Approaches to Evolutionary Architectural Design Exploration Using Grammatical Evolution

Jonathan Byrne

The thesis is submitted to University College Dublin
for the degree of Ph.D.
at the School of Computer Science and Informatics
Research Supervisors:
Dr. Michael O'Neill
Prof. Anthony Brabazon

External Examiner:
Prof. Penousal Machado

August 6, 2012

Abstract

The architectural design process is both subjective and objective in nature. The designer and end user judge a design not only by objective functionality but also by subjective form. Despite the ability of evolutionary algorithms to produce creative and novel designs, they have primarily been used to aid the design process by optimising the functionality of a design, once it has been instantiated. Designers should be able to express their subjective and objective intentions with a design tool. Grammatical evolution (GE) is a form of genetic programming that allows evolutionary techniques to be applied to systems that can be represented as a grammar. This thesis examines approaches that allow grammatical evolution to be used in the exploration phase of the architectural design process as well as optimising the design to maximise functionality.

The primary focus of this thesis is to increase the amount of direct and indirect interaction available to the designer for evolutionary design exploration. The research gaps which this thesis investigates are the use of novel GE operators for active user intervention, the development of interfaces suitable for directing evolutionary search and the application of functional constraints for guiding aesthetic evolution. The contributions made by this thesis are the development of two component mutation operators, a novel animated interface for user-directed evolution and the implementation of a multi-objective finite element analysis fitness function in GE for the first time.

An examination of fitness functions, operators and representations is carried out so that the designer's input to the evolutionary algorithm can be enhanced. An extensive review of computer-generated architecture, interactive evolution and grammatical evolution is conducted. Initial investigations explore whether the constraints placed on architectural designs can be expressed as a multi-objective fitness function. The application of this technique, as a means of reducing the search space presented to the architect, is then evaluated.

Broadening interaction beyond evaluation increases the amount of feedback and bias a user can apply to the search. A study is conducted to examine how integer mutation in GE explores the search space. Two novel and distinct behavioural components in GE mutation are shown to exist, nodal and structural mutation. The locality of the operations is examined at different levels of the derivation process. It is shown that nodal and structural mutation cause different magnitudes of change at the phenotypic level.

An interface is designed that enables the architect to directly mutate design encodings that they find aesthetically pleasing. User trials are then conducted on an interface for making localised changes to an individual and evaluate whether it is capable of directing search. The results show that users initially apply structural mutations to explore the search space and then apply smaller nodal mutations to fine tune a solution. The novel interface is shown to enable directed evolutionary search.

To Andrea and my parents

Acknowledgments

First and foremost I would like to thank Michael O'Neill, my thesis supervisor, for making all this possible. Michael has a gift for directing my interest while simultaneously letting me explore and experiment with whatever captures my attention. The volume of work produced during this thesis would not have been possible without Michael's continuous encouragement and support

I want to thank all the Natural Computation Research and Application group members for creating such a positive work environment. Our group shared common interests that allowed us to bounce ideas off each other and discuss different approaches to problems. The excitement and acceptance of new ideas led to great collaborations amongst the group.

In particular Dr Erik Hemberg must be thanked for helping me bring my thesis to fruition. The most productive part of the day has always been the informative balcony rants that took place every morning. Everything was discussed, from the minutiae of mutation to the meaning of life. If it was not for his cruel and unusual demands for more chapters to review, this thesis may never have been completed.

I would like to thank Brian Lynch for keeping me sane during my daily commute these past three years. He has made me on laugh every day with his razor sharp wit and degenerate humour. His creative and aesthetic input on several of my projects has been invaluable and greatly appreciated.

My most heartfelt thanks must go to my parents. They have assisted and encouraged my education throughout my life, from my first Atari XE to keeping a roof over my head during my PhD. I am humbled by their support and forever indebted to them.

Most of all I have to thank Andrea for her limitless patience. She has endured interminable rants and is still unceasingly supportive of my work. Her goodness, love and enthusiasm have been my greatest motivation.

I would also like to thank Science Foundation Ireland. This research is based upon works supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

Contents

List of Figures

List of Tables

Publications Arising

1. Byrne J., Hemberg E., O'Neill M., and Brabazon A. A local search interface for interactive evolutionary architectural design. In *Applications of Evolutionary Computing, EvoApplications 2012: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, LNCS, Malaga, Spain, 2012. Springer Verlag

2. Jonathan Byrne, Michael O'Neill, Erik Hemberg, and Anthony Brabazon. Analysis of constant creation techniques on the binomial-3 problem with grammatical evolution. In Andy Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 568–573, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press

3. J. Byrne, M. O'Neill, and A. Brabazon. Structural and nodal mutation in grammatical evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1881–1882. ACM, 2009

4. Jonathan Byrne, James McDermott, Michael O'Neill, and Anthony Brabazon. An analysis of the behaviour of mutation in grammatical evolution. In *Genetic Programming, Proceedings of EuroGP'2010*, pages 14–25. Springer-Verlag, 2010

5. J. Byrne, M. O'Neill, and A. Brabazon. Optimising offensive moves in toribash. In R. Matousek, editor, *Proceedings of Mendel 2010 16th International Conference on Soft Computing*, pages 78–85, Brno, Czech Republic, 2010. Brno University of Technology

6. Jonathan Byrne, James McDermott, Edgar Galván López, and Michael O'Neill. Implementing an intuitive mutation operator for interactive evolutionary 3d design. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2010

7. Jonathan Byrne, Erik Hemberg, and Michael O'Neill. Interactive operators for evolutionary architectural design. In *GECCO '11: Proceedings of the 13th annual con-*

ference companion on Genetic and evolutionary computation, pages 43–44, Dublin, Ireland, 12-16 July 2011. ACM. doi: doi:10.1145/2001858.2001884

8. Jonathan Byrne, Michael Fenton, Erik Hemberg, James McDermott, Michael O'Neill, Elizabeth Shotton, and Ciaran Nally. Combining structural analysis and multi-objective criteria for evolutionary architectural design. In *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, volume 6625 of *LNCS*, pages 200–209, Turin, Italy, 27-29 April 2011. Springer Verlag

9. Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1035–1042, Montreal, 8-12 July 2009. ACM. doi: doi:10.1145/1569901.1570041

10. Michael O'Neill, James McDermott, John Mark Swafford, Jonathan Byrne, Erik Hemberg, Elizabeth Shotton, Ciaran McNally, Anthony Brabazon, and Martin Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 2011

11. James McDermott, Jonathan Byrne, John Mark Swafford, Michael O'Neill, and Anthony Brabazon. Higher-order functions in aesthetic EC encodings. In *2010 IEEE World Congress on Computational Intelligence*, pages 2816–2823, Barcelona, Spain, 2010. IEEE Press. doi: doi:10.1109/CEC.2010.5586077

Chapter 1

Introduction

Aesthetic and functional problems are ever present in the design process. The process of biological evolution has clearly demonstrated its power to design elegant forms and structures in the name of survival. As such, it is natural to turn to algorithms which are inspired by this process to tackle design problems. Evolutionary algorithms (EAs) are powerful problem-solving tools. This thesis will focus on novel methods for applying EAs to the architectural design process. The design process can be broken into two different components: the conceptual phase, where the design space is explored and the optimisation phase, where the design space is searched for the optimal implementation of the design.

Evolutionary design has focused on the optimisation component of the design process as it can be described with clearly defined fitness functions; accordingly, evolutionary algorithms excel at this task. Conceptual design is more focused on the subjective qualities of the design which only the designer can impart. There is also the potential for a design for which it is difficult to find qualitative metrics. The research objective of this thesis is to investigate alternative methodologies that allow the designer to use efficiently an evolutionary algorithm as a design tool.

Architectural design is an appropriate application for design exploration and optimisation. Architecture is as closely related to art as it is to engineering because the aesthetic of a structure is an important feature of the overall design. Yet architecture is more restricted than traditional art forms. There are several constraints placed on architectural designs such as functionality and structural integrity. The mix of objective and aesthetic considerations make it a suitable challenge for evolutionary design exploration.

1.1 Problem Definition

The experiments in this thesis investigate different methodologies for architectural design exploration. Exploration is different to optimisation in that the goals are less clearly defined. The intention of the user is to discover new designs or processes. Architectural design is both the art and science of building. It is more objective than pure design as the structures must be functional. Despite this, the architectural process has a consideration for aesthetic qualities that compliment the utility of a structure. The following sections describe the problem area in greater detail.

1.1.1 Computer Generated Architectural Design

While computers are ubiquitous in architectural design, they are normally used for analysis rather than design generation and exploration. In recent years software has been developed that allows the user to explore the search space of possible designs. Parametric design systems enable the user to create a design and then vary components to explore the search space, an approach that essentially describes the design as a function with inputs that alter the design output. This thesis examines whether aspects of parametric design systems can be introduced into evolutionary design. Structural analysis has also become an important component of computational design. Presenting the designer with information on the loading and stress points of a structure during modelling means the designer is capable of manipulating the structure in order to reduce the stresses.

1.1.2 Evolutionary Computation and Conceptual Design

While evolutionary algorithms (EAs) are predominantly seen as an optimisation technique, their stochastic approach to search means they are capable of producing novel and unexpected results. This ability means that EAs have been applied to design generation problems. Several implementations for evolving conceptual designs have been previously examined. One approach that takes advantage of EAs optimisation ability is non-interactive design evolution based on aesthetic rules [16, 118, 82]. The user weights their preference of a series of quantifiable aesthetic rules such as symmetry, curvature, proportion, etc. The algorithm then generates and evaluates designs based on these rules.

An alternative approach is to allow the user to assign fitness values to the designs that they prefer [44, 174, 153, 18]. Interactive Evolutionary Computation (IEC) was developed as a means of assigning fitness when no objective metric could be defined. Human interaction has allowed EC to be applied to problems such as music and computer graphics, and to act as an exploratory tool as opposed to its primary function as an optimiser.

Both approaches produced interesting results but they also had their disadvantages. The non-interactive aesthetic approach is capable of producing novel and appealing designs but there is a lack of any human intention. Using human evaluation inevitably slows down the algorithm as it creates an evaluation bottle-neck and it is a blunt tool for guiding the evolutionary process.

1.1.3 Grammatical Evolution

Grammatical evolution is an evolutionary algorithm that is based on genetic programming (GP) [140]. It differs from standard GP by representing the parse-tree based structure of GP as a linear genome. It accomplishes this by using a Genotype-Phenotype mapping of a chromosome represented by a variable length bit or integer string. The chromosome is made up of codons, e.g., integer values. Each codon in the string is used to select a production rule from a context free grammar. Production rules are selected from the grammar until all non-terminal rules are mapped and a complete program is generated. The immediate benefits of using this approach is that grammars ensure syntactic correctness, compression and abstraction of information and furthermore allow for domain knowledge to be embedded.

The advantage of using a grammar for design is that it is possible to generate anything that can be described as a set of rules. Grammars are capable of generating strings, mathematical formulas, pieces of programming code and even whole programs. Another advantage of applying GE to design is that generative processes, like the mapping process in GE, are required for the production of designs that can scale to a high level of complexity [85]. There is an additional implicit benefit from using a grammar based approach, grammars can be substituted without any need to change the underlying evolutionary algorithm. This allows, for example, grammars based on different languages but with similar functionality to be easily substituted.

1.2 Aim of Thesis

The focus of this thesis is to increase the amount of direct and indirect interaction available to the designer for evolutionary design exploration. The intention of this aim is to minimise the fitness bottleneck by improving the means of interaction between evolutionary algorithms and human designers. Improving the user interface allows the user's design preferences to be expressed and so GE becomes applicable as a design tool rather than a design optimiser. The system should allow the user's intentions to be expressed indirectly by specifying design constraints as well as directly by manipulating the evolutionary representation of a design.

1.3 Questions

The aim of this thesis is to increase the amount of direct and indirect interaction available to the designer for evolutionary design exploration. To this end, the following questions are addressed:

- *Can objective fitness functions be developed for evolving architectural designs?* Architectural design is a subjective activity but it does have certain constraints. For example, a design must be capable of being built to the right scale and of being able to support its own weight given the structure and materials. Can the designs that do not conform to the explicit constraints be filtered out before being presented to the user through the application of an automatic fitness function? Once the constraints are expressed they will often be of a different nature, e.g., minimising the cost of material used will often lead to minimising the structural properties, which is undesirable. How do we account for conflicting design constraints using such a fitness function?

- *Can search operators be developed that allow the user to directly interact with the design at a genotypic level?* When a user interacts with a finished design, the results are cosmetic changes to the phenotype that are difficult to integrate into the genome. This problem is avoided by permitting the user to manipulate the design at a genotypic level. To do this an understanding is required of how changes on the genotypic scale effect the resultant phenotypic output. One measure of the genotype-to-phenotype relation is locality, i.e., how much a change in the input changes the output [165]. Can genotype operators be developed based on the locality of their effect on the phenotype? Once operators have been implemented based on locality the next question is: Can these operators be applied by the user to explore the design space?

- *How does the design of the interface effect the user's interaction with the evolutionary algorithm?* User interaction with the underlying EA slows down the search process considerably. Can an interface be developed that presents more of the search space to the user? What techniques can be applied to exclude redundant designs? What approaches are available to maximise the number of individuals processed by the user?

- *Can a context free grammar based representation be used to generate architectural designs?* If a representation is to be used for design it must be generative and produce novel results while reducing the overall search space. A grammar allows incorporation of explicit design bias, i.e., domain knowledge that can reduce the search space but is still capable of presenting novel designs.

1.3.1 Objectives of Thesis

In order to address the questions and aims specified in this chapter, there are a number of objectives that must be fulfilled:

1. Survey the state of the art in evolutionary design and IEC.
2. Develop a fitness function that automatically selects instantiable designs from the search space.
3. Perform an analysis of mutation in GE.
4. Create mutation operators that enable different levels of phenotypic change for interactive evolutionary computation.
5. Implement interfaces that enables users to navigate search space.
6. Apply grammar based approaches to design generation.

1.4 Methodology

The subjective nature of design required that different methodologies be used throughout the investigations of this thesis. Non-interactive experiments were first introduced to examine the behaviour of mutation operators.

- *Non-interactive fitness comparisons:* Chapters 4 and 5 uses a Wilcoxon rank-sum to analyse the change in fitness. Chapter 5 compares the difference in best fitness for the components of standard integer mutation whereas chapter 4 compares evolutionary approaches against random search.

- *Non-interactive distance comparisons:* Chapter 6 uses a Wilcoxon rank-sum to compare differences in distance between nodal an structural mutation events on successive levels of derivation.
- *Interactive survey:* Chapters 4 and 6 conduct surveys on user preference of designs. Each query presents two designs side by side. A binomial test is used to conduct an analysis of significance.
- *Interactive target matching:* Chapter 7 asks the user to match a target image. The user results are compared against randomly generated results and analysed by using LOESS with bootstrapping and Wilcoxon rank-sum.

1.5 Contributions

The work described in this thesis has given rise to a number of contributions as described below. Two main strands of research were carried out in this thesis: An analysis of the GE algorithm and the application of GE to architectural design. Accordingly the contributions are placed in the relevant categories.

1.5.1 Analysis of GE

Analysis of the behaviour of mutation in GE: The impact and classification of different mutation events that occur during standard integer mutation is carried out in the experiments in chapter 5. The results show that there are two distinct behavioural components. Integer mutation is decomposed into two component operators.

Ripple mutation: It was shown that mutation events at particular points on the chromosome can alter the meaning of following codons. The result is that different magnitudes of change can occur for a single mutation event. The effect is similar to ripple crossover also witnessed in GE [95].

Analysis of locality: The effect of the two component behaviours of mutation was analysed at different stages of the derivation process. Experiments were carried out to examine the effects in detail in chapter 6.

Nodal and structural mutation operators After examining the component behaviours of integer mutation, it was shown that they had a different effect on fitness and generated changes with different locality. *Nodal mutation* generated high locality changes while *structural nutation* generated low locality changes throughout the mapping process.

Euclidean graph distance: A new measure was devised to analyse the locality of the graphs that were generated by the evolved programs in chapter 6. The measure was used to generate a distance value for comparison at the final output stage of the phenotype that was presented to the user.

1.5.2 Application of GE to Architectural Design

Literature review: A multidisciplinary survey of relevant literature in the fields of architectural design and evolutionary design is presented in chapter 2. Computer generated architectural design, interactive evolutionary computation, design representations and grammatical evolution are covered in detail. Two taxonomies are described in this section. The first taxonomy is a decomposition of architectural design into design exploration and design optimisation approaches. The second taxonomy compared the approaches of different evolutionary design software. A number of research gaps in the literature are identified and stated. A detailed description of the algorithm used in this thesis, grammatical evolution, is given in chapter 3.

Shape generating grammars: Context free grammars were developed that had the capability to generate designs that exhibited hierarchy, modularity and regularity. This was accomplished through the adoption of techniques from the functional programming paradigm, such as lambda functions, mapping and higher order functions.

Structural analysis grammar: A methodology was developed so that stresses could be applied automatically to the evolved structures in chapter 4. This allowed finite element analysis to be automatically performed on the generated designs.

Multi-objective grammatical evolution: A version of the NSGA2 [46] algorithm was adapted for use with GE in chapter 4. Implementing NSGA2 in GE allowed for multiple design objectives to be used to generate the designs.

Design surveys: Two surveys were conducted to analyse the subjective design preferences of experiment participants. The first survey was conducted in chapter 4 and queried the user's aesthetic preference of designs. It was shown that the user's preferred designs that did not meet the optimisation constraints. The second survey in chapter 6 presented the user with two mutation events and inquired which event was most similar to the original. The results showed that nodal mutations were more similar to the original than structural mutation.

Novel user interface: An interface based on animating mutation events was created in chapter 7 to allow the user to direct search during IEC. The results showed that animating the effects of mutation presented a greater amount of the search space to the user than static displays and allowed for evolution to be directed by the user.

Interactivity Experiments: An analysis of user behaviour in chapter 7 showed the users moved from low to high locality operators when evolving towards a target. The difference in the locality of the two different types of mutation event allows the user to both explore the search space and exploit designs

1.6 Limitations

The investigations of this thesis focused on a variety of methodologies that enable GE to be used for interactive evolutionary design exploration. Accordingly, it did not conduct an exhaustive analysis of all possible research avenues. There are several instances of issues worthy of further research that fell outside the scope of this study. Below is a list of these issues.

An exhaustive search of all possible parameter values and operator combinations. The analysis performed on integer mutation focused purely on mutation. While the behaviour of the operators could have been changed when combined with other search operators or at different rates. The aim of this work was to distinguish component behaviours rather than optimise the behaviours for the benchmark problems.

The use of a design tool is not just for target matching. Design is an exploratory process which does not necessarily have a pre-determined outcome. However analysing data based entirely on a user's subjective preference of a design only provides qualitative information rather than quantitative information. The target matching experiments allow for measures to be used to quantify changes, which in turn allows hypotheses to be stated and tested. While the target matching may seem like an artificial usage for a design tool it has commonalities design exploration. An example of this would be a user trying to combine aspects of a previously observed design with their current favorite design.

Only context-free grammars were used. Recently for example probabilistic grammars [81] and tree adjoining grammars [132] have been developed for GE. The experiments in this thesis only used traditional forms of grammar. The aim of this work was to examine architectural design generation by computers. Accordingly, the literature does not give an exhaustive description of the field of architecture. This work also addresses aspects of human interaction with evolutionary algorithms that potentially has value in the field of human computer interaction (HCI), however a detailed examination of HCI goes beyond the scope of this thesis.

1.7 Thesis Summary

Chapter 2 contains a review of computer generated architecture, interactive evolutionary computation and representations suitable for evolutionary design. Chapter 3 gives an overview of grammatical evolution. It describes context free grammars, the genotype to phenotype mapping and the operators used by the grammar to generate new individuals. Together, these two chapters detail of the related research and implementation discussed in this thesis.

Chapter 4 combines a shape generating grammar with engineering constraints to explore whether the requirements of a structure can by used to evolve designs. A multi-objective fitness function is introduced as structural analysis alone does not provide a sufficient fitness function for evolving designs. The results show that the average fitness of the population improves significantly over time. A survey was then carried out to examine whether users found a search space reduced to feasible designs more interesting and appealing. The results showed that they did not. Thereafter the studies focus on increasing user participation rather than automating aspects of the search process.

Chapter 5 examines the integer mutation operator in GE. It is shown that there are two component behaviours of integer mutation. These component behaviours are called structural and nodal mutation and they are defined using formal notation. The behaviour of the two operators is analysed against a series of benchmark problems. The experiments first look at the best fitness result over the course of a run. The second set of experiments analyse individual mutation events and then examine the results for statistical significance. It is shown that the operators have distinctive search characteristics, both from each other and from integer mutation.

Chapter 6 deepens the study carried out in the previous experiment by examining the mutation events effect on locality during the derivation process. Different metrics are defined for each level of the derivation process and the magnitude of change for nodal and structural mutations is compared. It is shown that structural mutation generates a larger change size for all the levels of derivation. In order to quantify the more subjective idea of similarity participants were asked to conduct an online survey. The participants found that nodal mutations produced smaller changes to an individual than structural mutation events.

Chapter 7 explored the application of nodal and structural mutation to individuals in a population by the user. The initial interface produced results that were not statistically significant from random. It was also shown that the user had no intuition of what change to expect from applying a mutation operator. The second experiment allowed the user to apply any mutation event and instead tracked whether it was nodal or structural. The interface animated the mutation events so the user could rapidly observe the changes to the individuals. The results showed that the participants could use the interface to evolve an individual towards a target. The analysis of the user's choice of mutation events showed they initially started by applying structural mutation and then moved to nodal mutation to fine-tune the solution.

Chapter 8 gives an overview of the work done in this thesis and summarises the contributions. The future work that could be done in this area is specified. Appendix A lists the grammar used to generate geometric objects and the grammars used to generate the bridges for the experiments. Appendix B contains the design brief that was presented to the UCD students. Appendix C contains the questionnaires and surveys the experiment participants were required to complete after conducting the user interface experiments.

Chapter 2

Related Research

This chapter presents related work in the field of architectural design. The investigations in this thesis focus on this area as the combination of objective and subjective design evaluation makes it a challenging application for evolutionary computation. The flexible and open-ended representation used by grammatical evolution make it a suitable approach for generating architectural designs. GE allows for the creation of novel designs that are not explicitly defined in the grammar and for increasing design complexity through the use of modularity [150, 81]. Architectural design is also an area of great economic importance. The size of the architectural services industry in the U.S. alone is 40 Billion Dollars [88]. Computational architecture is used extensively for analysis but has only found limited application for design generation. The intention of this thesis is to build on existing approaches and develop new techniques that allow evolutionary design to be used by the architect.

An overview of the structure of this chapter is shown in Figure 2.1. An overview of computer generated architectural design approaches is given in Section 2.1. The application of structural analysis for architectural design is examined in detail in Section 2.2. A justification is given for categorising architectural design into its exploration and optimisation components in Section 2.3. Different approaches to interactive evolutionary computation (IEC) are discussed in Section 2.4 and the distinguishing characteristics between narrow and broad levels of interaction are highlighted. The review then focuses in detail on methods where the user participates directly by applying evolutionary operators in Section 2.5. Finally, a number of relevant representations for generating designs are compared and contrasted, including grammatical representations, in section 2.6.

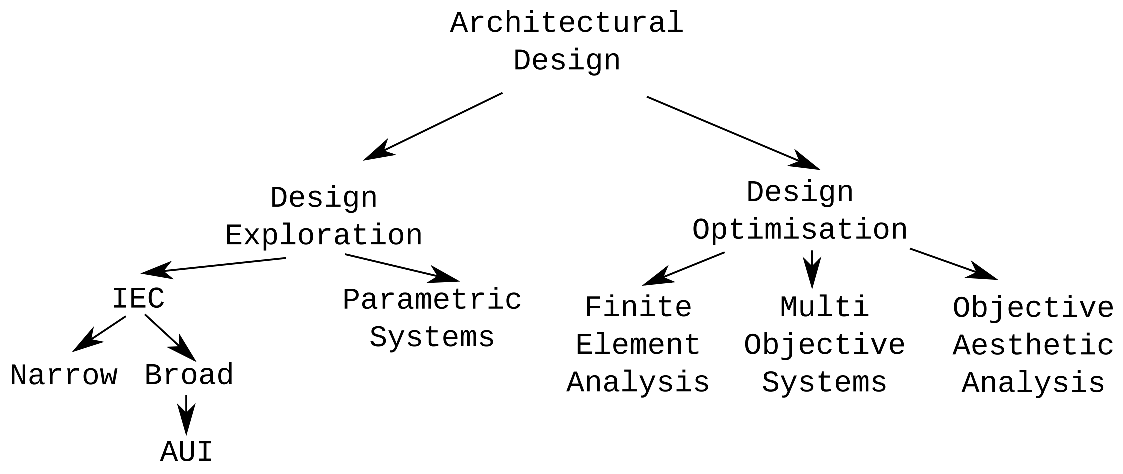


Fig. 2.1: An overview of the related research covered in this chapter.

2.1 Computer Generated Architectural Design

While computers are ubiquitous in architectural design, they are normally used for drafting or analysing designs rather than design generation. Examples of popular modelling software include Sketchup [68], Rhino [129] and AutoCAD [4]. In recent years software has been developed that allows the user to explore the search space of possible designs either through varying a set of design parameters or by generating designs that match specified constraints. These approaches are described in more detail in the following sections.

