`.

- As well as the providers of Latex and packages Makeglos, hyperref, and others that were invaluable in the preparation of this document.

C.2 Code Listing

This will be a listing of important pieces of code. These are fragments from the actual code used. Generally, only the header files will be presented and when they provide some special insight, certain functions will also be presented. In some cases it has been simplified for clarity of presentation. It is also by no means complete. To obtain the complete code, please contact the author directly.

C.2.1 Environment Class

C.2.2 Phenotype Class

C.2.3 Genetic Operations

C.2.4 Parallel Client-Server Code

Listing C.1: Receive fitness on the server side.

```
/*********************
 * recvResults
 *     companion function to receive the results
 */
int Evaluator::recvResults( )
{
  int sender(0);
  int index(0);
  MPI::Status status;  // MPI req
  int tag(1);    // MPI req

  // receive index
  MPI::COMM_WORLD.Recv( &index, 1, MPI::INT, MPI::ANY_SOURCE
      , tag, status);

  // extract process number
  sender = status.Get_source();

  // receive fitness
  MPI::COMM_WORLD.Recv( &(genomeStack[index].fitness), 1,
     MPI::DOUBLE, sender, tag, status);

  return (sender);
}
```

Listing C.2: Send fitness on the client side.

```
/*********************
 * sendResults
 *    client routine to send the results back to the server
 */
int Evaluator::sendResults( )
{
  // send index
  MPI::COMM_WORLD.Send( &index, 1, MPI::INT, 0, 1);

  // send fitness
  MPI::COMM_WORLD.Send( &(myItem.fitness), 1, MPI::DOUBLE,
      0, 1);

  return (0);
}
```

Listing C.3: Receive the genome on the client side.

```
/*********************
 * recvData
 *  companion function to receive data on a client
 */
int Evaluator::recvData( )
{
  MPI::Status status;  // mpi req
  int tag(1);  // mpi req
  char *buf;
  int size(0);

  // get the index
  MPI::COMM_WORLD.Recv( &index, 1, MPI::INT, 0, tag, status)
     ;
  if (index < 0) // we are done
    return (-1);

  // get the buf size
  MPI::COMM_WORLD.Recv( &size, 1, MPI::INT, 0, tag, status);

  // get the buffer
  buf = new char[size+10];
  memset( buf, '\0', size+10);
  MPI::COMM_WORLD.Recv( buf, size, MPI::CHAR, 0, tag, status
     );

  // translate the data to a useful format
  std::string myString(buf);
  std::stringstream myStream(myString, std::stringstream::in
      | std::stringstream::out );
  myStream >> myItem.genome;

  // clean up
  delete [] buf;

  return (0);
}
```

Listing C.4: Send fitness on the client side.

```
/*********************
 * sendData
 *  send a piece of data to the client
 */
int Evaluator::sendData(int item, int dest )
{
  // translate to buffer
  int size;
  char *buf;
  std::stringstream myStream (std::stringstream::in | std::
     stringstream::out );

  myStream << genomeStack[item].genome;
  size = (myStream.str()).size();
  buf = new char[size+10];  // we also leave extra space,
     just in case
  memset( buf, '\0', size+10); // make sure there is nothing
      else
  (myStream.str()).copy( buf, size);  // copy the string to
     the buffer

  // send the index
  MPI::COMM_WORLD.Send( &item, 1, MPI::INT, dest, 1);

  // send buf size
  MPI::COMM_WORLD.Send( &size, 1, MPI::INT, dest, 1);

  // send the buffer
  MPI::COMM_WORLD.Send( buf, size, MPI::CHAR, dest, 1);

  // clean up
  delete [] buf;

  return (0);
}
```

Listing C.5: Client main loop.

```
/*********************
 * clientStart
 *   mpi client routine
 */
void Evaluator::clientStart( )
{
  // init stuff
  int notDone = 1;  // keeps track of when we are done
  int retVal(0);

  // loop while we have stuff to do
  while (notDone) {
    // receive data
    retVal = recvData();
    if (retVal < 0 )  // got a done signal
      return;

    // evaluate
    myItem.evaluate();

    // send results
    sendResults();
  }
}
```

Listing C.6: Server main loop.

```
/*********************
 * serverStart
 */
void Evaluator::serverStart( )
{
  int notDone(1);
  int currentSendingItem(0); // tracks which item we are up
     to
  int numReceived(0);  // tracks what has been returned
  int sender(0);
  int numItems=genomeStack.size();

  // Phase 1:  Send one job to each client
  for (int j=0; j<(numProc-1); j++ ) {
    sendData(currentSendingItem, j+1);
    currentSendingItem++;
  }

  // Phase 2
  while ( currentSendingItem < numItems) {
    sender = recvResults();
    sendData(currentSendingItem, sender);
    currentSendingItem++;
    numReceived++;
  }

  // Phase 3
  while (numReceived < numItems) {
    recvResults();
    numReceived++;
  }
}
```