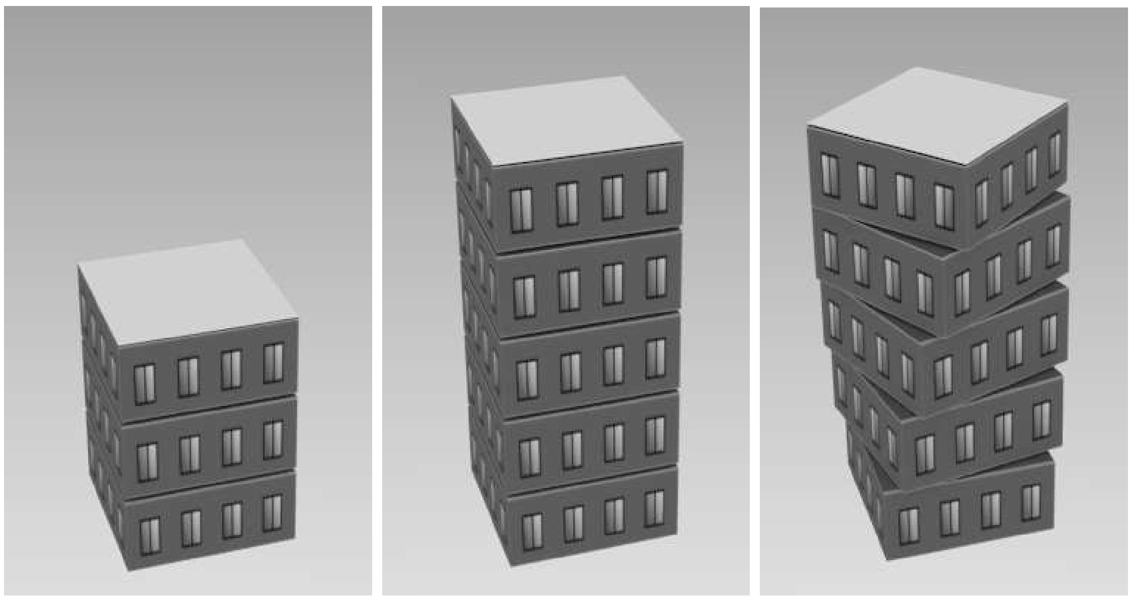
2.1.1 Parametric Systems

A direct approach that allows the designer to explore the design search space is to implement a parametric system. The system uses algorithms to generate components of a design and the user manipulates the variables that are inputs to the algorithms. The user modifies the design or components of a design by changing the input. An example of such a function and the design output of a building are shown in Figures 2.2 and 2.3.

An important aspect of parametric design is that the user observes the effects caused by manipulating a variable in real time. This allows the user to treat the algorithm as a black box and they only concern themselves with the change of output relating to their change of input.

```
def generate_building(floors, width, offset):
    parametric function that generates a building
    floors: total number of floors
    width: width of each floor
    offset: offset in degrees between floors
```

Fig. 2.2: A sample parametric function that generates a building.



(a) `generate_building(3,4,0)`. (b) `generate_building(5,4,0)`, increased floor count. (c) `generate_building(5,4,10)`, increased offset.

Fig. 2.3: A parametric design example with different parameters.

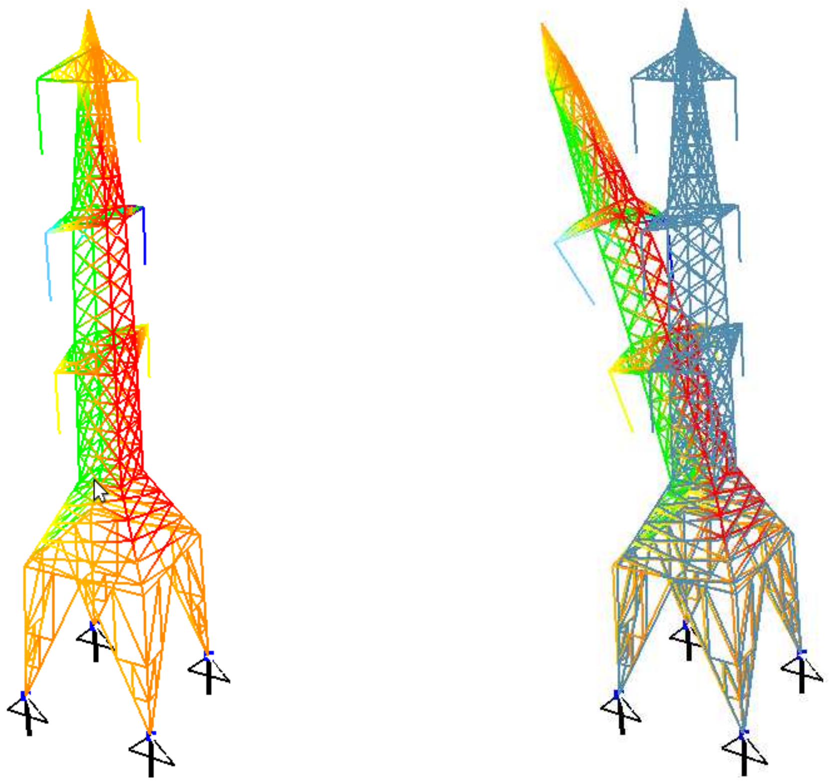
Parametric design tools have now been introduced into mainstream design software. There is the Grasshopper parametric design tool plug-in for the Rhino modelling system [177], Bentley Systems have implemented a program called Generative Components [183] based on the parametric design paradigm and NASA have recently released OpenVSP, a parametric design tool for designing aircraft [134]. Dassault Systems have developed CATIA, a CAD system combined with a parametric design tool. Parametric functionality was introduced to AutoCAD 2010 to allow for algorithmic manipulation of a design.

The parametric systems described above are the primary approach for using the computer as a generative tool to create architecture. As such it has allowed the architects to describe their designs algorithmically rather than directly specifying the final output. The next section examines how computers are also used to analyse architectural designs during the design process.

2.2 Structural Analysis

Structural engineering is the field of analysing and constructing designs that are capable of handling loads. The goal of the structural engineer is to maximise resistance to load while reducing material usage and cost. Computer modelling provides a methodology that allows designs to be easily and quickly analysed in a virtual environment. Simulated forces are applied to a structure and the resultant loads, stresses and deformations can be computed. An example of this is shown in Figure 2.4. The forces on a structure are traditionally calculated using partial differential equations (PDE). While PDEs can be calculated using a computer, a method based on discretisation for modelling stresses is the preferred approach [66, 200].

Finite Element Analysis (FEA) simplifies this process into a more computationally friendly form by converting PDEs into a set of ordinary differential equations (ODE). It does this by dividing a structure into discrete set of elements using a meshing algorithm, such as Delaunay triangulation, and then generates the ODE for each element. This system of equations can then be integrated using an iterative numerical method such as Euler's method or the Runge-Kutta method. FEA has been applied to a range of problems such as heat transfer, electromagnetic potentials and fluid dynamics. FEA methods are applicable to architectural designs if the components of a structure can be represented in a mesh



(a) High load areas shown in red. (b) Scale of displacement increased to highlight deformation.

Fig. 2.4: An electricity pylon simulating a cable break condition.

configuration.

Structural analysis is used to evaluate a design so that weak points can be found and corrected early on in the design process. The most commonly used standalone structural analysis packages are ANSYS [2], STAAD [10] and STRAP [69]. There are also plug-in packages that allow for analysis of the conceptual designs. Robot structural analysis software [5] is used in conjunction with AutoCAD to analyse designs. Scan and Solve [178] allows designs generated in Rhino to be analysed upon creation so that the user can check the structural integrity of their design. There are also open source analysis tools for design post processing such as SLFFEA [107], freeFEM [158] and openFOAM [168] that allow for analysis of correctly formatted meshes.

Combining parametric systems with structural analysis allows the user to make informed decisions about the geometric alterations during the conceptual design stage [84]. EIFForm successfully implemented a parametric design system based on generative components that optimises parametric structures by using a simulated annealing algorithm. The results have been used to design a structure in the inner courtyard of Schindler house [170]. Bollinger et al. [21] have developed parametric design systems that incorporate structural considerations and have used it to generate roofing structures for the BMW Welt Museum, Munich and the Rolex learning centre, EPFL, Lausanne. CATIA was combined with GSA structural analysis software [176] to evolve roofing structures for a football stadium [84].

Finite Element Analysis is also used to provide a fitness value for a structure so that designs can be evolved. These approaches are covered in greater detail in Section 2.3.1. The next section examines how evolutionary algorithms have been applied to generate architectural designs.

2.3 Evolutionary Architectural Design

> "Since design problems defy comprehensive description and offer an inexhaustible number of solutions the design process cannot have a finite and identifiable end. The designers job is never really done and it is probably always possible to do better." [106].

For an architectural design to be created several different constraints must be satisfied. Architects evaluate all aspects of the design, from broader issues of internal and external relationships to more detailed aesthetic measures such as material use, texture and light. Engineers ensure that a design is capable of meeting its structural requirements by evaluating the strength, efficiency and integrity of the structure. Design is not solely an optimisation process and trade offs must be made at all stages of design.

Simon [172] describes how human decision making processes are based on "satisficing". Humans rarely have complete information, are limited by their cognitive ability and do not have an infinite amount of time to reach a conclusion. Accordingly they do not attempt to find the global optimum solution but instead find a solution that satisfies some criteria while sacrificing others. The same problems occur during the design process. It is inefficient for a human designer to maximise all objectives simultaneously so instead they seek to find adequate solutions to the largest number of objectives. The final design is not the global optimum of all the design constraints but normally where several of the constraints have met adequate levels.

Although evolutionary algorithms are traditionally viewed as optimisers their search behaviour is more like that of a satisficer [14]. Instead of directly finding the global optimum, it finds improvements on existing solutions which then propagate throughout the population. Although the algorithm is capable of finding the global optimum, this is not guaranteed. The open-ended nature of design problems make evolutionary approaches particularly suited to design problems as there is always the possibility of evolving an improvement on an existing design. The stochastic nature of an evolutionary algorithm means that the algorithm is not limited to design optimisation but can also be used for exploring the search space of possible designs. Fogel et al. [59] states that the inherent randomness of the EA can create results that surprise us and introduces aspects of creativity to the evolutionary process [59].

Whether an unthinking and blind process can be considered creative is a difficult question due to the anthropocentric nature and subjectivity of creativity [13] and is not addressed in this work, but EAs have proved themselves capable of generating "novel solutions that are qualitatively better than previous solutions" [64] and of generating human competitive results [136] [11].

Another issue of major importance is the form a design should take. Before a design can be optimised it must first be instantiated. Normally a designer will explore the search space by developing several conceptual designs that are uninhibited by the expected functionality. This design exploration phase is of paramount importance to the final outcome of the design process. The stochastic nature of evolutionary algorithms is a suitable approach to design exploration. An EA has no preconceived notions of how the components of a design should be combined. This freedom allows the algorithm to generate designs that combine components in novel ways.

Design exploration requires that the representation being evolved is capable of generating novel and innovative designs that were not explicitly encoded in the representation. Although this is problematic for traditional fixed length genetic algorithms, the open-ended nature of genetic programming allows for the continual recombination of existing components in novel ways. The module based approach of GP is appropriate for design exploration and exploring different forms in the search space.

A sample of the existing approaches are shown in a matrix in Figure 2.5. The y-axis categorises the applications based on their focus on the form versus the functionality of a design. Although form and function are not mutually exclusive, they are two common measures upon which a design can be judged. The x-axis indicates the means of evaluation used by the algorithms. Non-interactive objective-based fitness evaluation is on the right of the matrix while subjective user evaluation is on the left of the matrix. In the following sections the application of evolutionary algorithms for design optimisation and design exploration is examined.

2.3.1 Evolutionary Design Optimisation

The capability of an evolutionary algorithm to optimise a solution has been applied to design problems since their inception. Rechenberg [162] demonstrated one of the earliest practical applications of evolutionary computation by optimising a two phase flashing nozzle by combining differently shaped cross sections of pipe. Rechenberg's approach eventually developed into the field of evolutionary strategies. The uptake in the use of computers for performing analysis of architectural design meant that the same evaluation process could provide a fitness function for an evolutionary algorithm. Computer evaluation meant that a designer could develop a conceptual design and then allow to the computer to further refine it.

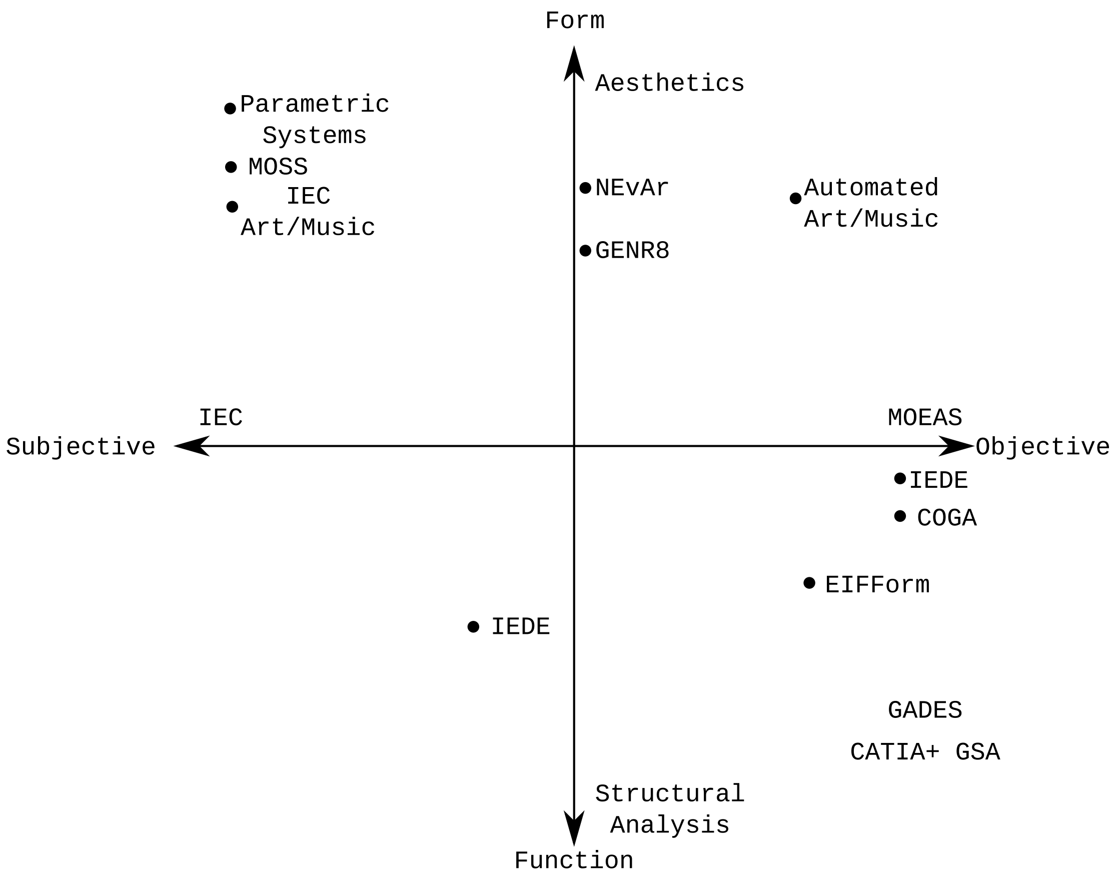


Fig. 2.5: A matrix displaying the focuses of different design software.

Structural Optimisation

The use of computers for structural analysis and FEA meant that they could also be used to provide an objective fitness value for an evolutionary algorithm. Accordingly, there has been a large amount of work in this area [99]. The computational cost of structural analysis meant that early papers focused on greatly simplified structures, such as two dimensional trusses [83]. As computational power increased, so did the scope of the applications. Structures such as bridges [189], electricity pylons [171], and even whole buildings [98] have been optimised using EC.

Structural optimisation is classified into three categories [91].

- Topology: The overall layout of the system
- Shape: The optimal contour for a fixed topology
- Sizing: The size and dimensions of the components

The different focus of the categories in relation to bridge design is shown in the following Figures. Topology optimisation is shown in Figure 2.6, shape optimisation is shown in Figure 2.7 and size optimisation is shown in Figure 2.8. The categories relate to the three major stages of the engineering design process, i.e., the overall layout is chosen at the conceptual stage, the shape of the structure is optimised during the embodiment stage and the final sizing is optimised during the detail design stage [99].

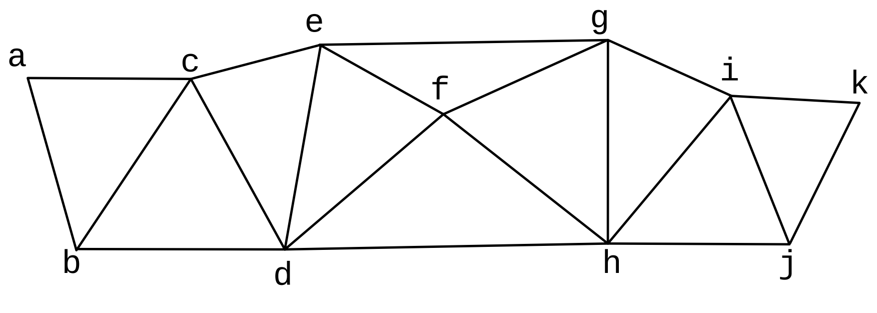


Fig. 2.6: Topology optimisation, how the nodes are connected.

The categories form a hierarchy for the structural engineer. A topology must be fixed before shape optimisation can be carried out. Accordingly, the components must be chosen

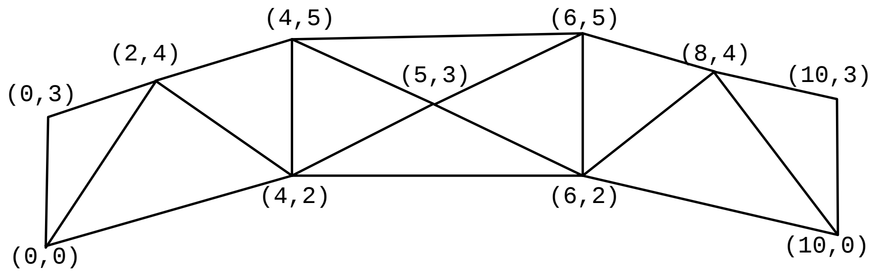


Fig. 2.7: Shape optimisation, how the nodes are positioned.

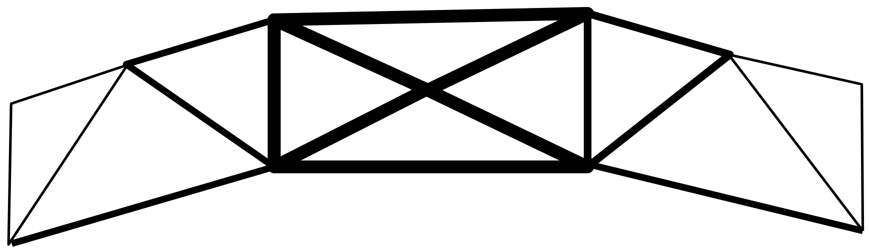


Fig. 2.8: Sizing Optimisation, the diameter of the edges.

before their sizing can be optimised. An example of optimising topology is the paper by Bohnenberger et al. [20] where the leg of an electricity pylon is evolved. The edges of the graph are subdivided and nodes added to increase the overall strength of the pylon leg.

An example of topology optimisation that is not graph based is the Voxel based work by Zuo et al. [202] where volumetric pixels were added and removed from a truss structure to maximise the stiffness of a Michell truss. The work by Shea et al. [171] is an example of placement optimisation. Their approach used simulated annealing to optimise both the number and placement of nodes in a transmission tower.

Topping and Leite [189] provide an example of sizing optimisation where the fitness was a function of the total volume. Their approach used a system of parallel GAs to optimise the sizing of a cable stay bridge. The overall topology of the structure remained fixed while aspects such as the cross sectional area of the girders and cables were allowed to vary.

Engineering constraints play an integral role in the design of architecture. If a system is going to be designed so that it is capable of producing feasible designs, it must incorporate structural analysis into the design tool. This will allow the user to make an informed decision about the structural integrity of their design and provides a fitness mechanism for evolving structurally sound designs.

Multi Objective Fitness

When there is a conflict between design objectives, it means that a trade off is required between them. An example of such a conflict during the design process would be minimising the amount of material used to build a structure while trying to reduce the load on the structure's components. When optimising designs with multiple constraints two approaches can be taken. The constraints can be weighted and summed, thus reducing a multi objective problem back to a single objective problem, this simplifies the problem but the correct weighting must be manually chosen.

The alternative approach is that there is no globally optimal solution. Instead the solutions can be placed on a "front" composed of the non dominated solutions in the population. The set of non-dominated solutions are solutions that are better than the rest of the population for at least a single constraint and at least equivalent for all other constraints. The set of non-dominated solutions form what is called the pareto front. Substituting one solution for another on the pareto front will always sacrifice quality for at least one constraint, while improving at least one other.

EAs have proved themselves a successful methodology for optimising multi-objective problems. Esbensen et al. [56] used a GA for optimising the floor plan of an integrated circuit. This problem has multiple conflicting objectives such as routing congestion minimisation and delay minimisation that require compromises. Their approach displayed graphs that showed the progression of the algorithm so that the user could navigate the pareto-front and weight the trade-offs. The constraints are path-delay, routing congestion, area minimisation and aspect ratio.

The design of an antenna requires a trade off between larger bandwidth versus higher gain. Koulouridis et al. [102] builds a pareto front using a novel approach, sequential categorisation of the designs for each objective and then only selecting feasible designs that fulfill the criteria. The graphs of the pareto front were color coded based on the size of the antenna which allowed the user to visually exclude improperly sized antennas from the pareto front.

Bentley [11] developed the evolutionary design system, Genetic Algorithm Designer (GADES). The system was tasked with optimising the room layout for a hospital floor plan. The criteria for the designs were specified, such as locating the x-ray departments and eye departments should be in darker areas of the building while the ward should be brightly lit. The evolved designs were evaluated by each criteria and a weighted total of the fitness values generated the fitness value.

A more advanced approach to calculating fitness for multiple objectives is the ranked-sum method used by Coia and Ross [39]. Their system developed conceptual building designs using a shape generating grammar combined with CityEngine software [53]. The designs were evaluated by geometric rules such as surface area, the number of unique normals and building height. The normalised rank for every individual in the population was calculated for each metric. The ranks were then summed to calculate the fitness for the individuals.

2.3.2 Evolutionary Design Exploration

Characterising design as a search problem oversimplifies the design process. A designer does more than optimise, they impart their subjective preferences and give artistic form to their designs. Evolutionary design systems should be seen as generative processes that are able to evaluate their own output [92]. If the representation used is sufficiently flexible, the EA has the ability to create surprising and unexpected designs. The representation used by grammatical evolution is especially suited to design exploration. Grammatical evolution allows for the definition of components that can then be recombined in novel and unexpected ways. This methodology means that, with a suitably expressive grammar, an endless number of forms can be generated [14].

EAs have the additional advantage that they have no subjective bias of what a design should look like. Nor does it suffer from a functional fixedness [52], i.e., an inability to use components of a design in a way that they were not originally intended. An example of this behaviour is the use feedback loops instead of boolean logic to generate functionality in a VLSI circuit [188].

One of the earliest researchers to adapt evolutionary processes to explore architectural designs was John Frazer. He pioneered early evolutionary design systems such as rep-tile [60] which incorporates notions of growth and evolution and investigated the fundamental form-generating processes in architecture. Their work considered architecture as a form of artificial life, and proposed a genetic representation in a form of DNA-like code-script, which could then be subject to developmental and evolutionary processes in response to the user and the environment.

There are two approaches to evolving aesthetically pleasing designs, evaluating the designs based on objective measures or evaluating the designs based on subjective user preference. The next two sections will discuss both approaches in detail.

Objective Aesthetic Exploration

An automated approach to evolutionary design exploration is to evaluate the designs using objective aesthetic measures. Either the user calibrates the measures manually or an algorithm attempts to learn the correct settings from previous user selections. Automating the evaluation process allows the algorithm to evaluate much more of the search space and reduces the fitness bottleneck (discussed in greater detail in Section 2.4).

The use of GAs in design by Bentley [16] and [17] showed early on that GAs could generate complex designs by using a mapping process and that these designs could by optimised by both structural and aesthetic objectives. The system focused on the geometry of three dimensional solid objects and used a multi-objective fitness function for specifying the desired output. The design was evaluated by attributes such as size, mass, area of flat surface and stability.

The work by Machado et al. [118] showed that adaptive automatic critics were capable of evaluating stylistic content. Their system used a feature extractor and an evaluator. Separating the evaluator module allowed for it to be applied to different domains such as music and visual art. The evaluator consisted of a neural network trained by back propagation to categorise different artistic styles. Their results showed that the critic was capable of learning differences in style from different domains. This meant that if the preferred style of a user could be learned the algorithm could automatically evolve designs of that style.

A novel approach by Ventrella [192] used both objective and subjective measures for evolving appealing gait animations. In an effort to capture the "expressivity" or aesthetic preference of the animator, the animator forms part of the evaluation process. The gaits were initially evolved using objective fitness functions such as locomotion distance and head height while designs were penalised for head collisions and excessive body flipping. The user then selected their preferred gaits from the optimised individuals for further evolution. The objective functions reduce the search space to plausible designs thus reducing the fitness bottleneck

A simplified approach to obtaining a user's stylistic preference is the work by Tsutsumi and Sasaki [190]. The algorithm developed an optimum roof design using the finite element analysis and a genetic algorithm. Aesthetic fitness is provided by a neural network (NN). A survey is administered to the user and their preferences recorded. The results of the survey are then analysed by a NN to learn the user preference The neural network then acts as the fitness evaluator in place of the user.

A design system for generating abstract conceptual design was developed by O'Reilly and Hemberg [153]. The system uses GE and Hemberg Extended Map L-Systems (HEMLS) to generate forms. Each design is evaluated by a series of metrics; symmetry, undulation, size, smoothness, etc. The user is able to weight the respective fitness for these metrics according to their preference. The user could also influence the algorithm by adding tropism before the evolution process began.

User preferences through weighting was also used by Parmee [156]. Their system added interactive weighting to their existing multi-objective design tool. The user first sets the objectives that require investigation. Although initially all objectives are equally weighted the user can their preferences for each of the metrics. This work was extended by Machwe et al. [120] into the interactive evolutionary design environment. Their approach added user interaction to a multi-objective design system by asking the user to weight the fitness functions. Machwe and Parmee [119] then developed this approach further by adding aesthetic rules for symmetry, slenderness and uniformity to the fitness evaluation

While objective fitness evaluation has shown itself capable of generating interesting and novel designs, the intention of the work described in this thesis is to act as a design tool that is used directly by the designer. Accordingly the next section examines approaches where the user participates in directing the evolutionary algorithm.

Subjective Aesthetic Exploration

The problem with objectifying aesthetic considerations is that it removes the designer from the design process. If an evolutionary algorithm is to be used for aesthetic design it must allow the designer to interact with the algorithm. As stated by Bentley and O'Reilly [15], "the designer must be allowed to design and the algorithm should act as a creativity enhancement". For this to work there must be a balance of control for the designers, let them interrupt and change and don't get it to do everything. "Best utility can be achieved from systems that enhance the designers inherent capabilities" [43].

O'Reilly and Ramachandran [154] described a form design tool that instead used GP and interactive rating to judge the form of the design. Their focus was on creating a tool for form discovery, not just optimisation and it attempts to use the architect's own language. This approach was further developed allow the user to influence the design by controlling the environment as well as the fitness function. O'Reilly and Testa [151] described MOSS (Morphogenic Surface Structure Generator) that grows designs using Lindenmayer systems in conjunction with repellors and attractors.

The functionality of interrupt, intervene and resume (IIR) [15] was built upon these existing systems in GENR8 O'Reilly and Hemberg [153]. GENR8 allowed for increased user input by adding tropism and adjusting the weights of the evaluation criteria like in their previous work but also allowed the user to interrupt the design process. Once the design process was stopped the user could change the weighting for evaluation constraints and then resume the evolutionary process. This approach was also used by Bezirtzis et al. [18] in which they used a GA to evolve the parameters for the structure of one person vehicles. User interaction was provided using sliders to vary the parameters for the purpose of fine tuning the search.

The user interaction with evolutionary algorithms that have just been discussed only focus on architectural design applications but there exists a much larger body of research in this area. As the aim of this work is to maximise the user input to the design tool, an overview of interactive evolutionary computation (IEC) when applied to different problem areas is given in the next section.

2.4 Interactive Evolutionary Computation

Human interaction was originally introduced to the evolutionary process for problems where no objective fitness function could be found. Interactive fitness functions have allowed EC to be applied to problems that have aesthetic considerations such as music and computer graphics, This allows EAs to function as an exploratory tool as opposed to being limited to optimisation.

One of the earliest attempts to introduce human evaluation was the work by Richard Dawkins called Biomorphs [44]. This work simulated the evolution of 2-D branching structures made from sets of genetic parameters, where the user selects the individuals for reproduction.

After Biomorphs there was an increase of research in the field of both interactive genetic algorithms (IGA) and interactive genetic programming (IGP). The seminal paper by Karl Sims [174] showed that basic user interaction was capable of creating complex and beautiful artwork. Sims used human interaction to create images, three-dimensional textures, and, by adding an extra dimension for time, animation. IEC has since been applied to fields as diverse as music generation [19, 127, 40], anthropomorphic symbols [51], 3D lighting [3], logo design [143], 3D image generation [135], and aircraft frames [157].

IEC has been applied as a supplementary tool in the design of various aspects of the design of vehicles, bridges, aircraft and other large-scale engineering projects. [120, 24, 119, 121]. Design problems have a subjective aesthetic element in their evaluation that is difficult to define in an algorithmic fitness function. This has led to the integration of user evaluation in form design tools [154, 15, 18].

The primary problem with user interaction is the “fitness bottleneck” [19]. An algorithm can evaluate individuals in parallel and is limited only by the speed and number of the processors. Humans, conversely, can only process small amounts of input that is limited by their perceptual abilities and can only assign fitness to individuals serially. Miller [130] showed that humans have difficulty handling more than approximately seven quantized measurements of information. This is true regardless of sensory modality. The result is that user input slows the process of the EA due to the limited number of evaluations that they can perform.

The subjective nature of human preference can also confuse an evolutionary algorithm. Dorris et al. [51] used IEC to evolve anthropomorphic men to symbolise different emotions, anger fear joy and sadness. They found that many of the participants didn’t agree on the results and that the resulting designs would not have passed ISO standards. Breukelaar et al. [25] compares human interaction against an ideal user in colour matching. The results showed that the human input is extremely noisy and can easily confuse the algorithm. Their work also showed that this noise cannot be simulated by adding randomised noise.

There are many different implementations that vary the level of interaction and the amount of information provided to the user. How the user interacts with the algorithm is of paramount importance. In his survey paper Takagi [186] broadly splits IEC into two different categories based on the level of interaction the user, narrow and broad.

Narrow IEC (NIEC) limits the user input to applying fitness evaluations on individuals in the population. The applications described above are examples of NIEC. Providing a fitness value for the individuals allows the algorithm to evolve the individuals with a high fitness. The questions that arise with NIEC is how best to present the individuals to the user for evaluation and how many levels of fitness value should the user be allowed to assign.

In an effort to ease the burden on the user and reduce the fitness bottleneck, several different approaches have been taken. Ohsaki [137] interactively evolved facial expressions. They compared differently quantized sets of discrete fitness and examined how the evaluation levels available affected convergence. They found that a smaller number of categories can significantly reduce the psychological stress of the human operators while not having a large impact on convergence.

While this shows the use of discrete fitness values reduced user burden while still allowing the algorithm to evolve improved solutions. The user is still limited to evaluation. In the next section the broader approaches to IEC are explored in more detail.

2.4.1 Broad IEC

While improvements to the evaluation interface have been shown to reduce evaluation times there have been other attempts that broaden the user's interaction beyond that of evaluation. Broad IEC can be accomplished by automatically generating a fitness function from the user's preferences or allowing the user to guide the algorithm through different means.

There have been several approaches to automatically intuit the user preferences based on their previous choices and so speed up convergence. Costelloe and Ryan [40] used tree based genetic programming to learn a user's historical preference data and then use this information to generate pleasing drum loops. Baluja et al. [7] used a GA to generate two dimensional gray-scale image and the user then selected the images they found most aesthetically pleasing. The user's preferred images were then used to train an artificial neural network which could then be used to generate images of a similar theme.

Instead of building an automatic fitness function based on previous user preference, several approaches augment user selections with a complementary fitness function. Gu et al. [70] uses a General Regression NN to discover the intentions of the user and reduce fitness bottleneck while evolving architecture. In a similar approach Ohsaki and Takagi [138] used a neural network (NN) and a Euclidean distance measure applied to fitness results to order the individuals before presenting them to the user. Llorà et al. [112] used partial ordering, induced complete ordering and support vector machines (SVM) to speed up rate of convergence of IEC. The aim was to produce a synthetic fitness from the subjective user fitness. The results on a one-max problem showed it increased speedup seven times.

While broad IEC techniques have proved themselves on several applications,they may not provide a benefit for design exploration. If the user does not know in advance what they are looking for, how is a predictive algorithm meant to deduce their preference? The approach discussed in this thesis utilises user evaluations to direct the evolutionary search but also combines it with lower level user input by allowing the user to manipulate the genome directly. The approaches that allow the user to directly interact with the evolutionary algorithm are now examined in greater detail.

2.5 Active User Intervention

Approaches that increase user participation in the evolutionary process through means other than evaluation are categorised as active user intervention (AUI) [186]. Several successful methodologies have been used to increase user participation.

2.5.1 Online Knowledge Embedding

Online knowledge embedding (OLKE) [185] provides a mechanism for directing the algorithm through hints, ideas or intentions. The user highlights components of a design that they think have high fitness and the genes relating to these components are then fixed, which reduces the search space.

An example of this is the tracking a criminal suspect through face-space with a genetic algorithm [35]. The main advantage of using IEC for this task was that it relied on human recognition instead of human recall. Identikit images that are normally used for criminal investigations were evaluated by the user. There were 10 discrete values for rating the likeness. If a user felt that they had matched a particular feature they could set it so it would stay fixed for the rest of the run. OLKE is only possible if each component of the output maps directly to a particular gene.

Hart [76] developed an evolutionary art that used a novel form of tree alignment to create better genetic dissolves. The user was given fine grained control of the mutation operator and was allowed to select or mask different types of mutation. They could also adjust the relative probability of mutation and bias the mutation rate depending on tree depth. This work is notable for letting the user apply their own intuition on the operator.

SBART [191] a 2D picture generator, enables the user to break parts of the population into different sub-populations. Each sub population evolves separately from their neighbours. Individuals may be saved and introduced into different populations, thus allowing individuals to be treated as "digital amber" [163], a metaphor for how the genetic code of an individual organism is preserved in the computer after a run is terminated, and whereby it may be reintroduced into future populations or reused in a different run entirely.

The approach of grouping and categorizing interesting individuals was also used in the evolutionary art tool NEvAr [117]. During the exploration stage of the selection process the chosen individuals are saved in a separate gallery. The user may then categorise the individuals as images that should be discarded, images that may be useful, images that should be further refined, and images that are artworks that fully satisfy the user's aesthetic criteria.

2.5.2 Visualised IEC

Visualised IEC (VIEC) collapses a multi-dimensional search space into a 2D representation. The individuals are then mapped to the 2D space and presented to the user. The user is able to observe the distribution and fitness of the population and direct the search to particular parts of the search space, thus combining both evolutionary and human search techniques. VIEC has been shown to dramatically improve convergence. Hayashida and Takagi [77] used self organising maps (SOM) to collapse an n-D Schaffer function into a 2-D representation and Hayashida and Takagi [78] compared different methods for mapping an n-D function to a 2-D representation. The problem with VIEC is that a meaningful mapping from n-D to 2-D space must be performed and the topological relationships must remain intact.

2.5.3 Human Based Genetic Algorithm

Human based genetic algorithms (HBGA) enable the user to apply low level genetic operators such as mutation, initialisation, selection and crossover to the population [101]. Using humans is useful in problems such as evolving natural language statements, where it is hard to design efficient computational operators. HBGA requires that an individual in the population can be understood by the user and that the operators perform in a manner intuitive to the user.

Hyper-interactive evolutionary computation (HIEC) extends HBGA by giving the users access all the genetic operators [27]. HIEC treats the operators as a tool set for the user. Additional operators such as duplicate, delete are available to the users. The approach taken in this thesis is similar to HBGA in that the users choose when and where to apply mutation operators. The difference is that the users are presented with the consequences of applying an operator and they select the change they want.

McDermott et al. [123] compared several techniques for synthesising a target sound. One approach implemented a sweeping interface that allowed the user to interpolate between a population of three individuals. The interface provided direct feedback to the user as they heard the sound created by the interpolation of the individuals. The results showed that the interface provided a benefit in some circumstances, such as specifying timbre.

2.6 Representation for Design

Evolutionary algorithms do not have access to the entire problem space, only the solutions that they are capable of generating. In this case, this limits all hypothetical exploration to a subset of all possible designs. This point was further elucidated in the thesis by Jones [93]. The representation cannot be considered independently from the operators that are used to navigate the search space. A representation is the combination of an encoding and the operators as illustrated in Figure 2.9. As such the EA does not explore all solutions in the representation space, only those that it can reach with the available operators.

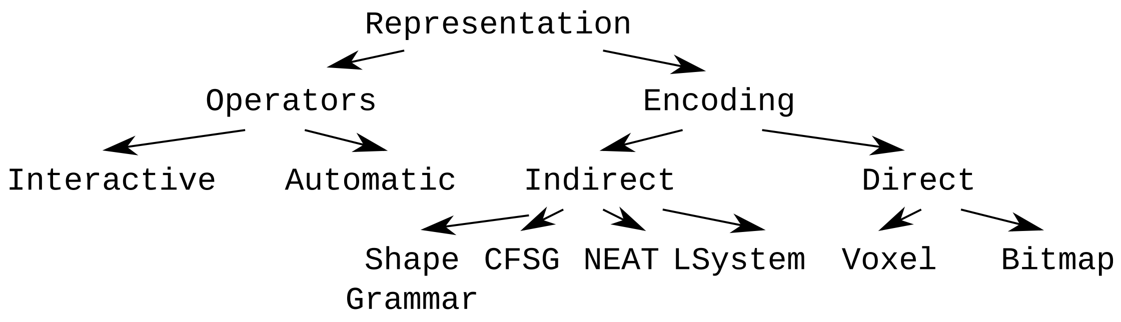


Fig. 2.9: Different components of the design representation.

The next section now discusses different choices of representation and operators for evolving designs.

2.6.1 Encodings

To evolve architecture, a technique is required to generate evolvable shapes. The search space of all possible designs is infinite [106]. The first approach to limiting the search space that must be searched is to limit the representation. The limitations of early GA systems was that they could not easily extend or grow their representations. Traditional Evolutionary Strategies (ES) and Evolutionary Programming (EP) use a one to one mapping from the genotype and the phenotype and so can only be used to evolve components of a design that can be represented as integer strings. The same is true for GAs if no additional mapping is used [20, 45].

A direct mapping means that the complexity of the design is proportional to the length of the genome. Anything more complicated requires the genotype to be extended and so the search space increases. Using the integer phenotype of a GA alone is not a sufficient representation for complex designs. Bentley [14] compared GA's with an explicit mapping and GP representations against ES and EP representations and found that a mapping process enabled the generation of more complex patterns. A sufficient representation and is necessary to generate complex designs.

If a representation is to be capable of generating complex designs it must exhibit modularity, regularity and hierarchy [86]. An important aspect is to have a component based representation [14]. Fixed length representations can optimise but are limited in their exploration. Components can allow for more creativity in knowledge lean environments. ES and EP have no mapping, they work directly on the solution. Component based solutions are required to see evolutionary "creativity". Unconstrained, knowledge lean environments help this process.

A mapping process is required to generate complex designs from integer representations. The effects of different types of mapping was explored by Bentley and Kumar [12]. The work compared different growth processes: as written by the programmer (external), a component based approach based on GP (explicit) and a combination of a GA using a cellular automata as the environment (implicit). The results showed that implicit encodings performed better but that both explicit and implicit encodings could scale to form complex designs. The following sections examine both direct and indirect encodings for generating designs.

2.6.2 Direct Encodings

Direct encodings have a one-to-one correspondence between the genotype and phenotype, i.e., the genotype is the same as the phenotype and the evolutionary operators are applied directly to the output. While there is normally some mapping process, such as turning a linear genome into a two dimensional plane, this is applied after the output has been produced rather than rules mapping the genotype to the phenotype. Direct encodings have the advantage that they are easy to implement and can be evolved using fixed length GAs. Two direct encoding examples are now examined.

Bitmap Encoding

One of the simplest representations is the bitmap representation. By mapping each bit or codon in the genotypic representation of an evolutionary algorithm to a coordinate position of a pixel on a 2-D plane, an image or design can be evolved. Two dimensional examples of bitmap design evolution would be evolving the topology of a Mitchell truss by adding and removing elements [202] and developmental approach to evolving a french flag[131]. Devert et al. [50] developed Eiffelblob which used a developmental approach to optimise each beam as it is loaded and stressed. It performed actions at a geometric level on a 2-D graph structure.

Voxel Encoding

An example of a voxel representation defines volumetric pixels in a three dimensional space. An example of such an approach is the work by Baron et al. [9] which generated components for shape optimisation. Voxels allowed designs to be generated with no prior specification of the topology while still allowing geometric constraints to be imposed. The designs were evaluated using finite element analysis and then their topology was further optimised.

The disadvantage of direct encodings is that they do not easily allow for reuse or for modules to be defined. The encoding also scales poorly. The chromosome grows linearly with the size of the design that is being represented. As such a better approach is to have a mapping process to generate more complicated structures.

2.6.3 Indirect Encodings

An indirect encoding means that the genotype uses a mapping process to generate a phenotype. An example of this is codons being used to select rules from a grammar to generate a sentence. The advantage of using a generative encoding is that it allows a highly compact encoding to generate complex designs. The separation of the search space (genotype) from the problem domain provides a level of abstraction so the same EA may be applied to different problem domains.

Indirect encodings can also easily have repetition of components that work, instead of having to find out everything again from first principles. Modules and reuse enable the designs to become more complex. Another advantage of using the modular structure in the form of rules in a grammar is that it results in a compressed representation.

Shape Grammars

Shape grammars are a production system that is capable of generating geometrical shapes in two or three dimensions [181]. This formalism uses three types of rules, a start rule, at least one transformation rule and a termination rule. The transformation rules are applied to the start rule to generate shapes until a termination rule is reached. Shape grammars have had several successful applications for generating designs. Koning and Eizenberg [100] generated prairie houses in the style of Frank Lloyd Wright by starting at the central fireplace and building around it with variations on his tradition design. A shape grammar for describing and generating coffee maker designs was set out in [1] and there have been several shape grammars for capturing the aesthetic signature of classic designs such as the work by Pugliese and Cagan [161] on Harley Davidson motorcycles and the work by McCormack et al. [122] on Buick cars.

While shape grammars have a great expressive power, there is an inherent difficulty with implementing shape grammars on a computer. The computer must be able to recognise emergent shapes so that transformational rules may be applied to them. Computer vision algorithms are required to categorise these emergent shapes. The problem is explained in [65]. There are three components required to implement shape grammars on a computer. An interpreter is required to apply shape grammar rules, a parser to deduce if a shape is in the shape grammar, and an inference program that given a set of shapes would infer the rules that created them.

Sub-shape recognition has been shown to be NP-Hard [199]. In an effort to simplify the sub-shape recognition problem, Yue et al. [199], uses a graph like data structure that makes it easier to detect sub-shapes. Another approach [187] used a grid system and registration points for detecting shapes but the implementation limited the designs to 2-D rectangular shapes.

Lee and Tang [108] used parametric grammars and labels on a GP-GA hybrid to evolve variations on the design of a camera. Their approach used labelling to simplify the shape grammars and maintain stylistic consistency. The user was allowed to interact with the design process by manipulating the parameters during run time.

Context-Free Shape Grammars

Grammars are a simple way of describing a generative encoding that allow for syntactical correctness, compression and domain knowledge embedding [81]. A grammar defines everything that can be said in a language. A correctly formed grammar ensures that every completely mapped individual is valid. As well as ensuring syntactic correctness, grammars allow domain knowledge to be embedded in the system. By constraining the system to particular constructions or by calling predefined functions, the user can manipulate how the individuals generated by the grammar will behave.

Whigham [194] showed how a grammar can allow an evolutionary algorithm to be biased towards particular solutions. Whigham listed the biases that can be applied by a grammar as:

- The problem representation
- The operators used to search the representation space
- The structural constraints of the representations
- The search constraints when manipulating the representation
- The criterion used to evaluate proposed solutions

Grammars are a suitable choice for design representation for the reason listed above. McDermott et al. [125] compared shape grammars with context free grammars for design generation. It was shown that CFG grammars are easier to manipulate and eliminate much of the ambiguity present in shape grammar mappings-terminals. For the reasons listed above grammars have been used in several approaches to generate designs.

The work by Wonka et al. [196] used a grammar based approach for rapidly generating feasible architecture. By separating the generation process into split and set grammars they could generate varied architectural structures that still had aspects of regularity. Set grammars specify the overall layout of a design while split grammars break up an object into smaller interchangeable components. The results generated by the split grammar would then be used as components by the set grammar.

Coia and Ross [39] combined a context free grammar with CityEngine [53] for generating three dimensional buildings. The grammar applied rules such as extrude, rotate, split scale and insert, to a starting component to generate connected yet varied designs. The buildings were evolved using objective aesthetic constraints such as the number of unique face normals, overall surface area and overall height.

Machado et al. [116] used a graph based evolutionary approach combined with context free design grammars to evolve images. Context Free Art [41] is image generation software that uses context free design grammars to generate shapes and patterns. In this work the genotype was a well formed context free grammar that was represented as a graph. The graph used nodes to encode for non-terminals and the edges were annotated with the parameter values being passed to the non-terminals. The experiments used both objective and subjective fitness tests.

O'Neill et al. [149] used CFGs in conjunction with grammatical evolution to generate two dimensional shapes. An objective fitness function and a specific target was used to automatically evolve designs. This work was furthered in O'Neill et al. [150] to make three dimensional designs for a smoking shelter. The output was rendered using Blender [180] and a user interface was provided for interactive fitness evaluation. The results attained were limited by the fact that many designs in the grammar were not connected to each other.

In an effort to create more coherent designs and also to avoid problems of infinite recursion in a grammar, higher order functions (HOF) were introduced. Currying higher order functions, i.e., setting a function argument to a fixed value, allows recursion to be used without having to worry about non terminating recursive calls Yu [198]. McDermott et al. [126] incorporated this approach into a CFG. Combining higher order functions and lambda abstraction meant that the problem of connectedness in a generated design was addressed and it also allowed for more modularity, regularity and hierarchy in the resulting designs.

NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies (NEAT) is a methodology that allows for neural networks to be evolved [179]. It uses a genetic algorithm to evolve the weights and structures for a NN. Initially the NN is constrained to simple perceptron configuration and is allowed to increase complexity over time. A specialised version of NEAT called hypercube-based NEAT (hyperNEAT) has been used to evolve designs and art. HyperNEAT builds on compositional pattern producing networks (CPPN), a form of ANN, to output designs.

The first application of NEAT for art was Picbreeder [167]. This used the CPPNs to output two dimensional designs. The fitness for the algorithm was provided by an online community of participants. This work was developed further into three dimensional objects [38]. Endless forms is an online collaboration that uses CPPNs to develop shapes which can then be printed.

While CPPNs are capable of generating complex and beautiful designs, the complicated mapping process complicates user interaction. Woolley and Stanley [197] tried to re-evolve targets using an automatic measure, and failed to match them. The reason for this was that the original evolutionary process had to evolve small components and then combine them as opposed to evolving everything in one go.

Lindenmayer Systems

A Lindenmayer system (L-system) is a parallel rewriting system that was developed to model the growth processes of plant development on a computer [160]. It has since been developed to generate fractal designs and model other physical growth processes. As it had been previously proved itself as a compact representation for generating complex designs it was used for evolutionary design.

Hornby and Pollack [85] combined a 3D turtle graphics system with an L-systems to evolve table designs. They compared a generative and a non-generative encoding and found that the re-use and repetition of the generative encoding produced better designs more easily.

As an L-system is based on a rewrite system, it can be represented as a CFG. O'Neill and Brabazon [143] used a CFG to act as an L-System and generate images. The results of the L-system were graphed by generating instructions that could be placed in a postscript file and then viewed. Fitness was provided by the user and the system showed itself capable of generating varied and innovative designs. O'Reilly and Hemberg [153] used a variation of L-systems to generate conceptual designs in their design tool GENR8. Hemberg Extended L-systems (HEMLS) extended Map L-Systems [111] by adding turn angle and branch angle rules. This allowed the designs generated by the grammar to occupy a three dimensional space.

Smith and Rieffel [175] describes a generative system that uses a tetrahedron grammar also derived from a Map L-system. The grammar generated bodies for artificial life robots that are then evaluated using PhysX physics engine. Drawing the analogy between the edges of a graph and the faces of a tetrahedron, they extended the Map L-System to work on three dimensional faces rather than two dimensional edges.

2.6.4 Operators

The representation cannot be considered independently from the operators that are used to navigate it. If representations are to be discussed then the manner in which the operators move around the search space must also be examined. Jones [93] originally highlighted how operators in conjunction with the encoding define the search space. As grammatical evolution is the algorithm used throughout this thesis, the operators used by GE will be examined in detail in chapter 3.

2.7 Discussion

The intention of this thesis is to build on the works described above and apply their approaches to the area of evolutionary design exploration. Although interactive approaches to EC have been proven to allow directed evolution, interactive GE has only had limited application to the field of architectural design exploration. This thesis intends to broaden this research and to investigate and address the following research gaps; Active user intervention through the use of novel operators in GE, the application of structural analysis for aesthetic evolution and the development of user interfaces suitable for IEC.

Specifically, the combination of objective and subjective fitness evaluation for reducing the search space described in Ventrella [192] is implemented in Chapter 4. The analysis performed on mutation in Chapter 5 and Chapter 6 was done with the intention of developing user operators for active user intervention as described in Section 2.5. The idea of interrupt, intervene and resume described in Section 2.3.2 provides the motivation for the interface described in Chapter 7. By using techniques that have already proven successful when applied in different research areas, this thesis intend to broaden the application of EC to the field of architectural design.

2.8 Summary

This chapter presented an overview of literature related to the investigations of this thesis. Different approaches to computer generated architectural design and the application of EAs to design problems was then examined. An explanation of narrow and broad IEC was given and different methods for active user intervention were discussed in detail. Finally, different representations suitable for design generation were discussed. The next chapter presents a detailed description of the algorithm used in the investigations of this thesis, grammatical evolution.

Chapter 3

Grammatical Evolution

Grammatical evolution [145, 48] is a grammar based form of genetic programming [128] that uses evolutionary processes to automatically generate programs. This chapter describes the algorithm that is the basis of the investigations in this thesis. GE differs from traditional GP by replacing the parse-tree based structure of GP with a linear genome [140]. The algorithm is modelled after the process that transcribes a DNA sequence into a protein. This abstraction, called the genotype to phenotype mapping, is the primary difference between GE and tree based GP. The evolutionary operators of tree based GP directly manipulate the structure of a program tree. While the tree itself could be viewed as a genotypic encoding and the output of the program as the phenotype, this mapping is implicit. GE uses an explicit mapping procedure.

GE accomplishes the mapping process by combining a context free grammar (CFG) with a rule selection mechanism based on DNA. The DNA is represented as a binary or integer string and it is used to select rules from the grammar to generate an output. Abstracting the representation upon which evolutionary operators are applied from the final output allows GE to be applied to any problems that can be represented by a grammar. The investigations of this thesis intend to combine CFGs capable of generating three dimensional shapes with the GE algorithm and examine if evolution can be used to direct the design process.

Context free grammars allow complex language constructions to be represented efficiently in a compact format and GE provides a generic methodology for evolving that format. Accordingly GE has been utilised in a variety of applications such as financial modelling [23, 42], computer generated animation [133], computer game AI [29, 113], femtocell configuration [79], 3D image generation [135] and music generation [127]. The results of these applications showed that GE was capable of using evolution to direct search in these problem spaces.

This chapter is organised into the following sections. A definition of context free grammars and the formal notation used to describe CFGs are described in Section 3.2. An explanation of the genotype to phenotype mapping process is described in Section 3.3. Different approaches to initialising the population are addressed in Section 3.4. The operators applied to the genotypic representation are explained in Section 3.5. A summary of the chapter is given in Section 3.6.

3.1 Overview

The intention of this section is to give a brief explanation of the GE process, as shown in Figure 3.1. The first step is to create an initial population. The individuals in this population consist of integer strings. The integer strings, referred to in GE as chromosomes, are used to select rules from a context free grammar. Iterating through the rule productions outputs a string in the language represented by the context free grammar. The string is referred to as the phenotype and represents a program. The program is executed to generate an output. The output is then evaluated using a predefined fitness function.