Appendix D

Defense Slides

Automated Design Synthesis of Structures Using Growth Enhanced Evolution

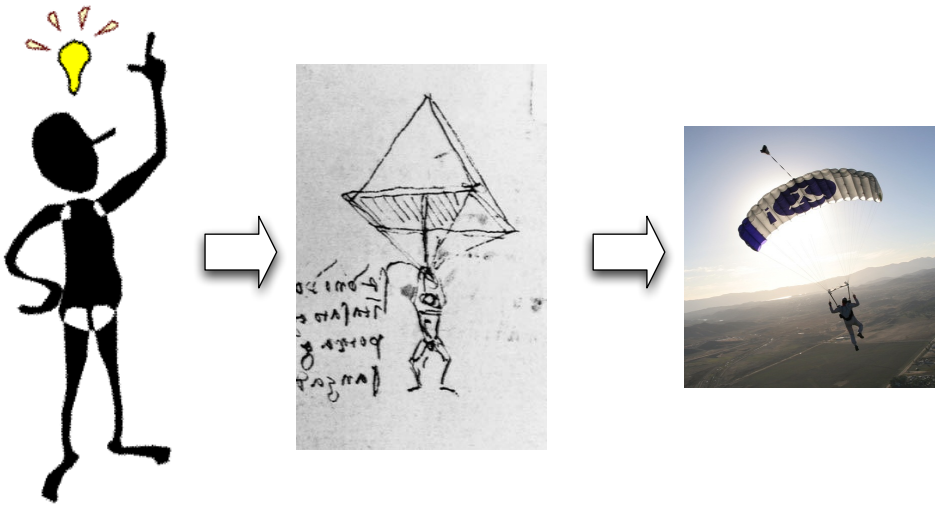
F. Nicaise

Engineering Design Research Laboratory
Department of Mechanical Engineering
Division of Engineering and Applied Science
California Institute of Technology
Pasadena, CA

19 September 2007
Thesis Defense Seminar

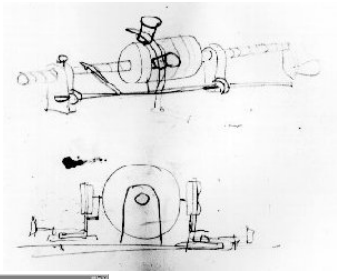
Motivation

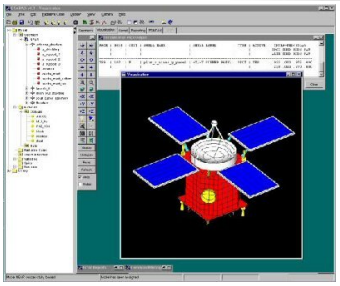
Motivation



Overview

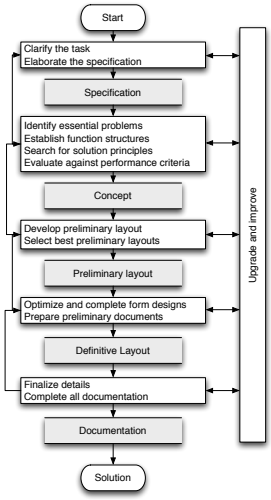
Conceptual Design





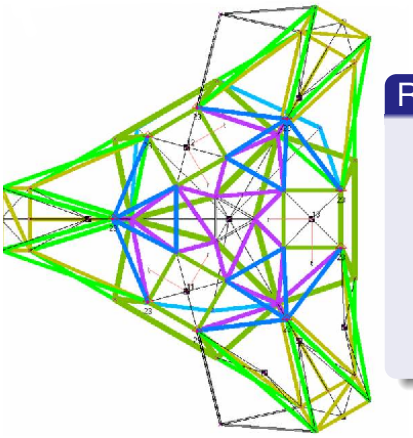
TEAMX
Jet Propulsion Laboratory

Bad Design?



(Pahl and Beitz)

Problems of Interest

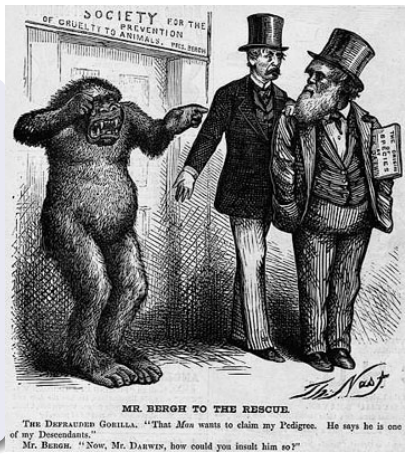


Restrictions on Domain

- Focus on structures (no moving parts)
- Further focus on discrete structures (trusses)
- Both of these can be added back in later...

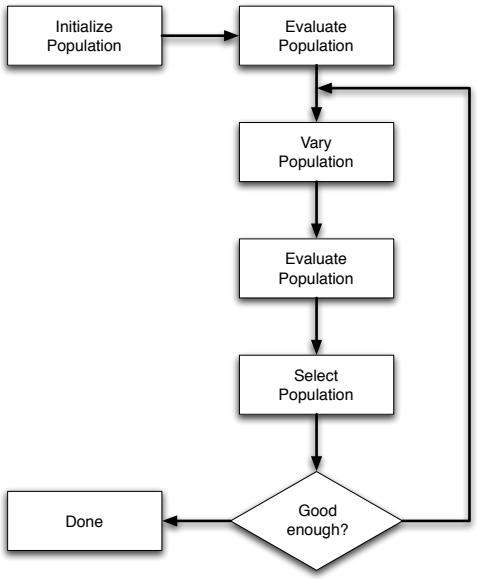
Evolutionary Algorithms (EAs)

- Based on Darwinian evolution from nature
- Maintain a *population* of solutions (*individuals*)
- Represent the design as a *Genome*
- Introduce *Variation* to change the population
- *Select* for best characteristics

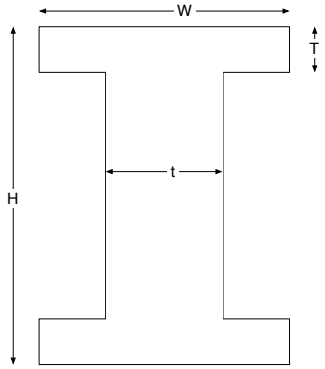


Thomas Nast, Harper's Weekly August 19, 1871

EA Flowchart



EA Example



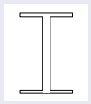
Consider I-Beam Optimization

- Genome is a list of parameters
- `[w][H][T][t]` (in inches)
- `[2.0][3.0][0.2][0.3]`

EA Example (cont.)