Once a fitness value has been assigned to every individual in the population, a subset of the population is chosen to generate the next population. The selection operators use the fitness values and a probabilistic mechanism to pick individuals. If the best individual in the population has reached the stopping criteria then the process is terminated, otherwise the selected subset of the population are passed to the variation operators.

GE is similar to traditional genetic algorithms as the evolutionary processes are carried

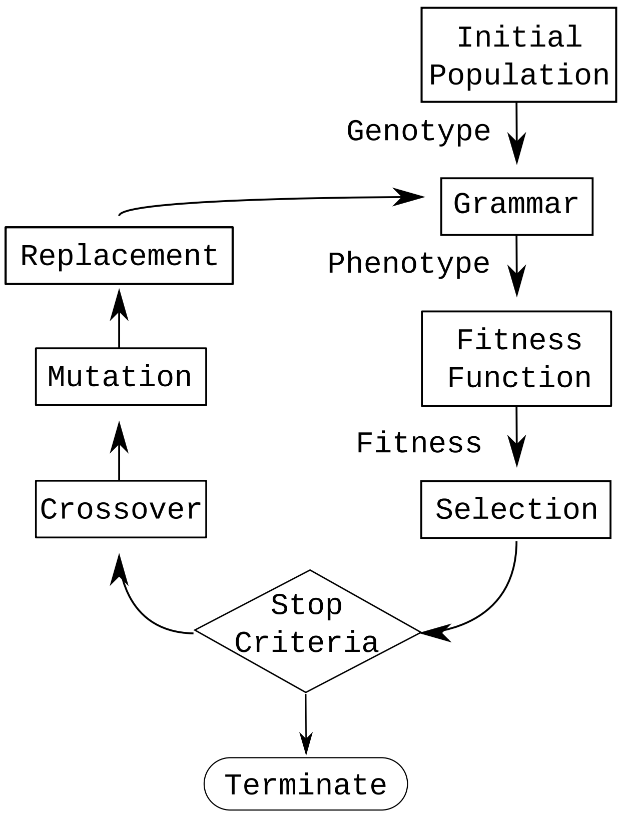


Fig. 3.1: An overview of the GE algorithm.

out on the integer string. The crossover operator exchanges components of individuals with other individuals in the population. An element of mutation and variation is introduced to the population by randomly changing some of the integer values. Once a new population has been created, the replacement operator then adds the individuals to the new population. The process is repeated until the stopping criteria has been reached.

Several alternative grammars have also been used with grammatical evolution such as attribute grammars [37], logic grammars [94], tree-adjoining grammars [132], and probabilistic grammars [81]. As these approaches have not been explored in this thesis they shall not be described in detail. The next section gives an explanation of how context free grammars operate and why they were chosen as the basis for grammatical evolution.

3.2 Context Free Grammars

A grammar is a mechanism for producing sets of strings [75]. A formal grammar consists of a set of structural rules that define the composition of sentences or phrases in that language. The components of a rule are called productions and they consist of terminal and non-terminal symbols. Terminal symbols are literal characters in the language that cannot be broken down into smaller parts by the rules of a grammar. Non-terminal symbols are components of a language that can be replaced using the grammar rules.

Context free grammars are a subset of grammars in the Chomsky hierarchy that form the theoretical basis of most programming languages. A context free grammar is formal grammar where every rule is of the form:

$$A \to \alpha$$

where A is a single non-terminal and α is a string of zero or more terminals and zero or more non-terminals.

In a CFG the production of a non-terminal rule is not dependent on the symbols that surround it. CFGs allow components of a language to be easily grouped into larger modules. The simplicity of CFGs and their ability to be easily and efficiently parsed means that they are used to describe the structure of programming languages. The most common notation used to describe a CFG in computer science is Backus-Naur form (BNF).

Backus-Naur form is a notation developed by John Backus and extended by Peter Naur to represent context free languages [6]. It was originally developed for the Algol 58 programming language. The formal notation uses production rules that consist of terminal and non-terminal symbols to express a context free language as a grammar. Terminal symbols are literals in the language and cannot be changed using the rules of the grammar whereas non-terminals are symbols which can be replaced in the grammar. The non-terminal symbols are distinguished by using angle brackets, e.g., ⟨non-terminal⟩.

The rules in a BNF grammar are structured so that there is a non terminal on the left hand side (LHS) of the rule and a combination of terminals and non-terminals on the right hand side (RHS) of the rule. Terminal symbols never appear on the LHS of a rule. If there are multiple productions possible for a particular rule, these are separated by a vertical bar. An example of such a rule is shown in Figure 3.2.

```
<A> ::= <B> | <C> | d
```

Fig. 3.2: A BNF grammar rule. The LHS is a non terminal $\langle \mathrm{A} \rangle$ and the RHS is composed of two non-terminal productions, $\langle \mathrm{B} \rangle$ and $\langle \mathrm{C} \rangle$, and a terminal production d.

In order to generate a sentence in the context free language represented by a CFG, an initial non-terminal symbol is chosen as the start symbol. The symbol is then expanded using its respective BNF rule. The process then continues iteratively, expanding the left most non-terminal until all non-terminals in the derivation have been replaced by terminals. If there are multiple productions for a rule then a methodology must be used to choose that production. The approach used by GE for choosing rule production is described in the next section.

3.3 Genotype to Phenotype Mapping

GE uses a mapping process to create output called the genotype to phenotype mapping. The genotype to phenotype mapping is a terminology that has been adapted from the field of biology. A genotype is an encoding upon which the evolutionary operators of mutation and recombination act. A biological example of a genotype would be human DNA. Changes to the genotype are translated into changes to the phenotype. A phenotype is an observable characteristic of an individual. An example of this in humans would be eye-color or height. Evaluation of the fitness is performed on the phenotype.

This biological concept was introduced to the field of EC as a method for separating the search and solution space [8] and as a metaphor for describing the representations and mapping processes. An illustration of this process is shown in Figure 3.3. The binary string is analogous to the DNA double helix as both are responsible for defining the characteristics of the phenotype. In the biological case, the information is used to determine the combination of amino-acids to be joined together to create a protein while GE uses the information is used to select productions from a grammar to generate a program.

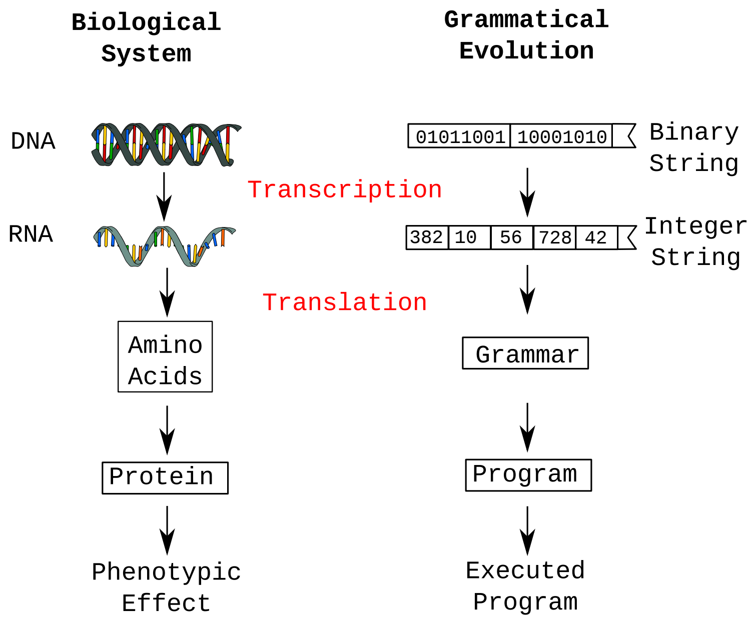


Fig. 3.3: A comparison between the genotype to phenotype mapping in molecular biology and in GE.

GE generates strings in a context free language by using a list of numeric values (called a chromosome) to select the production choices for non-terminals. The numeric values in the genotypic representation can be stored as either binary or integer values. The choice of genotype representation has an effect on the algorithm, as shown by Hugosson et al. [87], but this is not examined as integer representations are used exclusively in the investigations of this thesis. Once all the non-terminals in a derivation have been mapped to terminals an output string in that language has been generated. The output strings are the phenotypic form of the individual. The phenotypes are computer programs that may then be executed and evaluated by the fitness function.

The chromosome is made up of codons. Each codon in the chromosome is used to select a production rule from a BNF grammar. A simple example BNF grammar that could be used for symbolic regression is given in Figure 3.4. `<expr>` is the start symbol from which all programs are generated. The grammar states that `<expr>` can be replaced with either one of `<op> <expr> <expr>` or `<var>`. The recursive nature of this rule allows the grammar to represent an infinite number of expressions. The `<op>` rule has four possible productions and replaces the non-terminal with an addition, subtraction, multiplication or division operator. The `<var>` rule has three possible productions and replaces the non-terminal with an x, y, or z terminal symbol that represents a variable.

```
<expr> ::= <op><expr><expr>    (0)
         | <var>               (1)

<op>   ::= +                   (0)
         | -                   (1)
         | *                   (2)
         | /                   (3)

<var>  ::= x                   (0)
         | y                   (1)
         | z                   (2)
```

Fig. 3.4: A simple BNF grammar.

The integer codons decide which production is chosen by computing the total number of rule productions and using that number to calculate the modulus of the codon value. This can be represented with the following formula:

$$\textit{Rule (idx)} = \textit{Codon Value \% Num. Productions} \tag{3.1}$$

The BNF rules are applied to the left most non-terminal with the respective codon choosing the production, this process is continued until all non-terminals have been expanded and a derivation tree is built. This process is shown in Figure 3.5. The terminals at the leaf nodes of the derivation tree create a string from the grammar which, in turn, represents the program.

In GE there are several stages in the mapping process, each with its own observable characteristics. As each stage also contains a version of the final instantiation, it can be classified as a phenotype. The four main phenotypic stages of the mapping process are shown in Figure 3.6. The integer list is translated into a derivation tree using the BNF

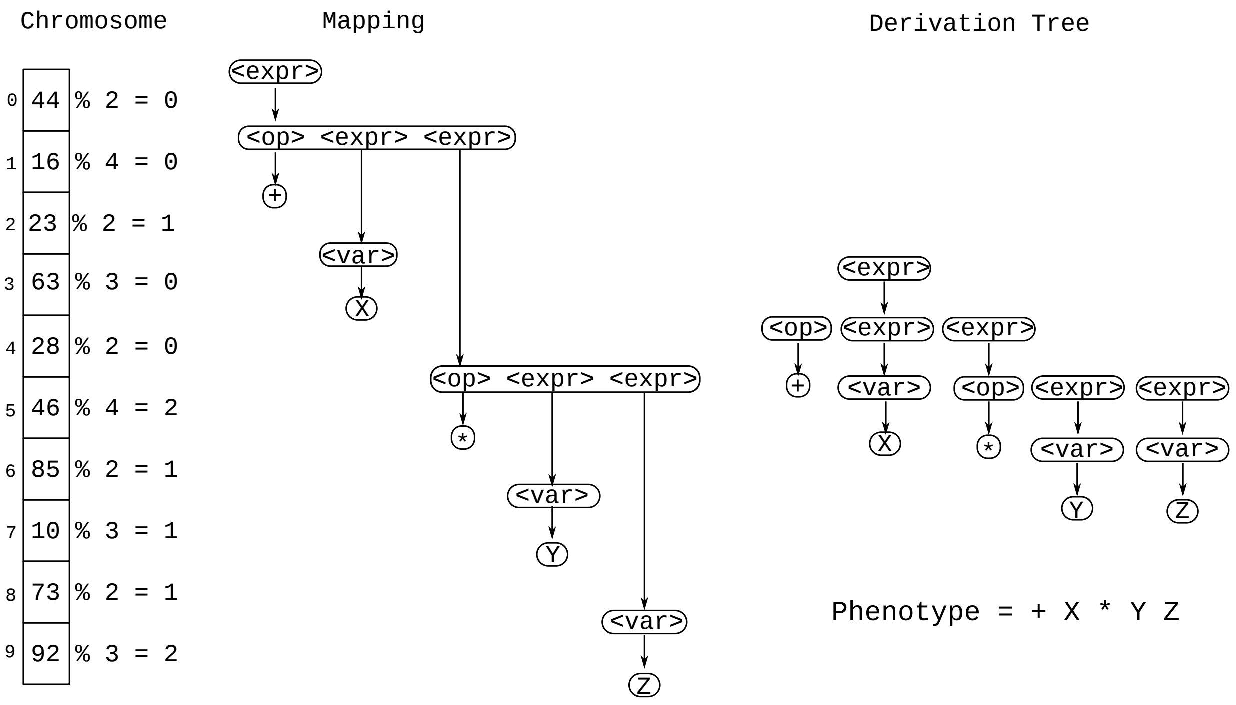


Fig. 3.5: A mapped example using the sample grammar and the resulting derivation tree.

grammar defined in Figure 3.4 and the mod rule. The terminal production choices at the leaves of the derivation tree are shaded green. The leaves form a string that represents the program, in this example the program is an equation. The string is then evaluated to produce the final output phenotype, in this case the equation forms a three dimensional plot of a plane. The final stage of the process is the evaluation. The output is analysed by a fitness function which returns a scalar value representing the quality of an individual based on the criteria of the problem.

The mapping process described above is the canonical depth-first mapping approach. Several alternative mapping methods have been explored such as breadth-first mapping [57], bucket mapping [96] and a position independent mapping called π-GE [155]. As the canonical mapping is used in the experiments in this thesis, alternative approaches are not described in detail. The next section discusses approaches for initialising the population.

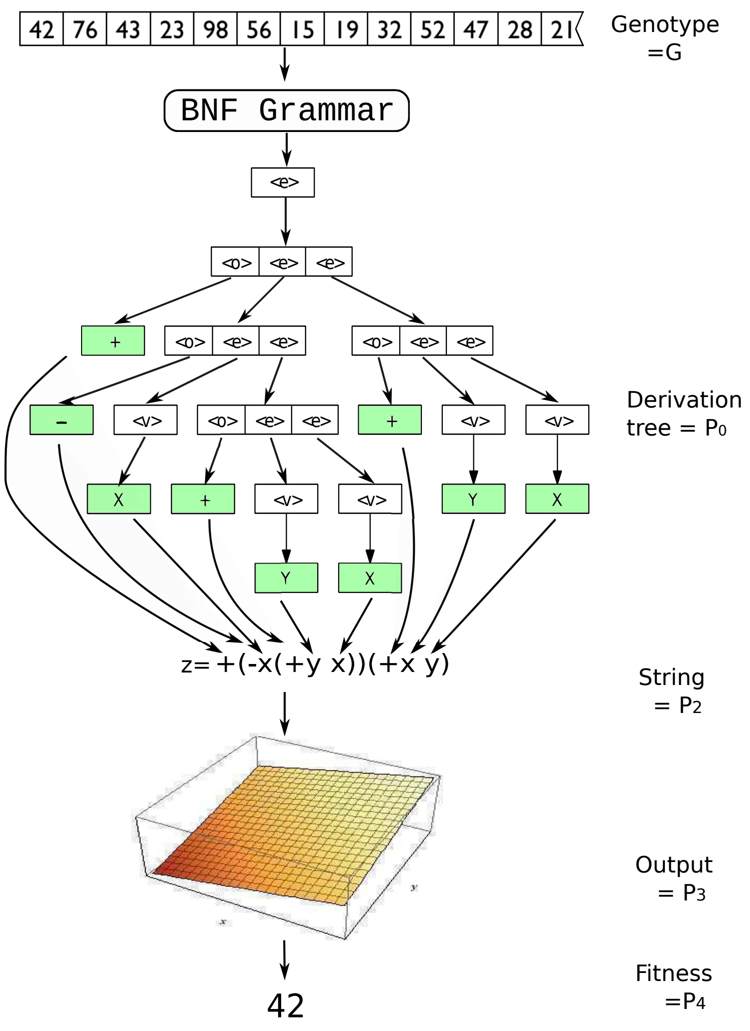


Fig. 3.6: The different stages of derivation in GE.

3.4 Initialisation

As the initial population form the basis of the remainder of the search execution, it is important that they are widely distributed throughout the problem space. There are several approaches for generating the initial population, this section describes the two approaches used in this work.

The most direct approach to initialisation is to randomly generate the integers for the chromosome. While this method is the simplest means of initialising a population it can result in a skewed distribution of small trees [73, 115]. A grammar that requires a recursive rule to increase tree depth, such as the grammar shown in Figure 3.4, has a 50 % probability of being expanded once, as there are only two productions. To get individuals of greater tree depths requires the recursive rule to be chosen repeatedly, as such the probability of greater tree depths decreases. Accordingly random initialisation should only be used for grammars that have a maximum bound on the problem space and do not require recursive rules to increase the depth of the program tree.

Ramped half and half (RHH) is the initialisation method used in traditional GP [103]. Initialising individuals using ramped half and half requires a grow and a full method. Both methods create individuals that are not allowed exceed the maximum allowable depth.

The full method grows individuals until all the leaves are uniformly at the greatest depth [159]. It does this by choosing non-terminal expansions until the maximum depth is reached and then only chooses terminal rules. The grow method generates trees where the maximum tree depth reaches the depth limit, i.e., the deepest leaf of the tree structure reaches the depth limit [159]. The other branches on the tree are allowed vary their depths. When the methods are applied separately they tend to generate a limited variety of individuals so RHH uses both methods to initialise the population.

Individuals in GE can be represented using a derivation tree structure and adapted grow and full methods can be used to initialise a GE population. While RHH can skew the distribution of tree structures [73], it is still a suitable method for initialising individuals where the tree depth is dependent on recursive rules for expansion. There are different approaches to initialisation such as sensible initialisation [166] but these alternatives are not explored in this thesis.

3.5 Operators

This section presents the different operations that are applied to the solutions by the evolutionary algorithm. Operators in GE are normally applied to the genotype representation before the mapping process occurs, except in cases where the operator is explicitly applied to a phenotype. This approach differs from standard GP operators that are applied to the phenotype, i.e., the program tree. A description of the variation, selection and replacement operators is now given.

3.5.1 Variation Operators

The variation operators are used to create new individuals out of an existing population. They explore the search space by altering existing individuals and exploit components by exchanging them amongst the population. The two variation operators used by GE are described below.

Mutation

Integer mutation is considered to be an explorative operator that maintains genetic diversity by altering the values of the codons. Mutation in canonical GE is applied on a per codon basis as opposed to a per chromosome basis so the mutation rate states the probability for each codon changing value. As such it is traditionally set quite low. While a mutation event in GP only effects a single node in the program tree, mutation events in GE can alter the meaning of other codons in the chromosome [95, 32].

The mapping process in GE means that every codon is dependent on the values of the codons that preceded it. If a codon is changed to a production of a different arity or if a non terminal that follows a different derivation path is selected, the following codons can change their meaning. The result of such a mutation is called the ripple effect.

An example of a ripple mutation event is shown in Figure 3.7. The codon at index four is mutated. The production chosen by the codon is changed from `<op> <expr> <expr>` to `<var>`. As the production is changed to a different non-terminal the expression of the codon that immediately follows it now encodes for a `<var>` non-terminal instead of an `<op>` non-terminal. The codons from index six onwards are no longer required as the arity of the production changed from three to one.

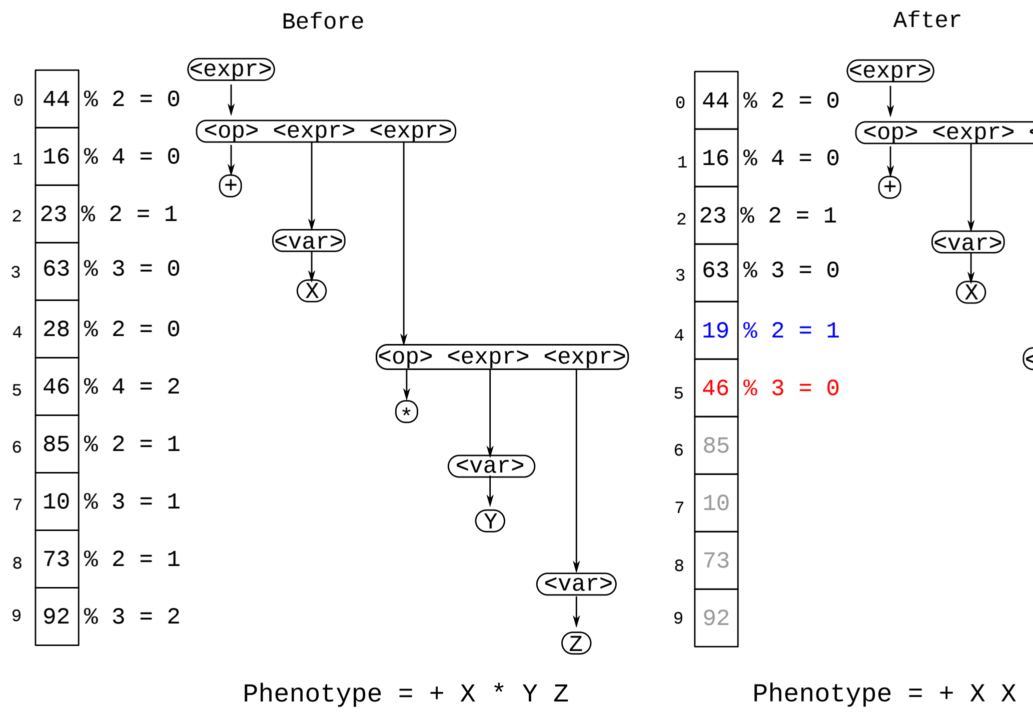


Fig. 3.7: An example of a ripple mutation event in GE. The mutated codon is shown in blue and the effected codons are highlighted in red and grey.

The implication of ripple mutation means that small effects at the genotypic level can have a large impact at the phenotypic level. The impact of this event is dependent on the placement of the codon in the chromosome. In the most extreme cases a ripple mutation at the start of the chromosome could generate an entirely different tree. One of the primary investigations of this thesis is an examination of the effects of ripple mutation and it is explored in greater detail in Chapter 5.

Crossover

The intention of crossover is to act as an exploitative operator that exchanges information between individuals in the population. Crossover in traditional GP involves transplanting subtrees of the program tree from one individual to another. GE performs a similar action by exchanging sections of the chromosome but as crossover is applied to the genotype, different mappings can occur. Crossover in GE is more akin to GA crossover as it operates on integer strings. Once two parents have been chosen for crossover, a point is selected on each of the parent chromosomes. The part on right hand side of the point is then swapped with the respective part in the other parent chromosome. An example of this process is shown in Figure 3.8.

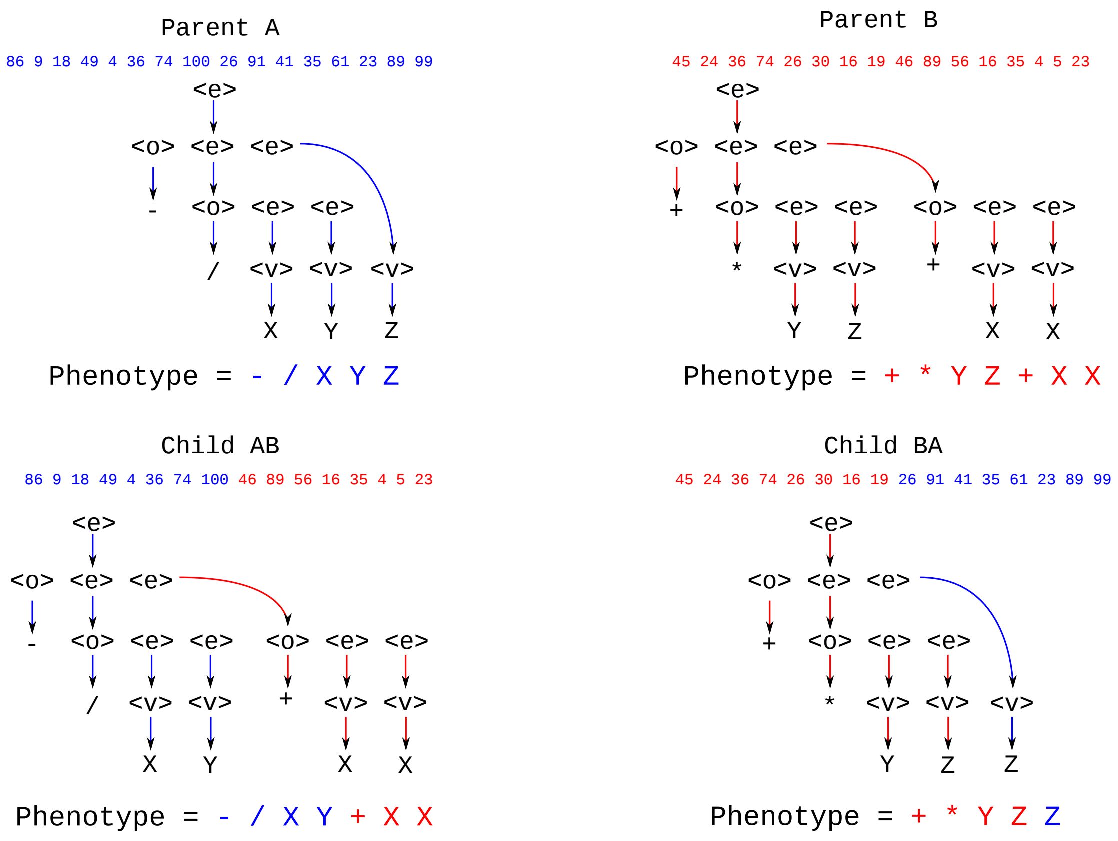


Fig. 3.8: An example of crossover in GE. Half of the blue chromosome of parent A is exchanged with half of the red chromosome of parent B.

As each codon depends on the values that preceded it during the mapping process, crossover can result in a different expression of the codon values in a way similar to ripple mutation. It has been shown that even though the expression of the derivation tree can change, that it performs in a statistically identical way to a homologous crossover operator and showed that it was not performing as a macro-mutation operator [145]. Harper and Blair [74] further addressed this by implementing a crossover operator that only choose points that preserve the derivation tree order. As single point crossover is used exclusively for the experiments in this thesis, alternative crossover methods are not examined.

3.5.2 Selection Operators

Selecting only the best individuals for variation is a greedy approach that results in convergence on local optima. It is better to have a probabilistic approach that allows inferior solutions to be occasionally selected as this broadens the variety in the population of solutions. Two approaches to selection are now discussed in this section.

Roulette wheel selection compares the whole population simultaneously for each selection. Each individual is given a probability of selection proportionate to their fitness where the sum of all the component probabilities is equal to one. This is analogous to the pockets on a roulette wheel where the size of the pocket is dependent on the solution's fitness.

Tournament selection only compares a subset of the population for each selection. The tournament size is selected in advance by the experimenter. A subset of individuals equal to the tournament size is selected at random from the population. The winner of the tournament is the individual with the highest fitness and is selected for the new population. The advantage of tournament selection is that it allows for the selection pressure to be altered. A tournament size of one is equivalent to random selection while a tournament size equal to the population size is equivalent to always selecting the best individual.

3.5.3 Replacement Operators

The iterative approach of evolutionary algorithms requires that individuals in the old population are replaced with new individuals. Two common approaches to replacement are steady state and generational. Steady state only replaces individuals in the old population with individuals from the new population if their fitness is better than that of an individual in the old population. Steady state maintains a population of good solutions to be selected from and so has a higher convergence rate than generational replacement.

Generational replacement discards the old population entirely and replaces it with the new population. The result is increased variance in the average fitness of the population, as there is no assurance that the new individuals will be as fit as the old individuals they replace. Generational replacement converges at a slower rate than steady state replacement and also requires some form of elitism, i.e., the best individual is added to the new population without any variation. Elitism prevents the EA from “losing” the best solution it has found during the run.

3.5.4 Alternative Search Engines

An evolutionary algorithm is one possible approach to searching problem spaces. The modular nature of GE allows for the evolutionary algorithm to be substituted with an alternative algorithm without having to redefine the grammar or change the experimental setup. Several alternatives search engines have been used such as the particle swarm algorithm, called grammatical swarm [142] and grammatical differential evolution [139] based on the differential evolution algorithm [182].

3.6 Summary

This chapter described the canonical form of grammatical evolution. A brief overview of the algorithm was given which was followed by a detailed description of context free grammars and the structure of BNF notation used by GE. The mapping process was explained and an example mapping was carried out. Initialisation of the algorithm and the operations carried out by the algorithm were then described in detail. The remainder of this thesis now examines the application of GE to architectural design exploration.

Chapter 4

An Automatic Fitness Function for Design Evaluation

4.1 Introduction

This study evolves and categorises a population of architectural designs by their ability to handle physical constraints. The design of a structure involves a trade-off between form and function. The aesthetic considerations of the designer are constrained by physical considerations and material cost. The objective nature of engineering constraints lend themselves to implementation as an automated fitness function and allow a design to be optimised accordingly [99]. This work implements a fitness function that applies engineering objectives to automatically evaluate designs, as shown in Figure 4.1. Automating the evaluation process reduces the search space as only feasible designs are presented to the user.

Architectural design is more restricted than other forms of artistic design. For example, an architectural design must be functional as well as structurally sound. The constraints placed on architectural design provide an opportunity for the evolutionary algorithm as they can be represented as a fitness function and optimised accordingly. The approach is similar to the work described in Ventrella [192] as only “plausible” designs that meet the constraints are presented to the user.

Combining the conceptual approach of computer generated architectural design with an automated fitness function has two main advantages. First, designs can be optimised to increase their functionality and strength while reducing the amount of material used. This will, in turn, make the designs more credible and realisable. Second, assigning an objective fitness rank to a design also provides a mechanism for clustering designs based on how well they fulfill the objectives [121]. The different clusters could further reduce the search space: if a user has a distinct aesthetic preference for a particular cluster then the algorithm can accelerate convergence on these aesthetically pleasing designs.

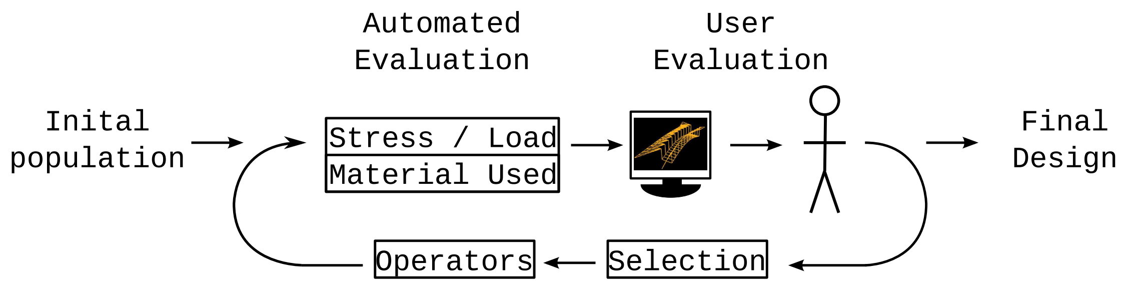


Fig. 4.1: An automated fitness function reduces the search space presented to the user.

This chapter is organised as follows. Section 4.2 gives a description of the approach taken for design generation and analysis. Section 4.3 investigates if designs can be optimised by a fitness function based on conflicting engineering constraints. Section 4.4 examines if the fitness values can be used to group the resulting designs. The conclusions are discussed in Section 4.5.

4.2 Overview of System and Grammar

This section describes the software used and the approach taken to generate and structurally evaluate designs. The experiment was run using Architype, an interactive design generation tool based on GE. Architype is based on PonyGE version 1.3, a Python implementation of GE that can be downloaded from the Google code website [80]. The system used in this experiment is comprised of four parts: an evolutionary algorithm, a design grammar, structural analysis software and a multi-objective fitness function. A diagram of the system is shown in Figure 4.2.

The system was designed to be applied to practical problems. Accordingly, a structural engineering class project was chosen as the problem. The design brief for the project can be found in Appendix B. A brief explanation of how the system was implemented is now given.

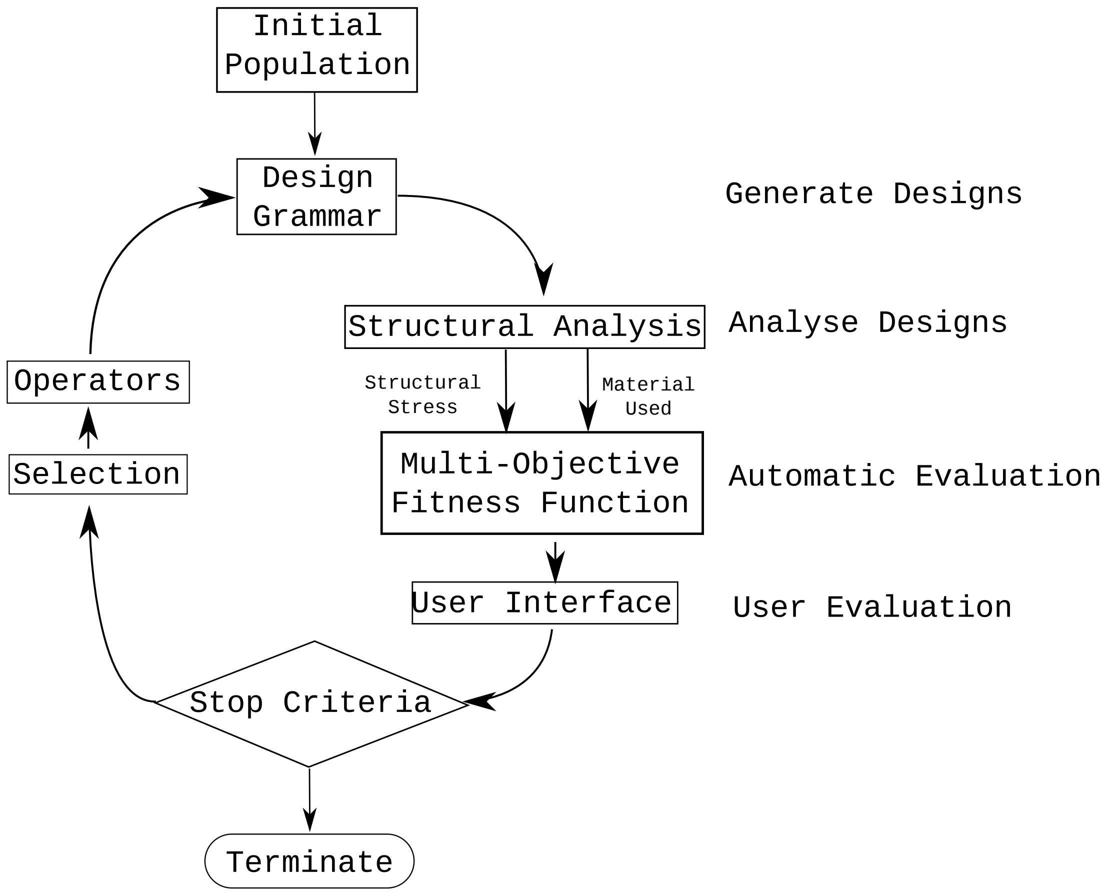


Fig. 4.2: The different stages of design generation.

4.2.1 Design Grammar

The grammar used in this experiment describes a rule system for generating bridges. It was originally conceived based on a brief provided to the third year architecture and structural engineering course of 2010. The students were tasked with designing a footbridge for the

St. Mullins archaeological site in county Carlow. The brief specified that the bridge was to be composed of timber, had an optional arch (rise height 0 to 2 metres), a width of 2 metres and bridge a span of 10 metres. Even though the grammar was created with no consideration for the structural soundness of the design phenotypes, applying a load allows us to judge the performance of the bridge relative to other bridges generated by the same grammar.

The grammar used to generate the bridges is shown in appendix A. The grammar creates graphs using networkx [72], a Python class for studying complex graphs and networks. The bridge designs are stored as undirected graph objects from the networkx class. Each graph consists of nodes and edges. Every node has a Cartesian coordinate attribute and a placement attribute describing what bridge component the node forms a part of, e.g., walkway, handrail, etc. The placement attribute was used to apply loads to the structure as is explained in more detail below.

Three desirable characteristics for a design generator are modularity, regularity and hierarchy [86]. These characteristics were implemented using the novel method of higher order functions. The work in this area is discussed in greater detail in [126] and is described in chapter 2. For structural analysis to be performed on the bridges, a mechanism was required that allowed for loads to be applied to the structure.

The approach used in this experiment was to add attributes to the existing grammar. This allowed us to label components depending on the function that created them. Labelling meant that forces representing the loads could be assigned to the structures automatically and accordingly, that different forces could be applied to different parts of the bridge. An example of a loaded bridge can be seen in Figure 4.3. Now that the grammar for generating bridges has been introduced an explanation of how structural analysis is performed on the bridge phenotype is given.

4.2.2 Structural Analysis

The analysis of structures as computable models is accomplished by using Finite Element Methods that were described in detail in Section 2.3.1 on page 22. Instead of calculating the partial differential equation for a whole design, a continuous structure is discretised into an approximating system of ordinary differential equations. The approximation can then be solved using numerical approximation methods for differentiation such as Euler's method or the Rung-Kutta method. The designs generated by the algorithm are particularly suited to Finite Element Analysis (FEA) as the structures are already discretised into a series of interconnected beams. To analyse the generated designs, San Le's Free Finite Element Analysis (SLFFEA) [107] was used. This software is freely available for download and has been used by engineering companies in industry. It is built on the Unix philosophy of minimalism, modularity and efficiency and, as such, it allows us to analyse the structures with great speed.

Structural optimisation is classified into three categories: topology, shape and sizing. Our work focuses on the topological and shape optimisation components of structural optimisation, as described in Section 2.3.1 on page 22. This is possible as GP can evolve structure and content simultaneously. This work does not examine sizing optimisation as the task was constrained by the design brief. The same fixed sizing was used for the wooden beams and the same material that the students used to build their entries. The modular nature of implementation for this experiment and the use of a design grammar does not rule out the possibility of evolving all three simultaneously, although it would greatly increase the combinatorial possibilities for a design.

The networkx graph objects were parsed into the SLFFEA data structure and output to a file. The data structure specifies the type of material used, the placement of the nodes, the edges and their respective loads. As the graph objects were being parsed, the total Euclidean length of the edges was calculated. The total edge length combined with the beam dimensions allows us to compute the amount of material used, thus providing a second fitness value for the bridge.

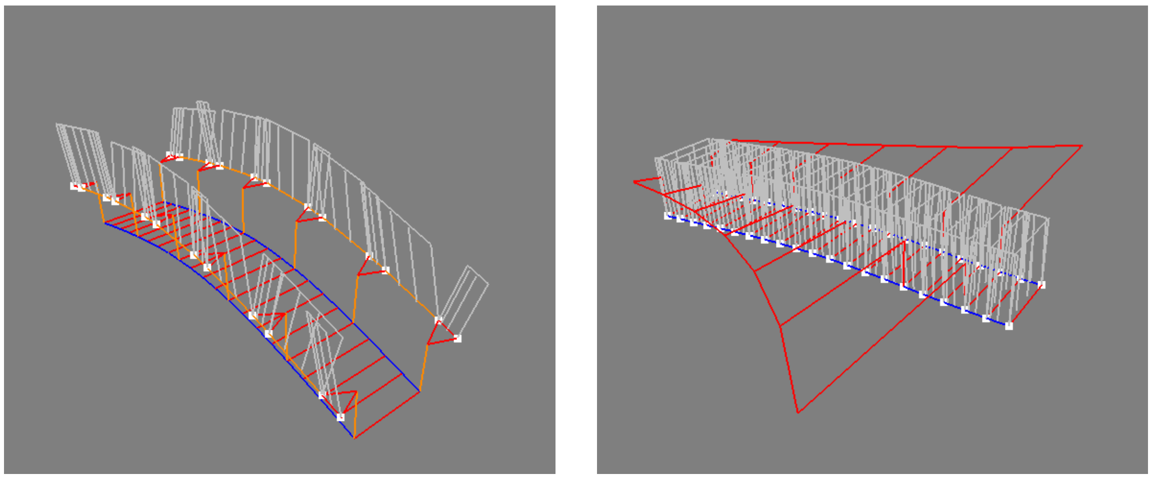


Fig. 4.3: Different magnitudes of stresses being applied to the handrail and walkway.

SLFFEA calculates the displacement for each element and returns the compression and tension forces of the x, y, and z planes for each element. To generate a fitness value from the analysis, we compute the average compression and tension on the structure. The average stress, $\overline{c}$, is calculated in Equation 4.1 and the material used, m, in Equation 4.2, where $B = \{Beams\}$, $|b| =$ is the Length of beam and κ is the area of the cross-section.

$$\overline{c} = \frac{\sum_{b \in B} b_x + b_y + b_z}{\sum_{b \in B} |b|} \tag{4.1}$$

$$m = \sum_{b \in B} |b| \cdot \kappa \tag{4.2}$$

One disadvantage of optimising a structure to reduce stress is that it can be easily reduced by increasing the amount of material used. To prevent individuals from adding redundant beams the overall length can be used as a second fitness values.

Two conflicting constraints can generate a fitness value if they are weighted. The limitation of weighting is that the evolutionary algorithm will quickly converge on individuals optimised to a specific weighting. It also requires that correct weights are known apriori. A better approach is find individuals that are good compromises. To do this a multi-objective fitness function was implemented. How it was implemented is discussed in the next section.

4.2.3 Multi-Objective Fitness Function

Design involves satisfying several objectives that may conflict with each other [173]. Multi-objective evolutionary algorithms (MOEAs) have been shown to be a useful approach for finding the best compromise when tackling a multi-objective problem [201]. Instead of weighting the objectives and allowing an evolutionary algorithm to converge on a single global optimum, the algorithm builds a pareto-front of the individuals that maximise the given objectives. A pareto front shows the individuals that best compromise the conflicting objectives, if the objectives were to be weighted equally. Using fronts can aid the design process by presenting the user with several pareto-equivalent designs and letting them select the design that most closely matches their requirements.

The two structural objectives that were chosen to minimise were stress under loading and the amount of material used. These objectives conflict as the overall stress on a structure can be reduced by adding redundant material. It is possible for the MOEA to generate a pareto front based on aesthetic constraints such as smoothness, curviness, etc, but this possibility is not explored in this thesis.

A GE implementation of the non sorting genetic algorithm II (NSGA2) [46] was used as the selection and replacement mechanism. For each generated structure two fitness values are assigned, one based on the amount of material used and one based on how well the structure endures a given load. A more detailed description of the objectives is given in Section 4.3.

Multi-objective search algorithms do not assume there is a globally optimal solution but instead there are a set of non-dominated solutions. The non-dominated solutions are solutions that are better than the rest of the population for at least a single constraint and at least equivalent for all other constraints. This can be stated mathematically as:

$\mathbf{f}$ is the set of fitness functions: $\mathbf{f} = [f_o, \ldots, f_n]$ such that $\forall f \in \mathbf{f}$ where $f^{non-dom} \leq f^{dom}$ and $\exists f \in \mathbf{f}$ where$f^{non-dom} < f^{dom}$

The NSGA2 algorithm uses a selection operator called the **fast non-dominated sort**. First, the non-dominated individuals in the population are found. The set of non-dominated solutions form what is called the pareto front. Substituting one solution for another on the pareto front will always sacrifice quality for at least one constraint, while improving at least one other. Once the initial grouping of globally non-dominated individuals is found, it is used to calculate the next least-dominated set of individuals. The process is continued until every individual in the population is in a front. The individuals in each front are dominated by the preceding front.

Normally multi-objective applications are only concerned with the individuals on the pareto front. The intention of Section 4.4 is to investigate whether the grouping property of the fronts in the NSGA2 algorithm could also be of benefit for guiding the search process. This section explained how the bridge designs are generated and evaluated, it is now examined in the next section if the designs can be optimised by using an evolutionary algorithm.

4.3 Optimising Designs using Structural Analysis

The first experiment was designed to test whether an evolutionary search is capable of generating designs that minimise the stress in a structure and reduce the amount of material used. As each objective is weighted equally there is no "best" individual in the population. Instead we evaluate the average fitness of the population and see if there is an observable improvement for each of the objectives. Fitness minimisation is fitness improvement in this experiment and throughout this thesis. A control population was generated to examine if the evolutionary process increased the rate of convergence.

The control population is different from a purely random search. The NSGA2 algorithm has a built in selection pressure that compares parent and child populations. This selection pressure alone could account for an improvement over time of a randomly generated population. To examine if beneficial information was being transferred between generations, the same selection scheme was used for both the control population and the evolved population. The difference with the control population is that the individuals in the child population were randomly initialised rather than being created from the parent population. Random initialisation meant that no genetic information was transferred from generation to generation.

4.3.1 Hypothesis

The experiment compares the results for two objectives μ and ν, where μ represents the average stress and ν represents the amount of material used. Given the average fitness of an evolved population for each objective, μ_1 and ν_1, and a randomly generated population, μ_2 and ν_2, after 50 generations, the following hypothesis is stated:

H_0 The average fitness of the population does not differ significantly with respect to the either the material constraint or the load reduction objective, $\mu_1 = \mu_2$ and $\nu_1 = \nu_2$

H_1 There is a statistically significant difference with respect to one of the fitness values $\mu_1 < \mu_2$ or $\nu_1 < \nu_2$

H_2 There is a statistically significant difference with respect to both fitness values $\mu_1 < \mu_2$ and $\nu_1 < \nu_2$

α The significance (α) level of the Wilcoxon rank-sum is 0.05.

4.3.2 Setup

The experiment was carried out using the implementation described in Section 4.2 and the bridge grammar described in Section 4.2.1. The experimental settings are shown in Table 4.1. It should be noted that mutation in GE is applied on a per codon basis.

The material from which the bridge was constructed was small scale air dried oak sections with a moisture content of 20% or more. The structural qualities of this wood were taken from the British Standards BS-EN-338-2003 as a grade D30 class of timber [26]. The material qualities were then assigned to the bridge beams for SLFFEA analysis. For

Tab. 4.1: Experimental settings.

Parameter	Value
Population Size	500
Generations	50
No. of Runs	30
Mutation Rate	1.5%
Mutation Operator	Integer
Crossover Rate	70%
Crossover Operator	Single Point
Selection Scheme	NSGA2
Replacement Scheme	NSGA2
Initialiser	Random
Wrapping	off
Random Number Generator	Mersenne Twister

stresses on a structure to be calculated, fixed points and loaded beams must be assigned. Normally this is done manually by the user. The implementation for this experiment automated the task by using attributes in the grammar, as described in Section 4.2.1.

While the experiment tried to replicate a load that a bridge might be subjected to during actual usage, the main purpose was to compare how well the bridges performed relative to other bridges also generated by the design grammar. The bridges were subjected to a uniformly distributed load (UDL) of 5kN/m upon the walkway itself and a separate 1kN/M load was applied to the handrails. The loads for the bridge were taken from [58].

There were two constraints placed on the designs, one of which was stress based and one that was based on material usage. The stress objective is based on the normalised compression and tension values generated by SLFFEA and given by Equation 4.1. If a beam failed then the bridge was assigned a default fitness of 100,000, the worst possible fitness. This meant that high stress designs were removed from the population and the fitness pressure aimed to reduce stresses over the structure as a whole.

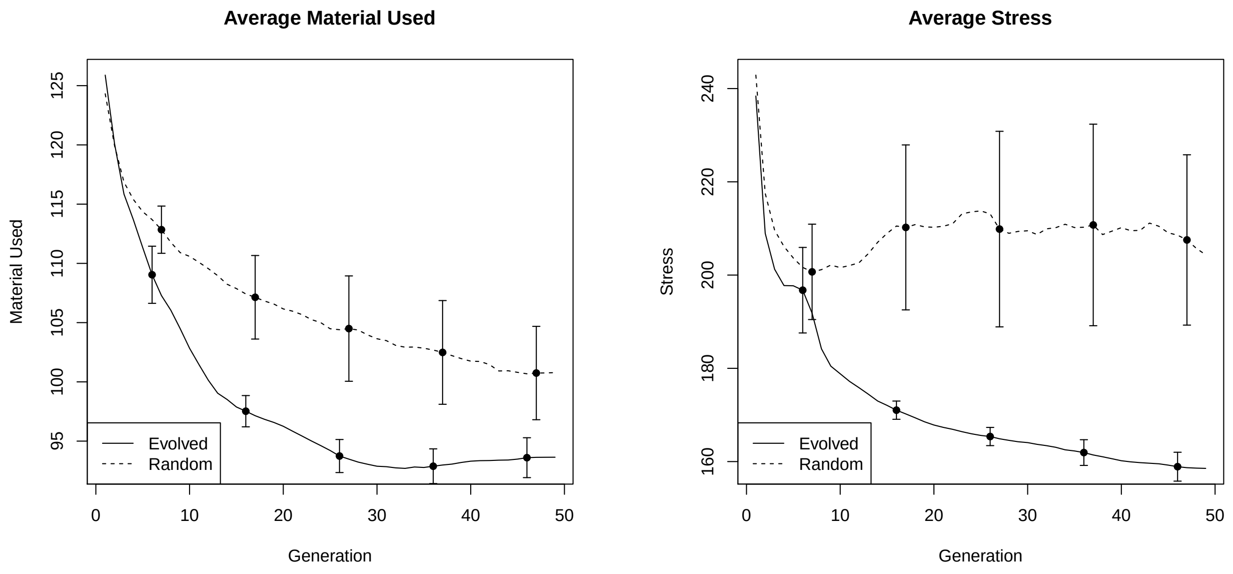
The material constraint aimed to reduce the number of beams used in a structure. This fitness metric is opposed to the stress constraint as one method for reducing the average stress on the beams is by adding more unnecessary beams. By adding a penalty for the total weight of material used, it can force the algorithm to simplify the design. Reducing the total weight of material used also translates into direct savings when manufacturing an instance of the design.

4.3.3 Optimisation Results

The results for the experiment are shown in Figure 4.4. In these graphs, minimisation is fitness improvement. As there is no global best individual for a multi-objective problem, the analysis instead examines the mean fitness of the population. Thirty independent runs were carried out for both the control and evolved populations. The results obtained showed that there is a reduction in the average amount of materials of 24% (Figure 4.4(a)) and a reduction of the average load placed on the structure of 44% (Figure 4.4(b)) after 50 generations.

To investigate if this improvement was due to the evolutionary algorithm or whether it was a result of the selection pressure from the NSGA2 algorithm the analysis compared the evolved results with the control results. A Wilcoxon rank-sum was performed between the random and evolved results for both length and stress and both were shown to have a p-value of less than 2.2×10^{-16}. The result shows there is a statistically significant difference between the two samples and so the null hypothesis can be rejected. Although the evolutionary algorithm outperformed the control population, it is interesting that selection pressure in the NSGA2 algorithm alone managed to reduce the amount of material used by 16% and the average stress by 14.5% respectively.

The results show that using structural analysis and an MOEA in combination with an evolutionary algorithm can significantly reduce the stresses and self weight of a design. The findings support the second hypothesis that multiple objectives can be optimised simultaneously. Figures 4.5(a) and 4.5(b) show a density plot of the individuals of a population at generation 0 and generation 50 respectively. The x axis represents the stress fitness and the y axis represents the amount of material used. The darkness of the color represents the density of the population at that point. While the individuals are still



(a) Average material used.

(b) Average amount of stress.

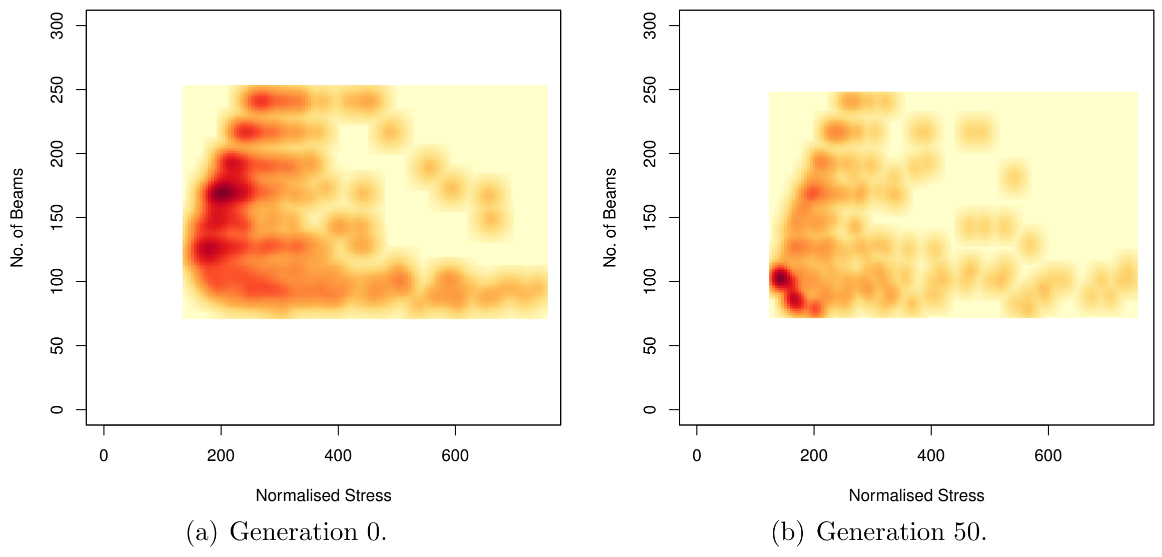
Fig. 4.4: The fitness minimisation average of the population over 50 generations.

distributed over a wide are at both generation 0 and generation 50, it is clear that main population concentration and the pareto-front are moving toward a pareto-optimality over the course of 50 generations.