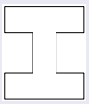
Parent 1

[2.0][3.0][0.2][0.3]



Parent 2

[3.0][3.5][0.8][1.0]

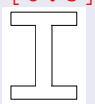


Mutation

[2.0][3.0][0.25][0.3]

Crossover (mean)

[2.5][3.25][0.5][0.65]



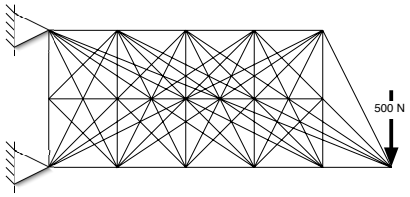
Types of EAs

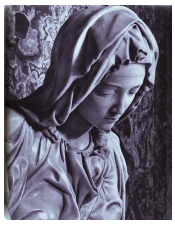
There are many types of Evolutionary Algorithms:

- Genetic Algorithms
- Genetic Programming
- Genetic Classifiers
- Simulated Annealing (mostly)
- ...

They are all treated as essentially the same.

Basic Truss Encoding

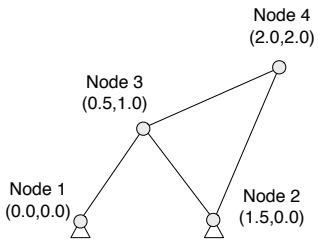




Base Structure

- Genome is on/off value for each link
- Extended to 3-Dimensions
- "Michelangelo Encoding": *In every block of marble I see a statue as plain as though it stood before me, shaped and perfect in attitude and action. I have only to hew away the rough walls that imprison the lovely apparition to reveal it to the other eyes as mine see it.*

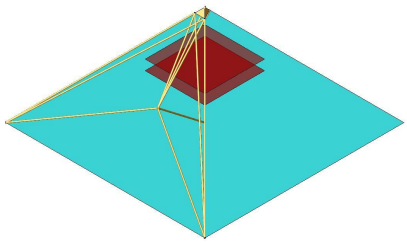
Basic Truss Encoding (cont.)

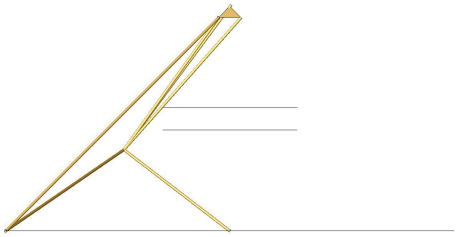


FEM Encoding

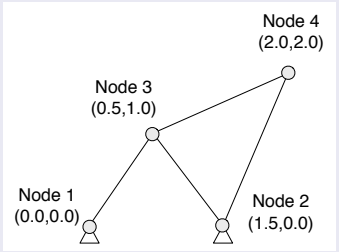
- Genome is set of two strings
  - List of nodes:
    `[0.0 0.0][1.5 0.0]...`
  - List of connections:
    `[1 3 5.0][2 3 3.0]...`
- Easily translated to analysis tools

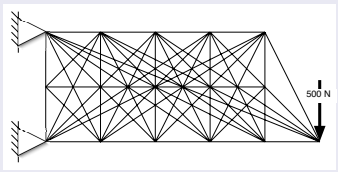
Use of Direct Truss Encoding





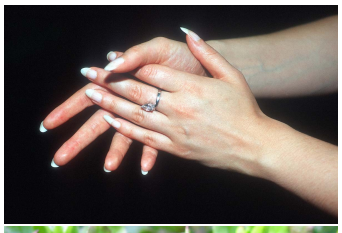
Problems with EAs



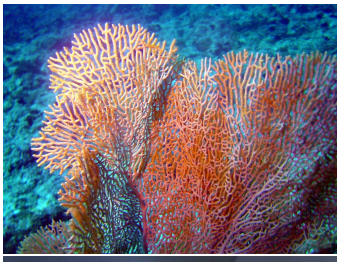


- Lack of *Scalability*
- Lack of *Emergence*
- Difficulty of *Topology Change*
- Lack of *Modularity*

Nature is Fascinating



Nature has Structures too



Keys to Biology

- Multi-cellular
- Abstract representation
- Flexible set of rules
- Local actions

Digital Agar

- 3-Dimensional continuous domain
- Gravity present but not enforced
- *Nodes* act like *cells*
- Cells grow connections between themselves
  - While growing: *proto-link*
  - Once connected: *link*
- Chemicals (*proteins*) allowed to diffuse across the space

Diffusion

- Modeling diffusion on an infinite, gridless domain is difficult
- Start with diffusion equation:

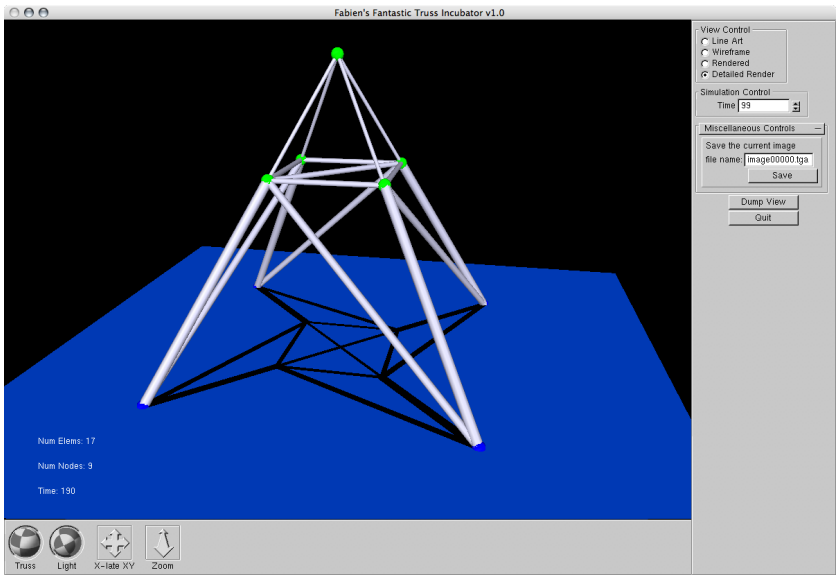
$$\frac{\partial \phi}{\partial t} = D\nabla^2 \phi\left(\vec{r}, t\right)$$