4.3.4 Discussion

This is the first implementation of GE that uses a multi-objective fitness function based on pareto dominant selection. The experimental results demonstrate that it can function as a selection scheme for GE. This is a promising advance as the NSGA2 algorithm has proved itself capable of being applied to many different types of multi-objective problems. Combining NSGA2 with also allows multi-objective selection and replacement to be applied to complex problems that are represented by a mapping.

The results show that the system is capable of evolving structures that increasingly satisfy the constraints specified in the multi-objective fitness function. This is very important when progressing a design from a conceptual stage to something that is instantiable. This is a process that engineers and architects face on a daily basis. After using the NSGA2



(a) Generation 0.

(b) Generation 50.

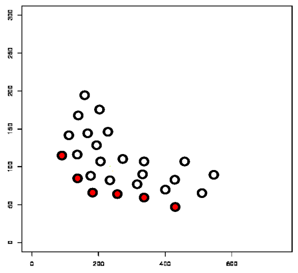
Fig. 4.5: Scatter plot with a density estimation function that shows the progression of front over 50 generations.

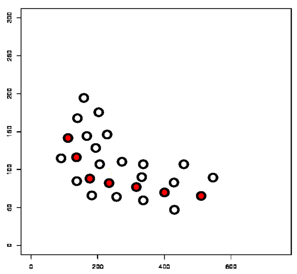
for optimisation, the next section explores if the NSGA2 algorithm could also be used for categorising the designs based on how well they meet the objectives.

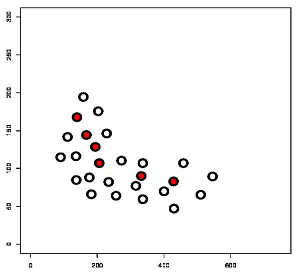
4.4 Categorising Designs using Structural Analysis

While optimisation plays a significant part in the design process, design is not purely about optimisation. The intention in creating an evolutionary design tool is not to exclusively optimise designs but to allow the architect to explore the design search space. To this end, it is imperative to bias the algorithm towards designs that the user finds interesting and appealing. A feature of the NSGA2 algorithm is that it categorises the individuals based on how well they conform to the objectives while building a pareto front of the population. These categories could provide a means of reducing the search space presented to the user.

The experiment carried out in this section uses the NSGA2 algorithm to group the bridge designs by instantiability, i.e., how easily they can be created. The NSGA algorithm calculates the globally non-dominated individuals to show the first front and then uses these individuals to define each successive front. This process is shown in Figure 4.6.


(a) Front 1.


(b) Front 2.


(c) Front 3.

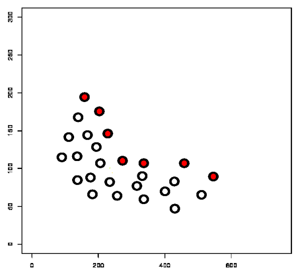

(d) Front 4.

Fig. 4.6: Each front is used to define the next one.

The groupings created by the fast non-dominated sort are shown in Figure 4.7. The population is broken into sub-populations based on their relative fitness. If the user finds designs that optimise the constraints to be more aesthetically appealing, then selection pressure could be applied to drive the pareto-front in this direction. Categorising designs by their place on the fitness landscape allows us to accelerate convergence onto more appealing areas of the search space. The experiments in this section seek to examine if the user finds objectively fit designs aesthetically pleasing.

4.4.1 Hypothesis

Given a set of user preferences for one of two design groupings, μ_0 and μ_1, that were categorised based on how well the fulfilled the physical design constraint, the following hypothesis is stated:

H_0 The users will have no significant preference for either group, $\mu_0 = \mu_1$

H_1 There is a statistically significant user preference for a particular grouping, $\mu_0 \neq \mu_1$,

α The significance (α) level of the binomial test is 0.05.

4.4.2 Setup

The settings for this experiment are the same as described previously except that the experiment shows the user the results of applying the non-dominated sort to the first

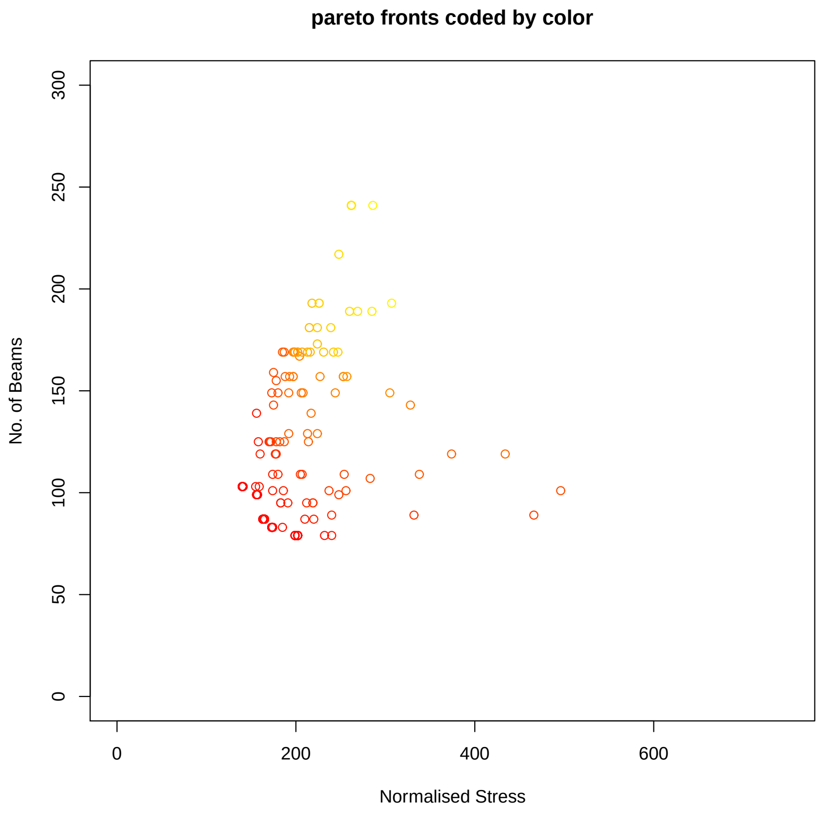


Fig. 4.7: An instance of the fronts produced by the NSGA2 fast non-dominated sorting algorithm.

generation of individuals. The reason for this is that the selection pressure of the evolutionary algorithm focuses the population towards the pareto front, as highlighted in the difference between Figures 4.5(a) and 4.5(b), and converges on a small subset of globally non-dominated individuals. Using the first generation allows for greater variation in the population and an increased likelihood of a user perceivable difference between the different fronts. The experiment randomly selected designs from the first two fronts and the last two non-empty fronts. To generate images of the designs, an open source mesh viewer called medit [61] was used. Medit was developed by INRIA as part of the finite element analysis program, freeFEM[158]. An example of the output can been seen in Figure 4.8.

An online survey was then conducted on the designs. The survey consisted of presenting two designs, side by side, as shown in Figure 4.9. The question presented to the user was "click on the design you find most aesthetically pleasing or on *no preference* if you have no preference". The wording of this question is important could have a significant effect on the result. For instance, if the user was asked which bridge they preferred, their personal preference could have been for a simpler yet less aesthetically impressive design. As the intention of the experiment was to segment the designs with the greatest visual impact, the question as stated above matched that criteria.

The user had to evaluate 100 comparisons. If the user had no preference for a particular design they can indicate this with the no preference button. The presentation of the images were randomised so that there was no bias for which side the images appear on. There was no time limit set for how long the users had to complete the survey. The survey was carried out by post-graduate computer science students and undergraduate volunteers from the school of architecture. The latter group was chosen because of their expertise in architectural design. As undergraduate students are considered a vulnerable group, this survey was authorised by the ethics committee and the head of the school of Architecture (reference number: LS-E-10-159-Byrne-ONeill).

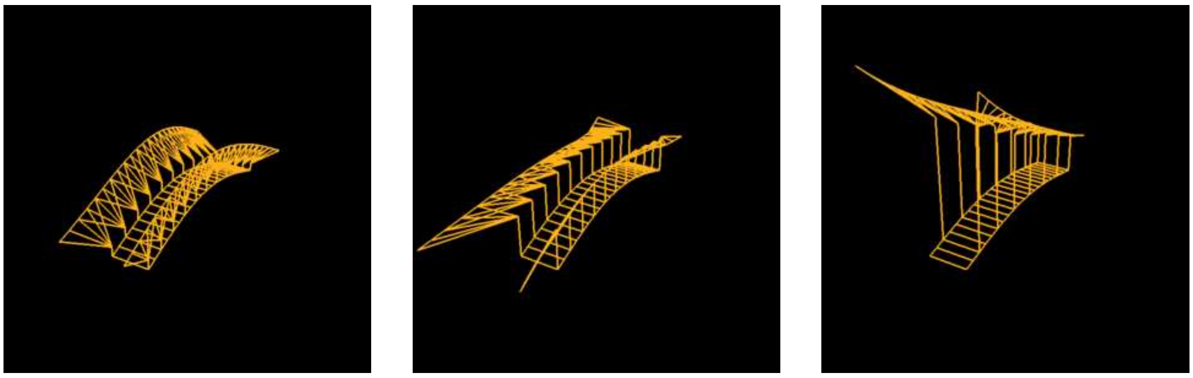
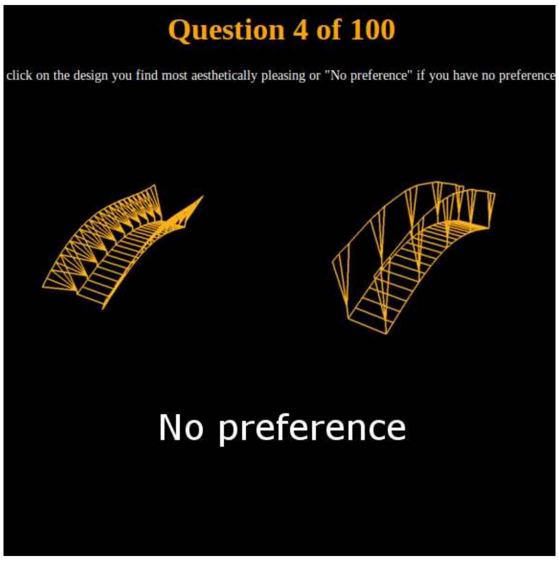


Fig. 4.8: Sample bridges from the survey.



Question 4 of 100

click on the design you find most aesthetically pleasing or "No preference" if you have no preference

Fig. 4.9: The layout of the survey.

4.4.3 Categorisation Results

The survey was completed by 28 individuals and consisted of 2800 evaluations. The results for the survey are shown in Figure 4.10. The users showed a preference of 55.9% for bridges that were highly dominated compared to a 36.84% preference for non-dominated bridges on the pareto front. The users had no preference on 7.26% of the designs. Although the user had three categories to select from, the instances where the user had no preference are ignored as they provided no indication of a bias towards either group.

A binomial test for significance was performed on the two groups where users had expressed a preference. The binomial test assumed that the distribution for the null hypothesis was 0.5, i.e., that the users had a 50% chance of selecting one group over the other. The result returned a p-value of less than 2.2×10^{-16} meaning that the survey results were statistically significant and that the null hypothesis was rejected. These results show that users had an aesthetic preference for designs that do not fulfill the engineering constraints.

4.4.4 Discussion

Although objective fitness functions have been successfully used for evolving aesthetically pleasing designs [118, 153, 192], the use of physical constraints did not generate aesthetically pleasing bridges. Putting functionality first did not result in interesting forms. The results imply that the physical constraints that were chosen for this experiment did not match the aesthetic preferences of the survey participants. Although aesthetic preference is a predominantly a subjective quality, the inclination towards unconstrained designs could be because of their unusual and unexpected configurations rather than the "ordinary" nature of structurally sound designs. The results indicate that grouping images by fast non-dominated sort creates two distinct categories.

The users preferred the designs that were not on the pareto front. What this means is that only presenting the user with individuals from the pareto front would excessively inhibit exploration of the search space. The user should be allowed to explore the possibilities first and then their chosen designs should be optimised rather than reducing the search space based on the objective constraints.

4.5 Conclusion

Design can be described as a purposeful yet explorative activity [63]. When architects design, they do more than simply optimise. Initially the designer must explore the search space to find a concept or form that is capable of fulfilling the design specification. Once the form has been chosen, the design process focuses on satisfying the constraints of the

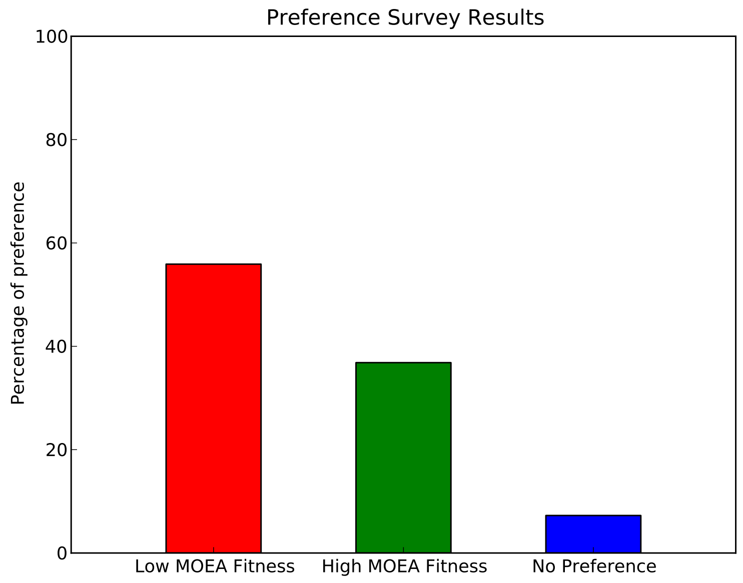


Fig. 4.10: Designs that were not on the pareto front were preferred by the users.

original design specification.

At the centre of this process there is a conflict between form and function. The heuristics an architect uses to evaluate a design are not the same as a structural engineer. Architects evaluate all aspects of the design, from broader issues of internal and external relationships to more detailed aesthetic measures such as material use, texture and light.

By eliminating designs that are less instantiable the design space presented to the user is reduced, but it also reduces the algorithm's capacity for design exploration. Accordingly, the intentionality of the architect should direct the search from the beginning of the search process. The application of an automatic fitness function is more suited for application during the later stages of design. The design selected by the architect would be developed into a more realisable form by optimising the physical constraints.

In this chapter material and physical constraints were encoded into a fitness function and it was shown to be a suitable methodology for evolving conceptual designs towards those objectives. This is a step towards making conceptual designs more realisable. It was also shown that multi-objective fitness functions are suitable for more than optimisation.

The NSGA2 algorithm allows for designs to be grouped into distinct categories based on their instantiability, although the categories preferred by the user are not on the pareto-front.

Using an automated fitness function based on physical constraints is beneficial for optimising designs but reduces the search space too much for design exploration. If it was possible for the users to direct search to areas of interest, this could provide a mechanism for design exploration. The following chapter explores this possibility by examining mutation in detail with the intention of developing user operators for directing search.

Chapter 5

An Analysis of Mutation in Grammatical Evolution

The intention of this thesis is to analyse the GE algorithm and to apply GE to architectural design. To this end, a study of traditional GE operators is performed with the intention of developing operators that could be used by the designer to direct search. The use of an automated fitness function showed that designs could be optimised using physical constraints but it did not provide a means of exploring the search space. The goal of developing user operators is to allow the user to vary their preferred designs and explore the surrounding search space.

This study decomposes the behaviour of mutation in Grammatical Evolution and examines the effect of mutation on fitness. Standard integer mutation in GE can be divided into two types of events, those that are structural in nature and those that are nodal. A structural event alters an internal node of a derivation tree whereas a nodal event simply alters the value of a terminal node of a derivation tree. The experiments carried out in this chapter analyse standard integer mutation and compare the behaviour of its nodal and structural components. This study increases the understanding of how the search operators of an evolutionary algorithm behave.

Much attention has been directed towards the behaviour of crossover in grammatical evolution due to the traditional importance placed on this search operator in Genetic Programming in general, e.g., [103, 140, 146, 74]). However, there has been little analysis of the behaviour of mutation on search in GE. Two papers that have explored mutation events in GE are the study by Rothlauf and Oetzel [165] on the locality of binary mutation and the study by Hugosson et al. [87] comparing the performance of different mutation operators on binary and gray code genotype representations. This study extends previous research by examining the different behavioural components of standard integer mutation and exploring how these components move the solution through the search space.

Locality - how well neighbouring genotypes correspond to neighbouring phenotypes - has been described as a key element in Evolutionary Computation [114]. It is critical to understand how the behaviour of mutation effects locality so that more efficient search operators may be designed. This study explores this important research gap by conducting an analysis of the behaviour of GE's mutation operator. This work examines what types of change occur on the phenotype during a mutation event and their impact on evolutionary performance.

The remainder of the chapter is structured as follows. Firstly, the related research in this area is discussed in Section 5.1. An analysis of the different behaviours of integer mutation in GE is undertaken in Section 5.2. We define two different types of mutation events, nodal and structural mutation, in Section 5.3. The procedure, grammar and settings for the benchmark problems used in this experiment are discussed in Sections 5.4.1 and 5.4. The nodal and structural mutation operators are applied to the benchmark problems to examine their effect on fitness in Section 5.5. A detailed examination of individual mutation events is then conducted in Section 5.6. The conclusions are discussed in Section 5.7.

5.1 Related Research

Locality is a metric that describes the similarity between genotypic and phenotypic neighbourhoods [164]. If a metric can be applied to a representation then the distance between two individuals using that representation can be computed. Two individuals are neighbours if their distance is minimal. Locality compares if neighbourhoods remain consistent after one representation is mapped to another. A mapping is *local* if neighbourhood is preserved under that mapping [114]. The definition of locality assumes that a distance measure exists on both genotype and phenotype spaces, and that for each there is a minimum distance, and that a neighbourhood can be defined in terms of minimum distance.

Rothlauf's research distinguished two degrees of locality: high and low locality. A representation has high locality if all neighbouring genotypes correspond to neighbouring phenotypes. On the other hand, a representation has low locality if many neighbouring genotypes do not correspond to neighbouring phenotypes. Low locality is undesirable in an operator as it is akin to random search. Rothlauf demonstrates that a representation that has high locality is necessary for efficient evolutionary search.

There has already been some investigation into locality in GE. The locality study by Rothlauf and Oetzel [165] on binary mutation investigated how binary changes in the genome impacted the derivation tree. It was found that in some cases (less than ten percent of the time) mutation events can result in small changes to the genotype resulting in not so small changes to the structures generated. More specifically, given a single unit of change at the genotype level (i.e., a bit flip), changes of greater than one unit of change at the phenotypic tree level occurred approximately ten percent of the time. 14% of these had a distance of greater than 5 units at the tree level. A unit of change at the phenotypic tree level corresponded to tree edit distance calculations which included deletion (delete a node from the tree), insertion (insert a node into the tree) and replacement (change a node label) change types.

Rothlauf also states that 90% of the time mutation has no effect due to the many-to-one mapping adopted in GE, although this is highly dependent on the representation used. The many-to-one mapping allows multiple codon values to correspond to the same production rule choice. Encoding a rule with two possible productions using an eight bit codon means seven of the eight bits are redundant. Only mutating the least significant bit would have any effect on rule choice. This would reduce the effective rate of mutation is reduced from 50% (for an integer encoding) to 12.5%.

This study was extended by Hugosson et al. [87] to compare the performance of binary and integer forms of mutation. Their work showed that using an integer encoding on certain problems produced significantly different results to a binary encoding. It is not clear how invalid phenotypes (i.e., phenotypes that are incompletely mapped containing at least one non-terminal symbol) were handled in Rothlauf and Oetzel's study. It is possible that any remaining non-terminal symbols were treated as different node content values in the tree distance calculations, but this is not exposed in their study.

In this chapter attention is turned to what is occurring when a unit of change arising from mutation at the genotype level is not perfectly correlated with a unit of change at the phenotype level. This work intends to establish if it is possible to design a mutation-based search operator that exhibits better properties of locality and search than is currently adopted. The next section defines different types of mutation event in GE.

5.2 A Component-Based View of Mutation in GE

In order to expose the impact of mutation on derivation tree structure a simple grammar a simple grammar is designed that uses binary rule choices. This allows us to condense codons (elements in the string representing the individual) to single bits. Below is a simple binary grammar which might be used in the case of application to a symbolic regression type problem with two variables (`x` and `y`).

```
<e> ::= <o><e><e>  (0)
      | <v>        (1)


<o> ::= +          (0)
      | *          (1)


<v> ::= x          (0)
      | y          (1)
```

Fig. 5.1: A simple binary grammar for creating functions.

This allows for the creation of genomes with binary valued codons to construct sentences in the language described by this grammar. Consider all genomes of length two codons (2^2 of them) and draw an edge between genomes that are a Hamming distance of one apart. If the corresponding partial derivation trees resulting from those genomes are presented as a graph the arrangement outlined in Fig. 5.2 can be seen. In this particular example we see that a mutation event at the first codon corresponds to a new derivation tree structure. Here a new derivation tree structure is defined as being one that has changed in length, i.e. , it contains a different amount of non-terminal symbols than its neighbour. Mutations from `00` to `10` (and vice versa) and from `01` to `11` (and vice versa) result in these structural changes, whereas the remaining mutation events result in node relabelling.

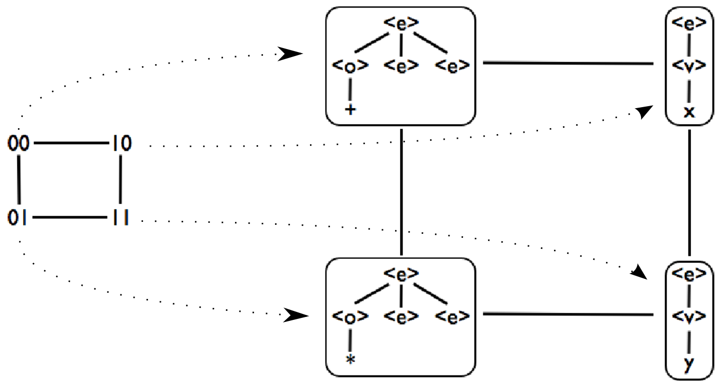


Fig. 5.2: The 2D neighbourhood for the example grammar (i.e., using the first two codons).

Extending the genomes by an additional codon the Hamming neighbourhood between the 2^3 genomes can be visualised both in terms of genome codon values and partial phenotype structures. These are illustrated in Fig. 5.3. Again, there is a clear distinction between mutation events that result in structural and non-structural modifications.

Mapping these codons back to the grammar it can be seen that structural mutations occur in the context of a single non-terminal symbol, `<e>`. It can be seen from this grammar that this non-terminal alone is responsible for structural changes, as it alone can increase the size of the developing structure. The rules for the `<o>` and `<v>` non-terminals are non-structural as they simply replace an existing symbol without changing structural

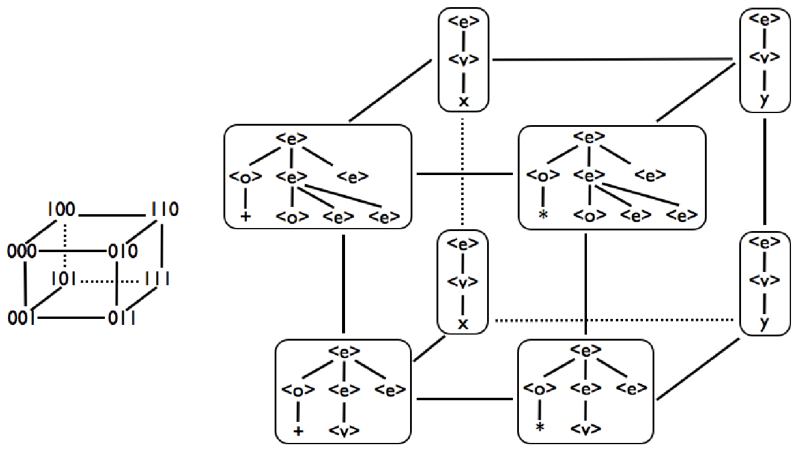
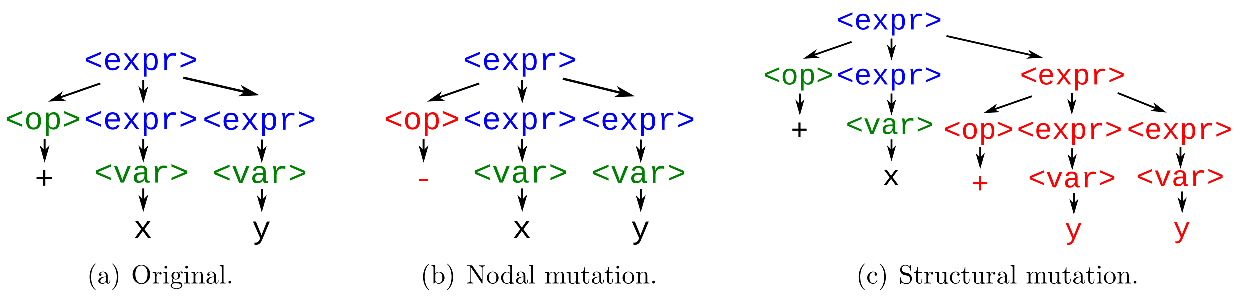


Fig. 5.3: The 3D neighbourhood for the example grammar (i.e., using the first three codons).

length. Another instance of this is shown in Figure 5.4. Both figures 5.4(b) and 5.4(c) are mutations of Hamming distance one from Figure 5.4(a). While the nodal mutation in Figure 5.4(b) changes only one leaf, the structural mutation in Figure 5.4(c) adds a whole new subtree.