- Use Gaussian pulse as an approximate solution:

$$C\left(t\right) = \frac{K_1}{\sqrt{t}} e^{\frac{-K_2 \vec{r}^2}{t}}$$

- Parameters K_1 and K_2 are selected based on domain

Custom Application



Simple Example

Genome

```
[SUPl2]::::::[PRDe1]
[SUPc1][NLL1][ANYe1][ADDe1]::::::[GRDe1]
[SUPc1]::::::[MOM] [ANDe4]::::::[MAT]
```

Genome

- Genome consists of sets of *If-Then* type statements

	Cond.	Exp.
Gene1	`[SUPe1][NNEi2]:::::::`	`[PRDe1][GRDe1]`
Gene2	`[ANYe3]:::::::`	`[SPLf2]`
Gene3	`:::::::`	`[MAT]`

- Genes are each evaluated in sequence
- Each cell in an individual evaluates the same genome
- Evaluated in a synchronous fashion

Conditional Rules

Rule	Evaluation value Φ	
NOC	$\Phi(\Theta, NOC)$	$= \Theta$; no change/neutrality
ADD{PTx}	$(\Theta, ADD[PTx])$	$= \Theta - \{PTx\}$
SUB{PTx}	$(\Theta, SUB[PTx])$	$= \Theta + \{PTx\}$
MUL{PTx}	$(\Theta, MUL[PTx])$	$= \Theta * \{PTx\}$
ANY{PTx}	$(\Theta, ANY[PTx])$	$= \Theta$, if $\{PTx\} \neq 0$
NNY{PTx}	$(\Theta, NNY[PTx])$	$= \Theta$, if $\{PTx\} = 0$
AND{PTx}	$(\Theta, AND[PTx])$	$= \min(\Theta, \{PTx\})$
NND{PTx}	$(\Theta, NND[PTx])$	$= -\min(\Theta, \{PTx\})$
ORR{PTx}	$(\Theta, ORR[PTx])$	$= \max(\Theta, \{PTx\})$
NOR{PTx}	$(\Theta, NOR[PTx])$	$= -\max(\Theta, \{PTx\})$

Conditional Rules

The gene

`[ADDi1][MULe3][ORRi5]::::::[MOM][GRDe2]`

The calculation

- Initialize:
  $\Theta_0 = 0$
- Add the concentration of internal Protein 1:
  $\{i1\} = 4.5 \Longrightarrow \Theta_1 = \Theta_0 + 4.5 = 4.5$
- Multiply by the concentration of external Protein 3:
  $\{e3\} = 2.1 \Longrightarrow \Theta_2 = \Theta_1 * 2.1 = 9.45$
- Keep the max of Θ and the concentration of protein 5:
  $\{i5\} = 9.0 \Longrightarrow \Theta_3 = \max(\Theta_2, 9.0) = 9.45$
- Pass final value $\Theta = 9.45$ to expressive portion

Veto Rules

Rule	Represses Gene if
$SUP\{CTPx\}$	cell is not of type CPTx
$NSU\{CTPx\}$	cell is of type CPTx
$ANY\{PxX\}$	there is no protein of type PTx
$NNY\{PTx\}$	there is no protein of type PTx
LGT	length of link is longer than gene
LLT	length of link is shorter than gene
NLM	number of connections is more than gene
NLL	number of connections is less than gene

Expressive Rules

Rule	Statement Description
$GRD\{PTx\}$	Grow proto-link following gradient of PTx
MAT	Change proto-link material
MOM	Increase the area of the proto-link
RLX	Scale the area of the proto-link
$MOV\{PTx\}$	Move the node following the gradient of PTx
$PRD\{PTx\}$	Produce more protein PTx
$SPL\{CTPx\}$	Split off the proto-link into a full link
DIE	Destroy the node and all attached links
NOP	No action, neutrality

Expressive Rules

The gene

```
[ADDi1][MULe3][ORRi5]::::::[MOM][GRDe2]
```

The calculation

- Remember $\Theta = 9.45$ from above
- Increase cross-sectional area by 9.45
- Grow proto-link by 9.45 in the direction of e2
- Discard Θ

Simple Hand Coded Example

Genome

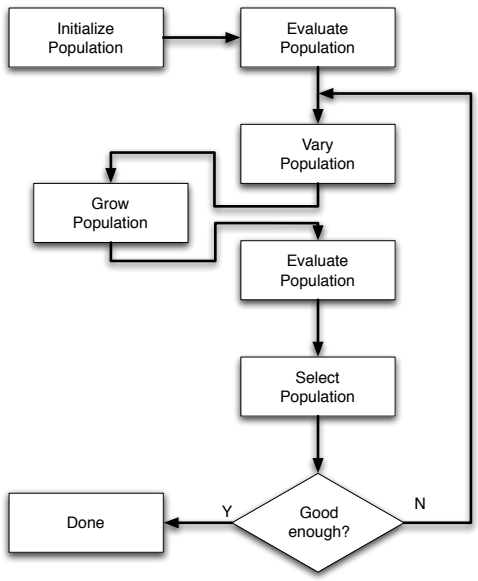
```
[SUPl2]::::::[PRDe1]
[SUPc1][NLL1][ANYe1][ADDe1]::::::[GRDe1]
[SUPc1]::::::[MOM] [ANDe4]::::::[MAT]
```