(a) Original. (b) Nodal mutation. (c) Structural mutation.

Fig. 5.4: Nodal (green) and structural (blue) nodes of a derivation tree.

It should be stated that it is possible to generate a grammar consisting entirely of nodal rules or of structural rules that map deterministically to nodal rules but this negates the point of using a grammar. The grammar allows for abstraction and modularity through the use of non-terminals. A grammar naturally forms a hierarchy of abstractions based on these non-terminals. What this chapter intend to investigate is whether mutating the abstractions or the literals can effect how the algorithm navigates the search space.

5.3 Definition of Nodal and Structural Mutation

The previous section highlighted two different component behaviours of integer mutation. Based on these findings, this section defines the component operators of integer mutation. Before nodal and structural mutation can be defined, context free grammars and GE's mapping process must be defined.

Definition 1 (Codon) c is an integer value that is used to select the production for a rule, where $c \in \mathbb{Z}$

Definition 2 (Chromosome) C is a finite list of length n that is composed of codons. $C = (c_1, c_2, .., c_n)$ where $c_1, c_2, .., c_n$ are codons

A Context-Free Grammar is a formal grammar where the rewriting of a non-terminal symbol is not dependent on the surrounding symbols. [22, 75]. GE uses codons to select rules from a CFG.

Definition 3 (GE Context-Free Grammar (GE-CFG)) A GE-CFG is a four tuple $G = \langle N, \Sigma, S, R \rangle$, where:

- N is a finite non-empty set of non-terminal symbols.
- Σ is a finite non-empty set of terminal symbols. $N \cap \Sigma = \emptyset$, the empty set. Σ^* is the set of all strings constructed from Σ and V^* is the set of all strings constructed from $N \cup \Sigma$
- S is the start symbol, $S \in N$.

- R is a finite set of rules such that $r_A : C \rightarrow V^* : r_A(c) = \alpha_i$ where $A \in N$, $\alpha_i \in V^*$, α_i is the ith production of r_A, $i \in \mathbb{Z}$, $0 < i \leq k$, where k is the number of productions for r_A and $r_A \in R$.

The derivation process begins with the derivation equal to the start symbol. It expands the leftmost non-terminal iteratively until the derivation contains no non-terminals.

Definition 4 (Derivation) the derivation $\delta = \alpha A \beta$ where:

- $\alpha \in \Sigma^*$, $A \in N$, and $\beta \in V^*$, A is the leftmost non-terminal (LMNT)
- $\delta_0 = S$ (i.e. $\alpha = \beta = \emptyset$)
- for $0 < i \leq n$ there exists $\delta_i = \alpha_i A_i \beta_i$
- for each step of the derivation $0 < i \leq n$: the LMNT of the derivation is rewritten using the codon at index i, such that $\delta_{i+1} = \alpha r_A(c_i)\beta$

Then nodal rules are the subset of rules where the productions of a rule only consist of terminal symbols.

Definition 5 (Nodal Rule) The set of nodal rules $R_{Nodal} \subseteq R$ is defined as $R_{Nodal} = \{r_A \in R : r_A(c) = \alpha_i \in \Sigma^* \; \forall c \in \mathbb{Z}\}$

The structural rules are the subset of rules where at least one production includes a non-terminal.

Definition 6 (Structural Rule) The set of structural rules $R_{Structural} \subseteq R$ is defined as $R \setminus R_{Nodal}$.

A mutation event changes the value of a codon in a chromosome.

Definition 7 (Mutation Event) A mutation event occurs at codon index i when the mutation function M is applied to $c_i \in C$ such that $M(C) = C'$ where $C = (c_1, c_2, .., c_i, .., c_n)$ and $C' = (c_1, c_2, .., c_i', .., c_n)$

Using the definitions above, standard GE mutation can be divided into two types of events, those that are structural in nature and those that are nodal.

A nodal mutation changes the value of a codon at index i that encodes for a nodal rule, thus changing the production value for that rule.

Definition 8 (Nodal Mutation) The mutation event at c_i is a nodal mutation event if the LMNT is rewritten such that $A_i \mapsto r_A(c_i)$ where $r_A \in R_{Nodal}$

Definition 9 (Structural Mutation) The mutation event at c_i is a structural mutation event if the LMNT is rewritten such that $A_i \mapsto r_A(c_i)$ where $r_A \in R_{Structural}$

Effectively the behaviour of mutation can be decomposed into two types of events. The first are events that are structural in their effect and the second are those which are nodal in their effect. We could consider both types of events as operators in their own right, and therefore define a *structural mutation* and a *nodal mutation*. It should be noted, however, that this is a specialisation of integer mutation in GE, as it is possible for both types of events to occur during standard application of GE mutation to an individual's chromosome. *Perhaps the search characteristics of mutation can be improved by applying the components of integer mutation with differing locality to a problem?* The following analysis and experiments seek to determine the relative importance of these behavioural components of mutation and begin to answer these kinds of questions.

If mutation events can be decomposed into different categories based on their locality, there should be an observable difference in how they explore the search space. It is now possible to investigate if there is a discernible difference in the performance of the operators with regards to fitness. In the next section experiments are conducted to examine if there is a significant difference between the operators.

5.4 Applying Nodal and Structural Mutation to Benchmark Problems

This section gives an overview of the procedure, settings, and grammars used in this experiment. The five problems that were examined are symbolic regression, the Santa Fe ant trail, word match, even five parity and the max problem. The Santa Fe ant trail, even five parity, and symbolic regression are the same benchmarks as used by Koza [103]. Word match is a specific alphabet based instance of a pattern matching task and is a suitable benchmark as it allows the difficulty to be scaled by increasing the size of the pattern. The max problem has been used as a standard problem for GP by Gathercole and Ross [62] and Langdon and Poli [105].

5.4.1 Experimental Procedure

This section describes how the experiment was implemented and the settings that were used. The experiments carried out in this chapter were implemented using GEVA version 1.2 [147, 148], this is an open source framework for grammatical evolution in Java designed by the Natural Computation Research and Applications group in University College Dublin, Ireland. It can be downloaded from the NCRA website [141]. Crossover was turned off so that the experiment could focus solely on the impact of mutation. A number of properties were kept constant over the experiment execution. The experimental settings are shown in Table 5.1. For consistency the fitness for all of the experiments are minimising, i.e., smaller fitness values are improvements. Each of the benchmark problems shall now be described in detail.

5.4.2 Symbolic Regression

Symbolic regression is the task of finding a function that matches a set of observed points for a target function. The grammar used in this experiment is shown in Figure 5.5. The grammar generates polynomial equations and consists of two nodal rules defining the op-

Tab. 5.1: Experimental Settings.

Property	Setting
Population Size	500
Generations	50
No. of Runs	30
Mutation Rate	1%
Crossover Rate	0%
Selection	Tournament Selection
Tournament Size	3
Replacement	Generational
Elite Size	1
Initialiser	Ramped Half and Half
Wrapping	off
Random Number Generator	Mersenne Twister
Seeding	System Clock

erations and the variables and one recursive structural rule. The target function used in this experiment is the polynomial $x^4 + x^3 + x^2 + x$. Twenty fixed sampling points between -1 and 1 were used as test cases to calculate fitness for the individuals. The fitness was computed by summing the difference between the evolved and target functions for these test cases.

```
<expr> ::= <op> <expr> <expr> | <var>
<op>   ::= + | - | * | /
<var>  ::= x | 1.0
```

Fig. 5.5: The grammar for symbolic regression.

5.4.3 Santa Fe Ant Trail

The goal of the Santa Fe ant trail is to find a program for controlling the movement of an artificial ant in order to find all of the food lying on an irregular trail. The trail exists on a 32 by 32 two-dimensional toroidal grid. 89 pieces of food are located along a broken trail, and the ant has 600 units of energy to find all the food. The grammar shown in Figure 5.6 consists of conditions and operations. The conditions allow the ant to sense if there is food in the single square it is currently facing. The operations allow the ant to turn right, turn left, look forward one square and move forward one square. All operations requiring one energy unit.

The grammar consists of three structural rules and one nodal rule. Two of the structural rules are recursive and allow the derivation tree to grow while the nodal rule controls the operations of the ant. There is a maximum amount of energy that the ant can use; after all 600 energy units are used the number of food left on the trail is counted. The fitness for the Santa Fe ant trail is the total amount of food minus the amount of food collected by the time the energy is exhausted.

```
<code>      ::= <line> | <code> <line>
<line>      ::= <condition> | <op>
<condition> ::= if(food_ahead()==1):
                  { <code> }
                else:
                  { <code> }
<op>        ::= left(); | right(); | move();
```

Fig. 5.6: The grammar for Santa Fe ant trail.

5.4.4 Word Match

The goal of word match is to generate a string that matches a target string. Fitness is evaluated by counting of the number of characters of the candidate solution which match the target string. The grammar shown in Figure 5.7 consists of two nodal rules and two structural rules. The match count is subtracted from the total character count (n) of the target string so that the results minimise. The target word used for this experiment is ”experimental”.

```
<string>    ::= <letter> | <letter> <string>
<letter>    ::= <vowel> | <consonant>
<vowel>     ::= a|o|u|e|i
<consonant> ::= q|w|r|t|y|p|s|d|f|g|h|j|k|l|z|x|c|v|b|n|m
```

Fig. 5.7: The grammar for word match.

5.4.5 Even Five Parity

The goal of even five parity is to sum the number of bits set to one and return true if that value is even. The grammar shown in Figure 5.8 consists of a recursive structural rule and two nodal rules for the boolean operators and inputs. The fitness function for this problem is the total number of test cases minus the number of test cases correctly classified.

```
<expr> ::= <op> <expr> <expr> | <var> | not <var>
<op>   ::= "|" | & | ^
<var>  ::= d0 | d1 | d2 | d3 | d4
```

Fig. 5.8: The grammar for even five parity.

5.4.6 Max Problem

The max problem tries to generate the largest number possible with a given terminal and functional set and with a fixed tree depth limit. Despite seeming to be a simple problem the fitness landscape is deceptive and leads to suboptimal solutions. This problem is an interesting application for the new component behaviours of the mutation operator as the grammar, shown in Figure 5.9, consists of only one structural rule(`<expr>`) and one nodal rule(`<op>`). The max problem requires that every element of the tree contributes to the final solution, i.e., there are no introns [105], which further removes any confounding factors.

There is a globally optimal solution for a specific depth, D, given by the formula $4^{2^{D-3}}$. A ramped half and half initialiser was used to create the derivation trees with an initialisation depth of 8. This equates to a phenotype tree of depth 6, the maximum depth allowed for this problem. This was necessary because nodal mutation by itself cannot alter the length of a phenotype and it required trees initialised to the maximum depth for a fair comparison. The fitness for the max problem is the value generated by the parse tree subtracted from the globally optimal value so that the fitness minimises.

Now that the problems have been defined, the next section describes in detail how the operators were tested.

```
<expr> ::= <op> <expr> <expr> | <var>
<op>   ::= + | *
<var>  ::= 0.5
```

Fig. 5.9: The grammar for the max problem.

Tab. 5.2: Fitness functions for benchmark problems.

Problem	Target	Fitness Function
Even Five Parity	Even == True	Hamming distance from target value
Santa Fe Ant Trail	Food	Total food minus food eaten
Symbolic Regression	$x^4 + x^3 + x^2 + x$	Summation of test case error values
Word Match	“experimental”	Character Hamming distance
Max Problem	$4^{2^{D-3}}$	Global optimum minus evaluated result

5.5 Analysis of Fitness

The goal of this experiment is to investigate whether the structural and nodal components of integer mutation have different search characteristics when applied to the benchmark problems. Every problem was tested without crossover so as to avoid any confounding effects. For each test problem four operators were tested, nodal mutation, structural mutation, integer mutation and variable mutation.

The variable mutation operator is a composite of nodal and structural mutation and has two mutation rates. The nodal mutation rate is initialised at zero and the structural mutation rate is set at the maximum mutation rate (1% for this experiment). The probability of nodal mutation increases linearly from zero to the maximum mutation rate during the course of a run while structural mutation rate does the opposite. At any point in the experiment the sum of the structural and nodal mutation probabilities equalled the overall probability. Variable mutation investigates if changing from structural to nodal mutation operators during the run improved search characteristics. Parameter control was shown by Eiben et al. [55] to be beneficial to search.

Each of these operators were tested with a per codon probability of mutation of 1%. As each operator was applied to subsets of the genome of different size, the total amount of mutations differed for each operator. The number of mutation events for integer mutation ended up being higher as the mutation probability was applied to *every* codon, whereas the other operators were only applied to a subset of these codons. As there was a lower number of mutation events for the component operators, integer mutation explored a greater area of the search space. The result would be a bias in favour of integer mutation.

Hypothesis

Integer mutation in GE can be categorised into component operators based on the locality of their changes to the derivation tree. Given the best fitness results after 50 generations for each mutation operator: μ_0, integer mutation, μ_1 structural mutation, μ_2 nodal mutation and μ_3, variable mutation, the following hypothesis is stated:

H_0 There will be no statistically significant difference in performance for any of the mutation experiments, i.e. $\mu_0 = \mu_1 = \mu_2 = \mu_3$

H_1 There is a statistically significant difference in performance for a particular operator, i.e. $\mu_0 \neq \mu_1 \parallel \mu_0 \neq \mu_2 \parallel \mu_0 \neq \mu_3 \parallel \mu_1 \neq \mu_2 \parallel \mu_1 \neq \mu_3 \parallel \mu_2 \neq \mu_3$

α The significance level (α) of the Wilcoxon rank-sum is 0.05.

5.5.1 Results

A Wilcoxon rank-sum (two-tailed, unpaired data) test was carried out for each experiment to test for significance. A pairwise Wilcoxon rank-sum was used for the operator comparisons, pairwise testing adds corrections for multiple testing. This bias of the mutation rate in favour of integer mutation should be taken into consideration when interpreting the results below

Even Five Parity Results

The results for this experiment are shown in Figure 5.10. Integer, nodal and variable mutation performed in a statistically identical fashion. This would imply that there are an adequate distribution of tree structures in the initial population that nodal mutation is able to optimise. Structural mutation produced only limited fitness gains that plateau early in the run. This result shows structural mutation could find improved tree structures but it was not able to optimise them to the extent that nodal or integer mutation could. This finding shows that structural mutation performs differently to nodal mutation on the even five parity problem and so rejects the null hypothesis. It also shows that only mutating a subset of the genome is capable of producing results that were not statistically different to full integer mutation.

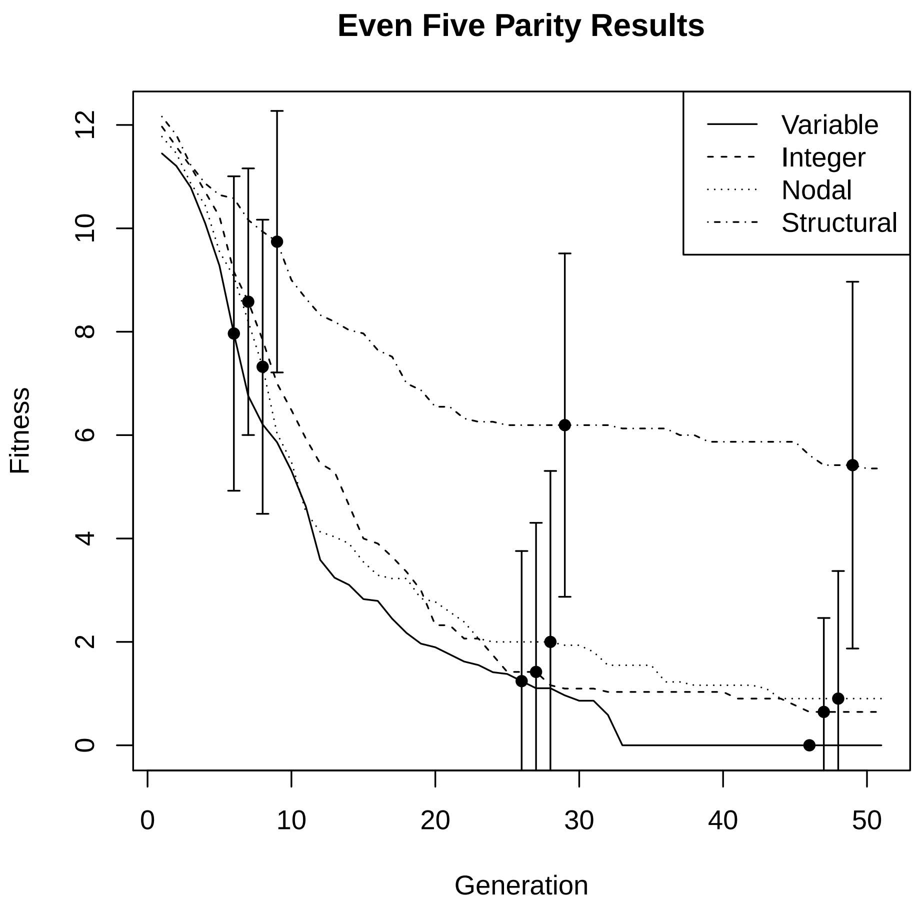


Fig. 5.10: Results for even five parity.

Santa Fe Ant Trail Results

The results for this experiment are shown in Figure 5.11. There was a statistically significant difference in performance for nodal and structural mutation over the other mutation operators and so the null hypothesis is rejected. The variance for structural mutation is much higher implying that it produced both positive and negative impacts on fitness but exploring the derivation tree structure had a significant benefit for the Santa Fe Ant Trail. Nodal mutation performed worst overall and had little improvement during the course of the run.

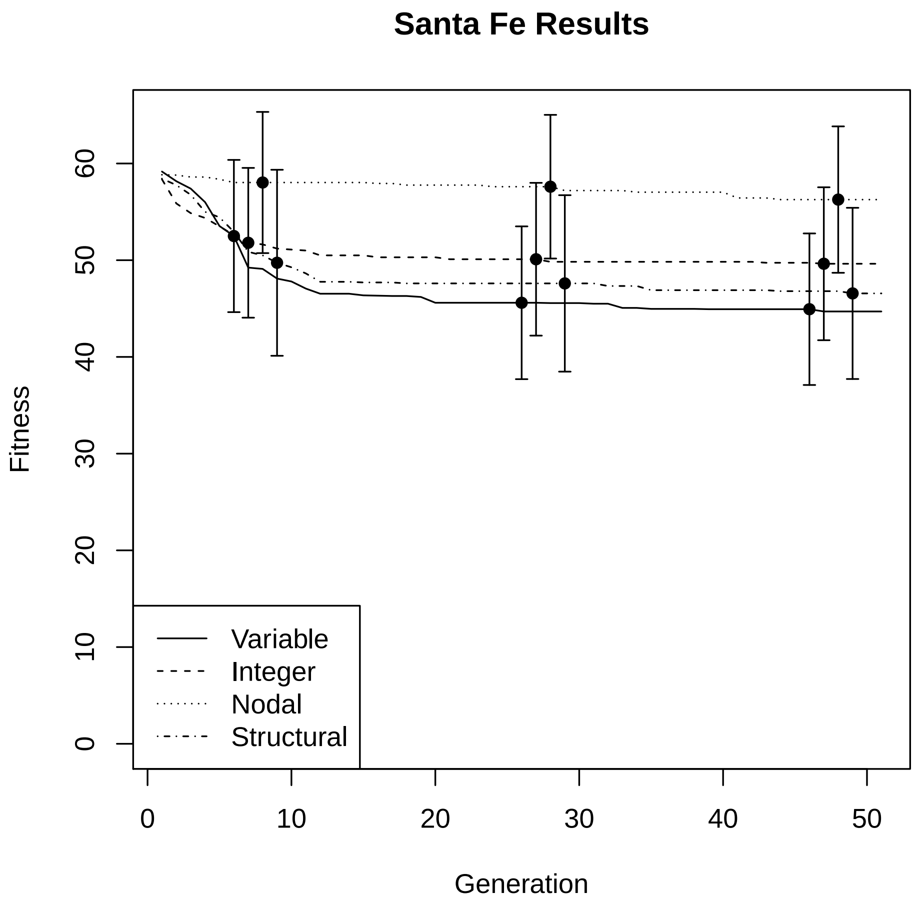


Fig. 5.11: Results for Santa Fe ant trail.

Symbolic Regression Results

The results for this experiment are shown in Figure 5.12. The Wilcoxon rank-sum test showed that there was statistically significant difference in performance for nodal mutation and so the null hypothesis is rejected. Overall nodal mutation performed worse than the other operations.

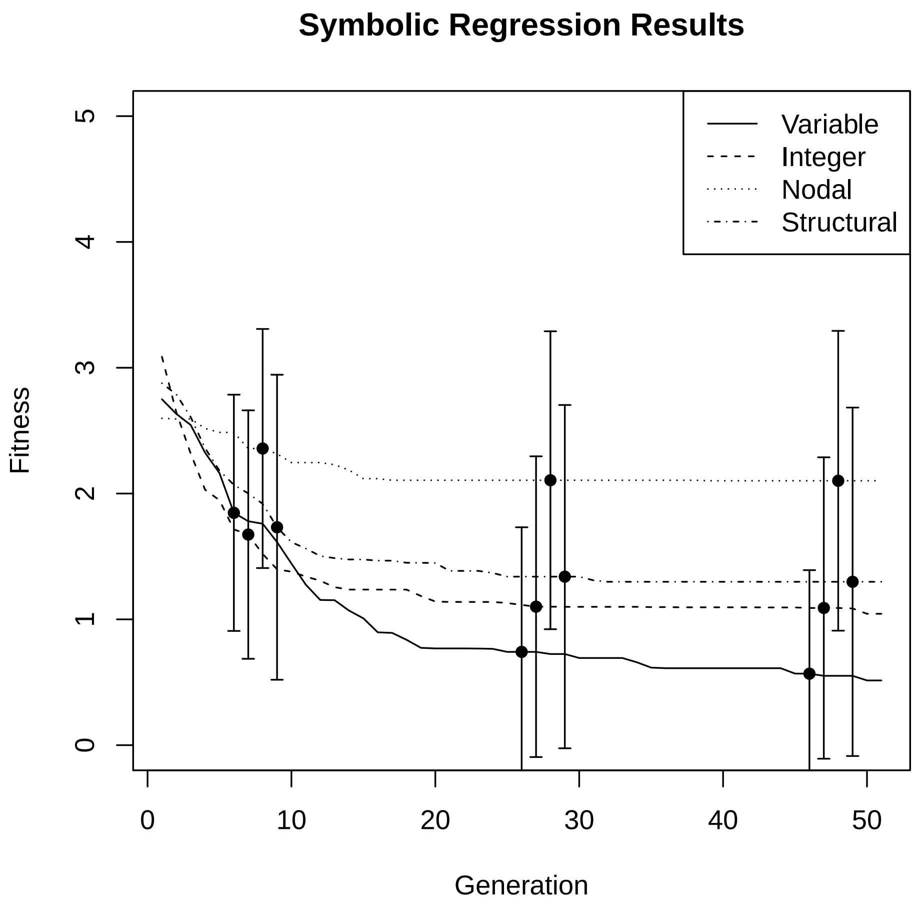


Fig. 5.12: Results for symbolic regression.

Word Match Results

The results for word match are shown in Figure 5.13. Integer, nodal and variable mutation again perform in a statistically identical way. This shows mutations occurring in the nodal sections of the genome are having the most impact on fitness. Although there is still a large amount of fitness variance using structural mutation, there is little overall improvement after 50 generations. Again this would indicate that the initial population for nodal mutation had a sufficiently diverse distribution of tree structures so that structural mutation did not infer any advantage.

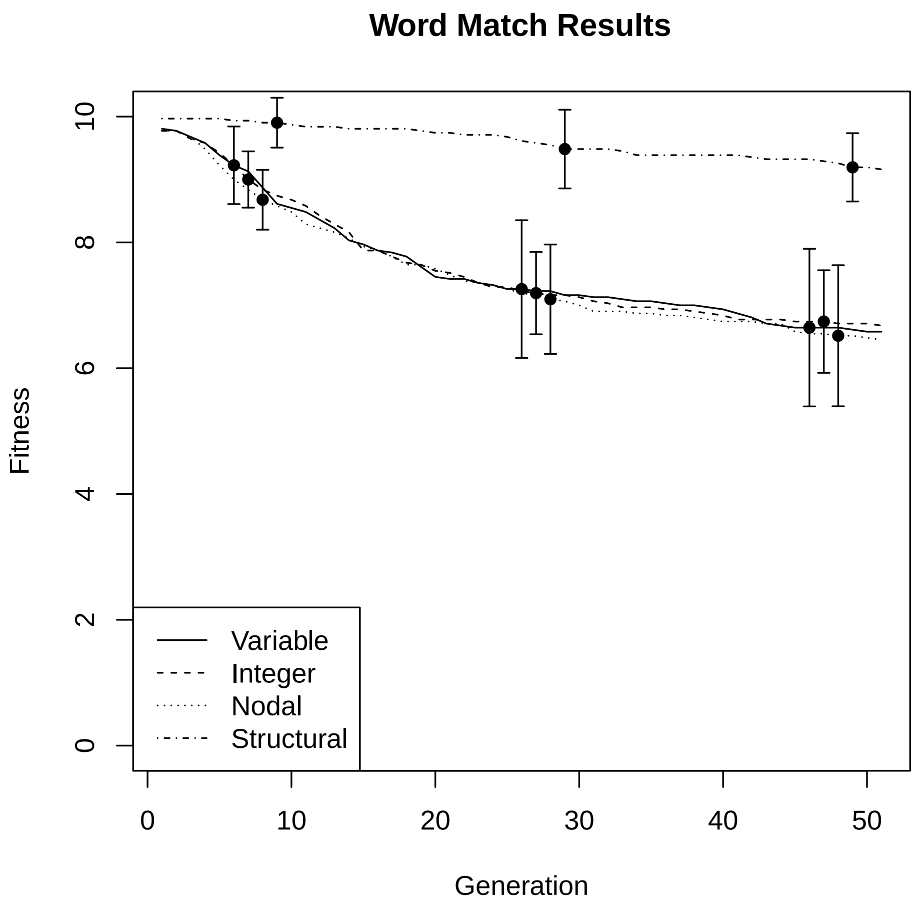


Fig. 5.13: Results for word match.