A Complex Hand Coded Example

A Complex Example Genome

```
[SUPL2]::::::[PRDE1]
[SUPC0][NLL1][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPC1][NLL1][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPC2][NLL1][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPC3][NLL1][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPC0]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPC1]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPC2]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPC3]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[ANDE4]::::::[MAT]
[SUPC0][NLL1][LLT750]::::::[SPL0]
[SUPC1][NLL1][LLT750]::::::[SPL1]
[SUPC2][NLL1][LLT750]::::::[SPL2]
[SUPC3][NLL1][LLT750]::::::[SPL3]
[SUPf0][NLL2][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPf1][NLL2][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPf2][NLL2][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPf3][NLL2][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE1]
[SUPf0]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPf1]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPf2]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPf3]::::::[MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM][MOM]
[SUPf0]::::::[PRDE2]
[SUPf1]::::::[PRDE3]
[SUPf2]::::::[PRDE4]
[SUPf3]::::::[PRDE5]
[SUPf0][NLM1][NLL4][ANYE3][ADDE3][ADDE3]::::::[GRDE3]
[SUPf1][NLM1][NLL4][ANYE4][ADDE4][ADDE4]::::::[GRDE4]
[SUPf2][NLM1][NLL4][ANYE5][ADDE5][ADDE5]::::::[GRDE5]
[SUPf3][NLM1][NLL4][ANYE2][ADDE2][ADDE2]::::::[GRDE2]
[SUPf3][NLM3][NLL6][ANYE3][ADDE3][ADDE3]::::::[GRDE3]
[SUPC0][NLL2][NLM0][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE3]
[SUPC1][NLL2][NLM0][ANYE1][ADDE1][ADDE1][ADDE1]::::::[GRDE4]
```

Algorithm Overview



Mutation

- Point mutations
- Base duplication
- Gene duplication
- Genome duplication

```
[SUPl2]::::::[PRDe1]
[SUPc1][NLL1][ANYe1][ADDe1]::::::[GRDe1]
[SUPc1]::::::[MOM]
[ANDe4]::::::[MAT]
```

Crossover

Select whole genes from each parent

Parent1	Parent2
[NSPl2]::::::[DIE]	[SUPl5]::::::[PRDe1]
[NLL1]::::::[GRDe1]	[SUPc1]::::::[RLX]
[ANDe4]::::::[MAT]	[NNYi3]::::::[SPLf4]

Child:

```
[NSPl2]::::::[DIE]
[SUPc1]::::::[RLX]
[NNYi3]::::::[SPLf4]
[ANDe4]::::::[MAT]
```

Fitness Evaluation

- Mass
$$f_m = \sum_i m_i, i \in \text{links}$$
- Stiffness
$$f_s = \min_i (f_i), i \in \text{natural frequencies}$$
- Stess (factor of safety)
$$f_\sigma = \max_i \left(\frac{\sigma}{\sigma_y} \right), i \in \text{links}$$
- Quality
$$f_t = [0, 1]$$

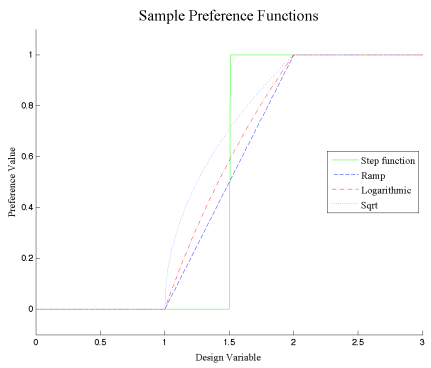
Fitness Evaluation (cont.)

Quality measure, f_t

- $[0, 0.25]$ 3 base nodes
- $[0.25, 0.5]$ Structure
- $[0.5, 0.75]$Ghost links
- $[0.75, 1.0]$ All load nodes

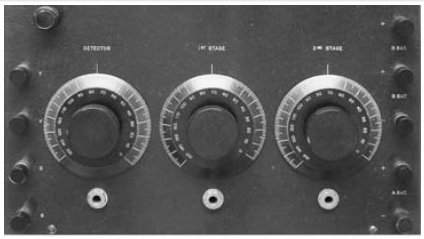
Evolution

Non-dimensionalization



- Make all parameters comparable
- Place on scale of $[0, 1]$
- Designer choices:
  - Minimum threshold
  - Maximum threshold
  - Transition

$$F_i = \mathcal{G}(f_i), i \in \{m, s, \sigma, t\}$$



Evolution

Aggregation

Method Of Imprecision

$$P = \left(\frac{w_m F_m^{s_1} + w_s F_s^{s_1} + w_\sigma F_\sigma^{s_1}}{w_m + w_s + w_\sigma} \right)^{\frac{1}{s_1}}$$

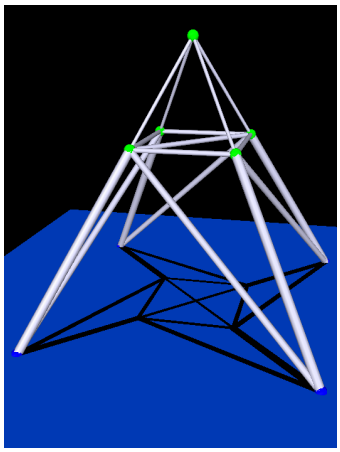
$$\text{Fitness} = e^{\left(\frac{P^{s_2} + w_t F_t^{s_2}}{1 + w_t} \right)^{\frac{1}{s_2}}}$$

- w_i represents the weighting factor
- s_1, s_2 represents the degree of compensation

Example 1

Example 2

Summary



- Explored the needs of design
- Reviewed relevant biology
- Created a growth mechanism
- Demonstrated growth possibility with examples
- Demonstrated evolve-ability with examples

Work Undone

- Extend the domain
  - Other structures
  - Mechanisms
- More interesting examples
- Wildcards
- Keep-out zones
- More evaluations
- Better diffusion
- Continuous evaluation
- Module identification (pheno+geno-types)
- More realistic examples

Acknowledgments

This task was funded in part by NASA-JPL, through a grant from the MSL program.

The supercomputers used in this investigation were provided by funding from the JPL Office of the Chief Information Officer.

Acknowledgments

I'd like to thank my advisor, Erik Antonsson, and Chris Adami for all of their support, guidance, and knowledge over the past few years.

I'd like to thank my committee for their contributions and dedication, and for putting up with two of these in a row.

And finally, I'd like to acknowledge my friends and colleagues, especially the SOPS soccer team, who managed to keep me sane (I think).

Glossary

A

ACO see *Ant Colony Optimization*.

Agent Based Design see *Intelligent Agent Design*.

Ant Colony Optimization A stochastic optimization technique based ant colonies and their search for food. **add citation**.

C

Crossover A genetic process by which two (or more) individuals are combined to make a new individual sharing traits from both parents.

D

design The act or process of creating a new concept for a perceived need.

E

EA see *Evolutionary Algorithms*.

Engineering Design The act or process of creating a new process through analysis and calculation for a specific need.

Evolutionary Algorithms A supperset of the various algorithms that all take their basis in biological evolution. ** clean up **.

F

FEM Finite E Methods or Finite Element Modeling. A computational method of performing structural analysis by breaking the structure into small pieces.

I

IA see *Intelligent Agent Design*.

industrial design see *design*.

Intelligent Agent Design A method of automated design based on giving heuristic design rules to a computing process. ***add citation***.

M

MPI Message Passing Interface. Consists of a library of message passing functions for use in high performance, parallel computing.

Mutation A process whereby a gene or portion of a gene is spontaneously changed to something else.

S

SA see *Simulated Annealing*.

Selection The process by which certain individuals are chosen to survive to the next generation.

Simulated Annealing A stochastic, point-to-point optimal search optimization. Based on the principals of metalurgical annealing, where atoms are excited and cooled to reach an optimal strength. See [21, 56].

Bibliography

[1] *MPI: The Complete Reference*. The MIT Press, 1998.

[2] *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press, 1999.

[3] *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000.

[4] ABD-EL-BARR, M., SAIT, S., SARIF, B., AND AL-SAIARI, U. A modified ant colony algorithm for evolutionary design of digital circuits. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2003)* (Canberra, Australia, december 9-12 2003), IEEE Press, pp. 708–715.

[5] ABRAHAM, A., AND RAMOS, V. Web usage mining using artificial ant colony clustering and Linear Genetic Programming. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2003)* (Canberra, Australia, december 9-12 2003), IEEE Press, pp. 1384–1391.

[6] ACAN, A., ALTINÇAY, H., TEKOL, Y., AND ÜNVEREN, A. A genetic algorithm with multiple crossover operators for optimal frequency assignment problem. *Evolutionary Computation 1* (2003), 256–263.

[7] ADAMATZKY, A., AND HOLLAND, O. Reaction-diffusion and ant-based load balancing of communication networks. *Kybernetes 31*, 5-6 (2002), 667–681.

[8] ADAMI, C., AND ASTOR, J. A developmental model for the evolution of artificial neural networks. *Artificial Life 6* (2000), 189–218.

[9] ALANDER, J. T., Ed. *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)* (Vaasa (Finland), 19.-23. Aug. 1996), Proceedings of the University of Vaasa, Nro. 13, University of Vaasa. (available via anonymous ftp site: ftp.uwasa.fi directory cs/2NWGA file *.ps.Z).

[10] ANDO, S., AND IBA, H. Ant Algorithm For Construction Of Evolutionary Tree. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (New York, 9-13 July 2002), W. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. Potter, A. Schultz, J. Miller, E. Burke, and N. Jonoska, Eds., Morgan Kaufmann Publishers, San Francisco, CA, p. 131.

[11] ANTONSSON, E. K., AND CAGAN, J., Eds. *Formal Engineering Design Synthesis*. Cambridge University Press, Cambridge, U.K., 2001.

[12] ARCISZEWSKI, T., SAUER, T., AND SCHUM, D. Conceptual design: Chaos-based approach. *Journal of Intelligent and Fuzzy Systems 13*, 1 (2003), 45–50.

[13] BACARDIT, J., AND KRASNOGOR, N. Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In *GECCO '06: Proceedings of the 8th*

annual conference on Genetic and evolutionary computation (New York, NY, USA, 2006), ACM Press, pp. 1441–1448.

[14] BERTELLE, C., DUTOT, A., GUINAND, F., AND OLIVIER, D. DNA sequencing hybridization based on multi-castes ant system. In *Proceedings of the BIXMAS/AAMAS Conference* (Bologna, Italy, July 2002).

[15] BLUM, C., AND SOCHA, K. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *Proceedings of Hybrid Intelligent Systems Conference, HIS-2005* (2005).

[16] CAMP, C. V., BICHON, B. J., AND STOVALL, S. P. Design of steel frames using ant colony optimization. *ASCE Journal of Structural Engineering 131*, 3 (2005), 369–379.

[17] CAMPBELL, M., CAGAN, J., AND KOTOVSKY, K. A-design: An agent-based approach to conceptual design in a dynamic environment. *Research in Engineering Design 11*, 3 (1999), 172–192.

[18] CAMPBELL, M., CAGAN, J., AND KOTOVSKY, K. Agent-based synthesis of electromechanical design configurations. *ASME Journal of Mechanical Design 122*, 1 (2000), 61–69.

[19] CAMPBELL, N. A. *Biology*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[20] CARO, G. D. *Ant Colony Optimization and its Application to Adaptive Routing in Telecommunication Networks*. PhD thesis, IRIDIA Universite Libre de Bruxelles, Brussels, Belgium, 2004.

[21] CERNY, V. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications 45* (1985), 41–51.

[22] CHEN, S.-J., AND HWANG, C.-L. *Fuzzy Multiple Attribute Decision Making: Methods and Applications*, vol. 375 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, New York, NY, 1992. In collaboration with Frank P. Hwang.

[23] CHUN, J.-S., JUNG, H.-K., AND YOON, J.-S. Shape optimization of closed slot type permanent magnet motors for cogging torque reduction using evolution strategy. *Magnetics, IEEE Transactions on 33*, 2 (1997), 1912–1915.

[24] COMELLAS, F., AND OZÓN, J. Graph Coloring Algorithms for Assignment Problems in Radio Networks. In *Applications of Neural Networks to Telecommunications 2* (1995), Lawrence Erlbaum Assoc. Inc. Pub., Hillsdale, NJ 07642, pp. 49–56.

[25] CONNOR, J. J. *Analysis of Structural Member Systems*. The Ronald Press Company, New York, 1976.

[26] CROW, K. D., AND WAGNER, G. P. What is the role of genome duplication in the evolution of complexity and diversity. *Molecular Biology and Evolution* (2005).

[27] DARWIN, C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, London, England, 1859.

[28] DARWIN, E. *The Temple of Nature or, the Origin of Society: A Poem, with Philosophical Notes*. J. Johnson, London, England, 1802.

[29] DEITEL, H. M., AND DEITEL, P. J. *C++ How to Program*. Prentice Hall, 1994.

[30] DORIGA, M. *Ottimizzazione, Apprendimento Automatico, ed Algoritmi Basati su Metafora Naturale (Optimization, Learning and Natural Algorithms)*. PhD thesis, Politecnico di Milano, Italy, 1992. Original source of Ant Colony Optimization.

[31] DORIGO, M., CARO, G. D., AND GAMBARDELLA, L. M. Ant algorithms for discrete optimization. *Artificial Life 5*, 2 (1999), 137–172.

[32] DORIGO, M., AND GAMBARDELLA, L. M. Ant colonies for the traveling salesman problem. In *BioSystems* (1997), vol. 43, pp. 73–81.

[33] E. J. DAVISON, R. A. Numerical optimization of large interconnected systems. *AIChE Journal 15*, 2 (1969), 276–281.

[34] EIBEN, A., AND BÄCK, T. An empirical investigation of multi-parent recombination operators in evolution strategies. *Evolutionary Computation 5*, 3 (1997), 347–365.

[35] EIBEN, A., VAN KEMENADE, C., AND KOK, J. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In *Proceedings of the 3rd European Con-*

ference on Artificial Life (1995), F. Moran, A. Moreno, J. Merelo, and P. Chacon, Eds., no. 929, Springer-Verlag, pp. 934–945.

[36] EIBEN, A. E., RAUE, P.-E., AND RUTTKAY, Z. Genetic algorithms with multi-parent recombination. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature* (1994), Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., no. 866, Springer-Verlag, pp. 78–87.

[37] ERTAS, A., AND JONES, J. C. *The Engineering Design Process*. John Wiley & Sons, Inc., 1996.

[38] ESQUIVEL, S. C., LEIVA, H. A., AND GALLARD, R. H. Multiple crossover between multiple parents to improve search in evolutionary algorithms. *Evolutionary Computation 2* (1999), 1594.

[39] FERNANDES, C., RAMOS, V., AND ROSA, A. Varying the Population Size of Artificial Foraging Swarms on Time Varying Landscapes. In *15th Int. Conf. on Artificial Neural Networks: Biological Inspirations (ICANN'05)* (Warsaw, Poland, 2005), W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, Eds., vol. 3696 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 311–316.

[40] FEYMAN, R. P. *"Surely You're Joking Mr. Feyman!"*. Bantam Books, 1985.

[41] FIGUEIREDO, K., VELLASCO, M., AND PACHECO, M. A. Mobile robot control using intelligent agents. In *Second international workshop on Intelligent systems design and application* (Atlanta, GA, USA, 2002), Dynamic Publishers, Inc., pp. 201–206.

[42] FOGEL, L., OWENS, A., AND WALSH, M. *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, New York, 1966.

[43] FUNES, P., AND POLLACK, J. Computer evolution of buildable objects. In *Fourth European Conference on Artificial Life* (1997), P. Husbands and I. Harvey, Eds., MIT Press, pp. 358–367.

[44] GIELEN, G., WALSCHARTS, H., AND SANSEN, W. analog circuit design optimization based on symbolic simulation and simulated annealing.

[45] GOMEZ-SKARMETA, A. F., PIERNAS, J., AND DELGADO, M. Fuzzy clustering and images reduction. In *Fuzzy Days* (1997), pp. 241–249.

[46] GRIFFITHS, A. J. F., MILLER, J. H., SUZUKI, D. T., LEWONTIN, R. C., AND GELBART, W. M. *An Introduction to Genetic Analysis*. W. H. Freeman and Company, New York, NY, 2003.

[47] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[48] HONDA, T., NICAISE, F., AND ANTONSSON, E. K. Synthesis of Structural Symmetry Driven by Cost Saving. In *31st Design Automation Conference (DAC)* (Sept. 2005), ASME. Paper Number: DETC2005-85111.

[49] HORNBY, G. S. Creating complex building blocks through generative representations. In *Computational Synthesis: from basic building blocks to high level functionality* (Stanford, CA, March 24-26 2003), H. Lipson, E. K. Antonsson, and J. R. Koza, Eds., AAAI Spring Symposium, AAAI, AAAI.

[50] HORNBY, G. S., AND LIPSON, H. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation 19*, 4 (August 2003).

[51] JACOBSON, S. H., AND YČESAN, E. Global optimization performance measures for generalized hill climbing algorithms. *J. of Global Optimization 29*, 2 (2004), 173–190.

[52] JONG, K. A. D. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[53] KAPLAN, R., AND KAPLAN, E. *The art of the infinite: The pleasures of mathematics*. Oxford University Press, 2003.

[54] KICINGER, R., ARCISZEWSKI, T., AND JONG, K. D. Morphogenic evolutionary design: Cellular automata representations in topological structural design. In *Adaptive Computing in Design and Manufacturing VI* (2004), I. C. Parmee, Ed., ACDM, Springer-Verlag.

[55] KICINGER, R. P. *Emergent Engineering Design: Design Creativity and Optimality Inspired by Nature*. PhD thesis, George Mason University, 2004.

[56] KIRKPATRICK, S., GELATT, JR., C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–679.

[57] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[58] LAHTINEN, J., MYLLYMÄKI, P., SILANDER, T., AND TIRRI, H. Empirical comparison of stochastic algorithms in a graph optimization problem. In Alander [9], pp. 45–60. (available via anonymous ftp site: ftp.uwasa.fi directory cs/2NWGA file *.ps.Z).

[59] LANGHAM, A. E., AND GRANT, P. W. Evolving rules for a self-organizing finite element mesh generation algorithm. In *Proceedings of 1999 Congress on Evolutionary Computation* (Washington, DC, July 1999), IEEE Computer Science, pp. 161–168.

[60] LEE, C.-Y. *Efficient Automatic Engineering Design Synthesis through Evolutionary Exploration*. PhD thesis, California Institute of Technology, Pasadena, CA, June 2002.

[61] LEE, C.-Y., AND ANTONSSON, E. K. Variable Length Genomes for Evolutionary Algorithms. In *GECCO 2000, Proceedings of the Genetic and Evolutionary Computation Conference* (July 2000), IEEE, Morgan Kaufmann, p. 806.

[62] LEE, J., AND CHOI, M. Y. Optimization by multicanonical annealing and the traveling salesman problem. *Phys. Rev. E 50*, 2 (Aug 1994), R651–R654.

[63] LINDENMAYER, A. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology 18* (1968), 280–315.

[64] LISCHNER, R. *STL: Pocket Reference*. O'Reilly, 2004.

[65] MANIKAS, T. W., AND CAIN, J. T. Genetic algoritms vs. simulated annealing: A comaparison of approaches for solving the circuit partition promblem. Tech. rep., Universy of Pittsburch, Dept. of Electrical Engineering, May 1996.

[66] MCCONNELL, S. *Code Complete*. Microsoft Press, 2004.

[67] MUSSER, D. R., DERGE, G. J., AND SAINI, A. *STL Tutorial and Reference Guide*, second ed. Addison-Wesley, Boston, 2005.

[68] NAGPAL, R., KONDACS, A., AND CHANG, C. Programming methodology for biologically-inspired self-assembling systems. In *Computational Synthesis: from basic building blocks to high level functionality* (Stanford, CA, March 24-26 2003), H. Lipson, E. K. Antonsson, and J. R. Koza, Eds., AAAI Spring Symposium, AAAI, AAAI.

[69] PAHL, G., AND BEITZ, W. *Engineering Design*. The Design Council, Springer-Verlag, New York, NY, 1984.

[70] PARK, M. K., LEE, M. C., AND GO, S. J. The design of sliding mode controller with perturbation observer for a 6-dof parallel manipulator. In *Industrial Electronics* (2001), vol. 3, IEEE, IEEE, pp. 1502–1507.

[71] PETROSKI, H. *Small Things Considered*. Alfred A. Knopf, 2003.

[72] PEYSAKHOV, M., AND REGLI, W. C. Using assembly representations to enable evolutionary design of Lego structures. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing 17* (2003), 155–168.

[73] PRESS, W. U., VETTRILING, W. T., TEUKOLSKY, S. A., AND FLANNERY, B. P. *Numerical Recipies in C++*, second ed. Cambridge University Press, New York, 2002.

[74] R. A. RIBEIRO, F. M. P. Fuzzy linear programming via simulated annealing. vol. 35, pp. 57–67.

[75] SAITOU, K., AND JAKIELA, M. J. Meshing of engineering domains by meitotic cell division. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (1994), The MIT Press, pp. 289–294.

[76] SCOTT, M. J. *Formalizing Negotiation in Engineering Design*. PhD thesis, California Institute of Technology, Pasadena, CA, June 1999.

[77] SCOTT, M. J., AND ANTONSSON, E. K. Aggregation Functions for Engineering Design Trade-offs. *Fuzzy Sets and Systems 99*, 3 (1998), 253–264.

[78] SERRA, M., AND VENINI, P. On some applications of ant colony optimization metaheuristic to structural optimization problems. 6th World Congress of Structural and Multidisciplinary Optimization.

[79] SHANDONG, W., AND YIMIN, C. Miam: a robot oriented mobile intelligent agent model. 51–54.

[80] SHEA, K., AND SMITH, I. Applying shape annealing to full-scale transmission tower redesign. In *Proceedings of DETC99: ASME Design Engineernig Technical Conferences* (New York, 1999), ASME. Paper Number: DAC-8681.

[81] SUH, N. P. *The Principles of Design*. Oxford University Press, New York, NY, 1990.

[82] VAN LAARHOVEN, P. J. M., AARTS, E. H. L., AND LENSTRA, J. K. Job shop scheduling by simulated annealing. *Operations Research 40*, 1 (1992), 113–125.

[83] VARRICCHIO, S. L., AND MARTINS, N. Filted design using a Newton-Raphson method based on eigenvalue sen. In *Power Engineering Society Summer Meeting* (2000), vol. 2, IEEE, pp. 861–866.

[84] WANG, B. *Information-Teoretic Methods for Modularity in Engineering Design*. PhD thesis, California Institute of Technology, Pasadena, CA, June 2007.

[85] WOLFE, K., AND SHIELDS, D. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature 387* (1997), 708–13.

[86] WRIGHT, R. S., AND LIPCHAK, B. *OpenGL Superbible*, third ed. Sams, Indianapolis, 2005.

[87] YOGEV, O., AND ANTONSSON, E. K. A Novel Synthesis Design Approach for Continuous Inhomogeneous Structures. In *19th International Conference on Design Theory and Methodology (DTM)* (Sept. 2007), ASME. Paper Number: DETC2007/DTM-35662.

[88] YOGEV, O., AND ANTONSSON, E. K. Evolution of Continuous Structures. In *16th International Conference on Engineering Design (ICED)* (Aug. 2007), The Design Society.

[89] YOGEV, O., AND ANTONSSON, E. K. Growth and Development of Continuous Structures. In *GECCO 2007, Genetic and Evolutionary Computation Conference* (London, UK, 2007).