Max Problem Results

The results for the max problem are shown in Figure 5.14. The Wilcoxon rank-sum showed that while integer and variable mutation produced similar results, both nodal and structural mutation had significantly different behaviour. Nodal mutation produced much better fitness than any of the other operators while structural mutation produced no fitness improvement during the course of the run.

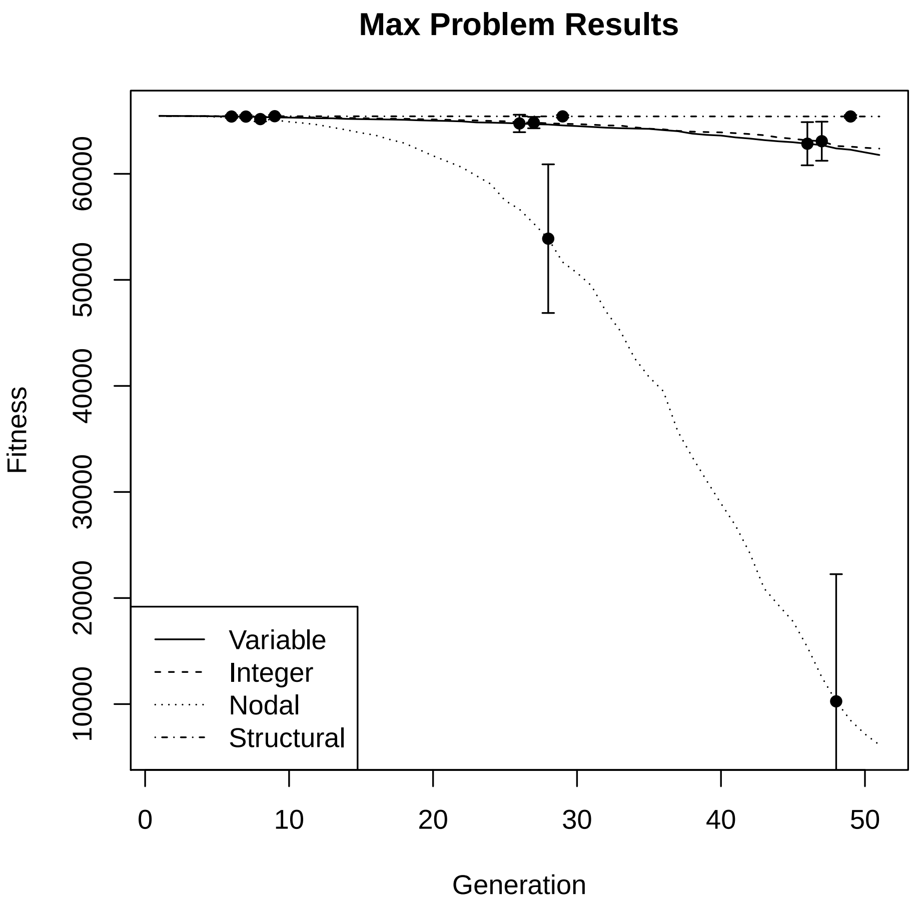


Fig. 5.14: Results for word match.

Langdon and Poli [105] and Gathercole and Ross [62] both used the max problem to study GP crossover and it is considered difficult for GP as populations converge quickly on suboptimal solutions that are difficult to escape from, except through a randomised

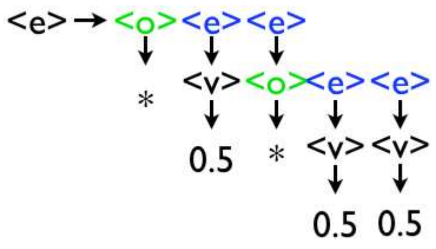
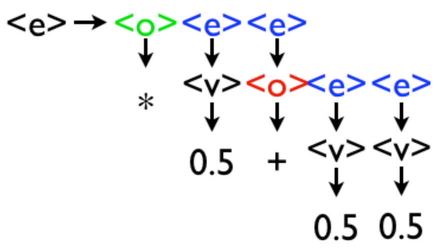
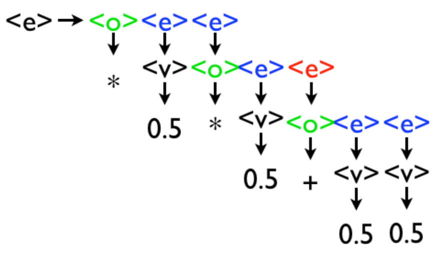


Fig. 5.15: An Example max problem parsetree. Total =0.125.

search [105]. The aim of the max problem is to generate a tree that returns the largest real value within a set depth limit. The optimal solution to this problem is to have addition operators at the leaf nodes of the tree so that it creates a large enough variable (greater than one) for multiplication to have an effect.



(a) Nodal mutation of Figure 5.15. Total = 0.5.



(b) Structural mutation of Figure 5.15. Total = 0.25.

Fig. 5.16: Different fitness results for nodal and structural mutation.

Structural mutation was responsible for exploring tree structures whereas nodal mutation optimised the nodes of a tree structure, as shown in Figures 5.15 and 5.16. The ramped half-and-half initialisation gave nodal an unfair advantage as it started with a broad sample of different tree depths to optimise. As nodal mutation cannot alter the structure of a tree, a different initialiser that did not fully explore the tree depth could have resulted in far worse performance. Despite this, the fact that nodal and structural mutation both perform differently to integer mutation clearly indicates that there are two separate behavioural components operating in integer mutation.

5.5.2 Discussion

The experimental results showed that selectively altering subsets of codons from the chromosome can have a dramatic effect on how GE navigates the search space. The different results from each benchmark problem also show the impact representation has on the effect of the operators. Both structural and nodal performed differently to each other on all of the benchmark problems. Nodal mutation performed in a significantly different way to integer mutation for symbolic regression and the max problem while structural mutation performed differently for the Santa Fe ant trail, the word match problem and even five parity problem. Every experiment showed that mutating subsets of codons could produce results that were quite different to those generated when mutating the whole genome.

Variable mutation matched the performance of integer mutation for all of the benchmark problems. Both operators act upon the whole chromosome. Focusing on different parts of the genome during the search produced no discernible benefit. Both nodal and structural mutation exhibit different performance on all of the benchmark problems. This evidence would suggest that how each operator navigates the search space has a different impact on fitness. The next section examines the impact of these mutation events in greater detail.

5.6 Analysis of Chromosomal Fitness

The previous experiment shows how each operator performs during the course of a run but they do not have the granularity to see what was actually happening during each mutation event. There was also a bias of using a per codon mutation rate when nodal, structural and integer mutation effected different numbers of codons. The next experiment analyses the operators' impact by examining individual, independent mutation events. In Rothlauf's study [165], changes to the derivation tree were recorded but as it is known what kind of phenotypic impact the operators have, this experiment instead looks at each operator's impact on fitness during individual codon changes.

This experiment is not intended to reflect how these search operators function during search. In practice the relationship is more complicated. For example, bad fitness results generated by an operator are bred out of the population and, conversely, a good mutation is always of benefit to a population regardless of its size. These metrics attempt to show some of the mechanics at play during mutation. This experiment highlights the effect of selectively altering codons that encoded for either a terminal or non-terminal production.

5.6.1 Hypothesis

Given a sample of fitness changes for each mutation operator: μ_0, integer mutation, μ_1 structural mutation and μ_2 nodal mutation, the following hypothesis is stated:

H_0 There will be no statistically significant difference in the scale of fitness change for the operators, i.e. $\mu_0 = \mu_1 = \mu_2$,

H_1 There is a statistically significant difference in fitness change for a particular operator, i.e. $\mu_0 \neq \mu_1 \parallel \mu_0 \neq \mu_2 \parallel \mu_1 \neq \mu_2$,

α The significance level (α) of the test is 0.05.

5.6.2 Setup

The experiment was run against the benchmark problems described in section 5.4 with the same settings. The fitness was recorded by generating a random individual and then traversing each codon with a 10% probability of mutation, when a mutation occurred the change in fitness was recorded and then the codon was returned to its original value. This meant that each mutation could be considered independent of those preceding it. If a mutation created an invalid individual then this was recorded but no penalty was added to the fitness. Assigning an arbitrary penalty value would distort the results, instead invalid individuals are examined separately. This was continued until a sample size of 10,000 mutations was gathered for each operator.

5.6.3 Results

The results for the experiments are shown in Tables 5.3, 5.4, 5.5, 5.6 and 5.7. A Wilcoxon rank-sum test was performed on the fitness change samples to examine if there is a discernible difference between the operators. It should also be noted that nodal mutation produced no invalids because, by its definition, cannot alter the structure of individuals, only substitute one terminal for another.

5.6.4 Even Five Parity Results

The results for the Wilcoxon rank-sum showed there was a significant difference between nodal and structural mutation. There was no statistical difference between integer mutation and its two component operators. Nodal mutation had fewer events that resulted in a change and when they did occur the change in fitness was less than that of a structural mutation event. Overall structural mutation had a greater impact on fitness.

Tab. 5.3: Results for even five parity.

Property	Nodal	Structural	Integer
Positive Mutations	97	397	127
Negative Mutations	327	1318	659
Neutral Mutations	9576	8207	9190
Invalids	0	58	24
Total Fitness Gain	98.0	420.0	135.0
Total Fitness Loss	361.0	1702.0	773.0
Average Gain	1.01 +- 0.1	1.06 +- 0.24	1.06 +- 0.24
Average Loss	1.1 +- 0.34	1.29 +- 0.56	1.17 +- 0.56

5.6.5 Santa Fe Ant Trail Results

The results for Wilcoxon rank-sum showed that there is a statistically significant difference between structural and nodal mutation and also that the distribution of nodal and integer mutations differs significantly. Nodal mutation produces more events that cause change although the scale of that is less for each mutation event. Overall nodal mutation had a greater impact on fitness.

Tab. 5.4: Results for Santa Fe ant trail.

Property	Nodal	Structural	Integer
Positive Mutations	1774	830	1183
Negative Mutations	3051	2158	2709
Neutral Mutations	5175	5520	5288
Invalids	0	1492	820
Total Fitness Gain	11280.0	5661.0	7627.0
Total Fitness Loss	19664.0	14620.0	18969.0
Average Gain	6.36 +- 5.87	6.82 +- 7.19	6.45 +- 7.19
Average Loss	6.4 +- 5.7	6.71 +- 5.55	6.95 +- 5.55

5.6.6 Symbolic Regression Results

The results for the Wilcoxon rank-sum showed there was a statistically significant difference between nodal and structural mutation and between nodal and integer mutation. Once again nodal produced more events that caused a change in fitness but the scale of change for each event was less in magnitude. Overall nodal mutation had a greater impact on fitness.

Tab. 5.5: Results for symbolic regression.

Property	Nodal	Structural	Integer
Positive Mutations	2328	1082	1140
Negative Mutations	3862	2706	2284
Neutral Mutations	3810	5265	6218
Invalids	0	947	358
Total Fitness Gain (log_{10})	3684.39	2336.29	1903.32
Total Fitness Loss (log_{10})	7918.61	8060.36	5258.42
Average Gain	1.58 +- 1.21	2.16 +- 1.12	1.67 +- 0.96
Average Loss	2.05 +- 1.25	2.98 +- 1.63	2.3 +- 1.39

5.6.7 Word Match Results

The results for the Wilcoxon rank-sum showed there was a statistically significant difference between nodal and structural mutation and between nodal and integer mutation. As nodal mutation can only change one character at a time, the fitness change is limited to plus or minus one. Nodal mutation events produce a greater number of changes than structural mutation events. Overall nodal mutation had a greater impact on fitness.

Tab. 5.6: Results for word match.

Property	Nodal	Structural	Integer
Positive Mutations	428	144	250
Negative Mutations	1519	1086	1064
Neutral Mutations	8053	8770	8686
Invalids	0	0	0
Total Fitness Gain	428.0	144.0	250.0
Total Fitness Loss	1519.0	1166.0	1130.0
Average Gain	1.0 +- 0.0	1.1 +- 0.44	1.08 +- 0.37
Average Loss	1.0 +- 0.0	1.11 +- 0.31	1.08 +- 0.28

5.6.8 Max Problem Results

The results for the Wilcoxon rank-sum show that nodal mutation had significantly different performance to the other operators. Nodal mutation generated the greatest number of beneficial mutation events followed by integer and structural while at the same time generating the most negative mutations. Structural mutation generated the most invalids on this experiment. Over a quarter of the mutation events produced invalids. Upon further investigation it was found that 2511 of structural invalids and 1173 of the integer invalids had surpassed the depth limit set for the problem. It should be noted that over half the time the GE mutation operators generated a neutral mutation. This was because of the structure of the grammar. Each rule had two possible outcomes, mutation would either change the outcome or leave it untouched. Overall, nodal mutation produced the best fitness increase and was the least destructive operator. The variance for the structural mutation operator (+- 27.95) showed that it was capable of generating very destructive mutations.

Tab. 5.7: Results for max problem.

Property	Nodal	Structural	Integer
Positive Mutations	2339	580	750
Negative Mutations	2715	1810	1375
Neutral Mutations	4946	5062	6638
Invalids	0	2548	1237
Fitness Gain	5762.10	442.31	1425.45
Fitness Loss	6236.46	8211.06	3925.26
Average Gain	2.46 +- 7.85	0.76 +- 2.17	1.9 +- 6.37
Average Loss	2.3 +- 6.7	4.54 +- 27.95	2.85 +- 6.01

5.6.9 Discussion

The results from this experiment show that selectively altering subsets of codons from the chromosome can have a dramatic effect on how GE navigates the search space. There was a statistically significant difference in fitness change between nodal and structural mutation on all of the benchmark problems and that the distribution of nodal mutations differed significantly from that of standard integer mutation on four of the five problems.

The analysis of chromosomal mutation showed that nodal mutation had a greater number of small fitness changes and would therefore be beneficial in fine tuning a solution. Individual nodal mutation events produced less fitness change than structural mutation but as more of the events occurred the overall change to fitness was greater. While the creation of invalids is generally regarded as detrimental to the search process, the large changes in fitness show that structural mutation explored different areas of the search space regardless of whether the result is beneficial to the population or not.

Structural mutation did not exploit a solution but instead performed a more global exploration of the search space. While the benefits of structural mutation were not seen in the chromosomal experiments, it would be wrong to think that just because the changes it made did not have an immediate beneficial effect that it had no effect at all. Nodal mutation showed that it was capable of optimising a structure but without the structural component of the mutation operator, different tree structures could not be explored. Exploration of different tree structures increases the diversity of the population.

The intention of this section was not to show which was the best operator, something which is problem dependent in any case [195]. Instead it was to decompose GE's integer mutation operator into its component behaviours. This information can now be used to create a mutation operator that applies these behaviours to a problem as and when they are needed. The use of intelligent operators might help GE escape local optima as well as find the optimal solution more efficiently. This could be of particular benefit in dynamic environments where an evolutionary algorithm must be able to adjust quickly in response to a changing fitness function [47].

This presents the idea that the differing levels of locality in integer mutation help it to explore different areas of the search space. Once the general structure of a good solution has been selected, the direction of GE's mutation could be focused on the content of that solution so that it may be further optimised.

5.7 Conclusions

This chapter analysed the behavior of mutation in GE. Two different components of integer mutation were investigated, nodal mutation and structural mutation. A third variable operator was created to examine the effects of applying these different operators during different stages of the search. The first set of experiments analysed how the operator performed on a set of benchmark problems by comparing their effects on population fitness. The results indicated that both nodal and structural mutation had a distinct impact on evolutionary performance and in some cases improved performance over standard integer mutation.

The second set of experiments then examined mutation operators at the chromosomal level to judge the impact of individual codon changes on fitness. The results once again showed that nodal and structural mutation events had a statistically significant different performance on all of the problems and in several instances nodal and structural mutation had a significantly different performance to integer mutation. The following chapter will conduct a further investigation into nodal and structural mutation by examining the locality of changes at different stages of the derivation process.

Chapter 6

Mutation Operators for Interactive Evolutionary Design

6.1 Introduction

Fitness evaluation is an indirect method for guiding search. After a fitness value has been assigned, the EA uses stochastic processes to generate new individuals. This can result in highly fit individuals not being selected for the next generation or, if they are selected, the child individuals bear little resemblance with their parents. To a designer unfamiliar with interactive evolutionary computation (IEC) this can result in confusion and frustration when interacting with the algorithm. Interactive evolutionary design presents the user with designs from different parts of the search space but when designers find an interesting individual they generally want to explore the search space around it. They perform a localised search to see if a design can be further improved.

One approach that allows users localised search functionality is to allow the user to apply operators directly to the individual. Localised search presents the user with many small variations on the original design thus allow the user to explore a design "theme", i.e., a group of designs with feature commonality. The advantage of using an operator to manipulate the design rather than direct manipulation of the design is any changes made to the phenotype are automatically encoded in the genotype, allowing for the new individual to be reintroduced into the population and then further evolved.

Direct application of mutation operators has been used to guide interactive evolutionary computation. Hart [76] allowed the user to specify which type of mutation (and mutation rate) to apply because of the assumption that users sometimes have an intuition about what type of change is required to an individual. This work stems from the study that was performed on interactive design evolution [150]. During this study it was found that the user applied the mutation operator to explore areas of the design space that had produced interesting aesthetic designs. For mutation to be useful in this circumstance, it requires locality to be maintained from the genotype to the output.

In GE, the evolutionary processes of the algorithm are carried out on the genome, which is then mapped to produce an output. The mapping process makes it different from standard GP as it goes though several transformations before it is finally output. This means that there are several different stages during the process where neighbours could be mapped to non-neighbours. If a user is to directly apply operators to a genotype then the effect should behave in an intuitive manner, i.e., small changes to the genotype should result in small changes to the phenotype. An example of this would be a representation for creating bridges that allowed for minor changes to be made to the walkway. If operators are to be developed for interactive evolution then the effect of GE's mapping process on locality must be examined. To do this distance metrics must be applied to different stages of the generative process. The distance metrics provide a quantitative measure for the operators and are used to record the locality of the changes they cause.

An operator with controllable locality provides a mechanism for user directed search during interactive design. If the user wants to indirectly influence a design through the use of an operator it must perform in a way that makes sense to the user. This type of interactivity in the evolutionary process is categorised as a human based genetic algorithm (HBGA) [186, 101]. If the user applies a small change to the genotype then they expect a small change in the phenotype. Low locality can also increase user fatigue as the user perceives it as a random search and will stop using the operator. This study examines the locality of the mutation operator when applied to designs in grammatical evolution.

The aim of this chapter is to investigate the impact of mutation on locality during the generative process. To perform the analysis both qualitative and quantitative techniques are used. The experiments examine how nodal and structural mutation effect the locality on different phenotypic stages when applied to the problem of interactive design generation. The subjective preferences of the user are then compared to explore if there is a correlation between the metric's idea of locality and the perceived locality at the final output stage

This chapter is organised as follows, the generative derivation process involved in GE is discussed in Section 6.2. Section 6.3 describes the distance measures that were used for comparing locality during the derivation process. Two experiments are conducted in this chapter. The first compares locality changes at different phenotypic stages in Section 6.4 and the second examines whether there is any perceivable locality difference between mutation operators in Section 6.5. Finally, in Section 6.6 the conclusions are discussed.

6.2 Generative processes in EC

The genotype to phenotype mapping is a terminology that has been adapted from the field of biology. A genotype is an encoding upon which the evolutionary operators of mutation and recombination act. An example of this would be human DNA. Changes in the genotype are translated into changes in the phenotype. A phenotype is an observable characteristic of an individual. An example of this in humans would be eye-color or height. Selection is performed on the phenotype. This biological concept was introduced to the field of EC as a method for separating the search and solution space [8] and as a metaphor for describing the representations and mapping processes.

It could be argued that in traditional GP there is no generative process. The trees that define the programs are directly manipulated by the operators, therefore there is no generative stage. Although, this is only true if you examine tree structures exclusively. The tree itself could be viewed as a genotypic encoding and the output of the program as

```
<expr> ::= (<op><expr><expr>)         (0)
         | <var>                      (1)
<op>   ::= +                          (0)
         | -                          (1)
<var>  ::= x                          (0)
         | y                          (1)
```

Fig. 6.1: A sample BNF grammar.

the phenotype. An example of this is the cellular encoding used by Koza [104] to create evolvable hardware. Furthermore, in Genetic Algorithms there is a mapping of the evolved integer string translating it into a meaningful application. This highlights that a genotype to phenotype to fitness mapping exists in all forms of evolutionary computation except the most basic one to one mappings.

In GE there are several stages in the mapping process, each with its own observable characteristics. As each stage also contains a version of the final instantiation, it can be classified as a phenotype. The three main phenotypic stages are shown in Figure 6.2. The integer list is translated into a derivation tree using a BNF grammar defined in Figure 6.1 and the mod rule. The terminal rules that make up the finished string are shaded. The string is then evaluated to produce the final output phenotype, in this case a 3 dimensional plot of a plane.

This study is interested in determining how differences in locality effect the different phenotypic stages of the generative process. In this study a single change on the genotypic stage is applied, the equivalent of a Hamming distance of one, and the corresponding locality of the change at each stage of derivation is observed.

To study locality, it is necessary to define a metric for measuring distances in the search space. In his work, Rothlauf claimed that for two different search spaces (e.g., genotypic and phenotypic search space) it is necessary to define two different metrics. As the experiment examines different phenotypic stages in the genotype-phenotype mapping representation, different metrics needed to be used for each one. The next section describes the metrics that were used for comparing different phenotypic output.

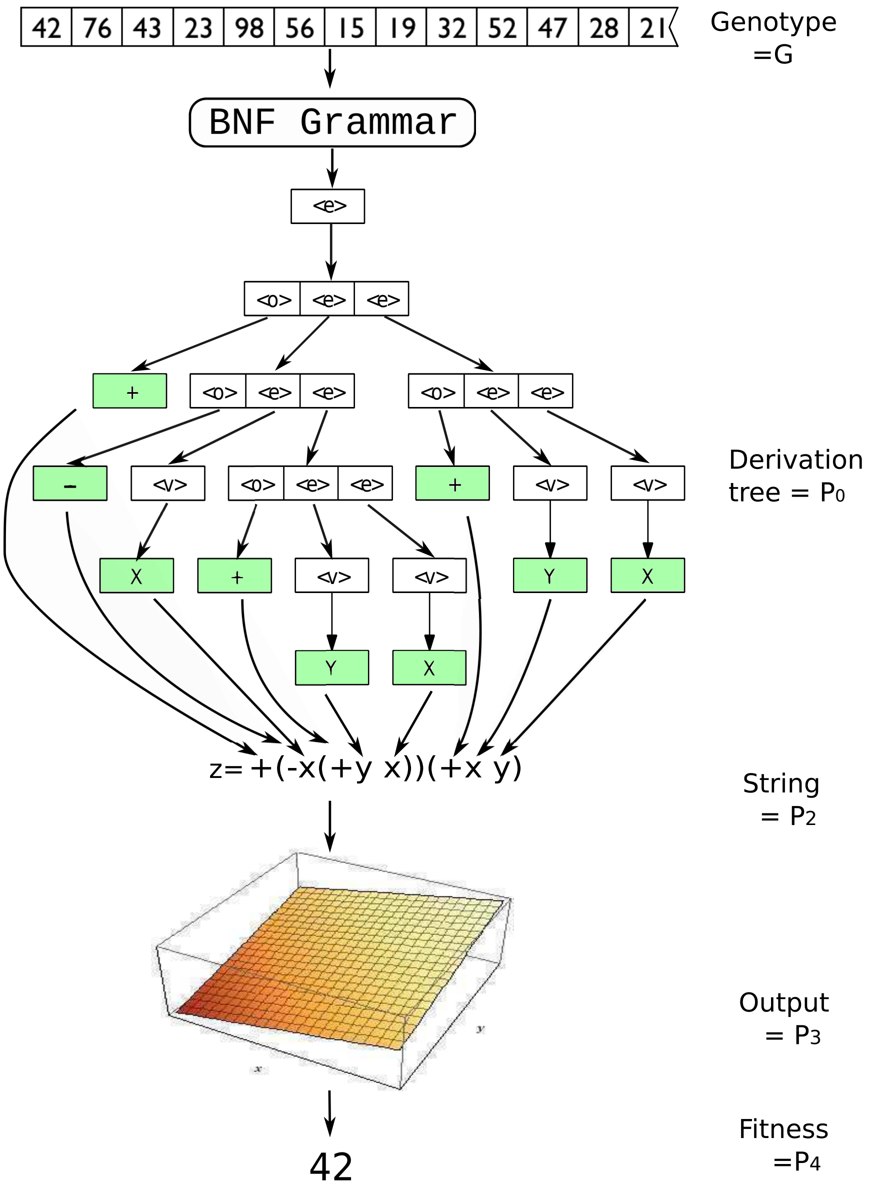


Fig. 6.2: stages of derivation in GE.

6.3 Distance Measures for Grammatical Evolution

This section reviews and provides motivation for the four distance measures, namely, tree edit distance, Levenshtein distance, normalised compression distance and Euclidean graph distance. A metric defines the distance relationship between elements of a set. For a function to be valid metric it must satisfy the following conditions: non-negativity, identity of indiscernibles, the triangle inequality and symmetry. If a function does not satisfy the above conditions it can still be used as a measure if it satisfies the following conditions: non-negativity, countable additivity and a null empty set. Three of the four measures below describe metrics with the exception of Euclidean graph distance. The justification for using Euclidean graph distance as a valid measure is described in detail in Section 6.3.4.

6.3.1 Tree Edit Distance

Traditional GP is based on programs that are structured as trees. Accordingly a method comparing the similarity of programs based on their tree structure was proposed by O'Reilly [152]. The distance is calculated by computing the minimum number of edits that is required to transform a given tree to a target. There are three possible edits that can be made: (a) Substitution: changing a node into another, (b) Insertion: adding a node within the tree and (c) Deletion: removing a node from the tree. The approach taken by O'Reilly was based on the work reported in [169, 184]. As GE does not use a traditional tree based approach to generating programs, the distance is applied instead to the derivation tree generated when the context free grammar is mapped to the individual.

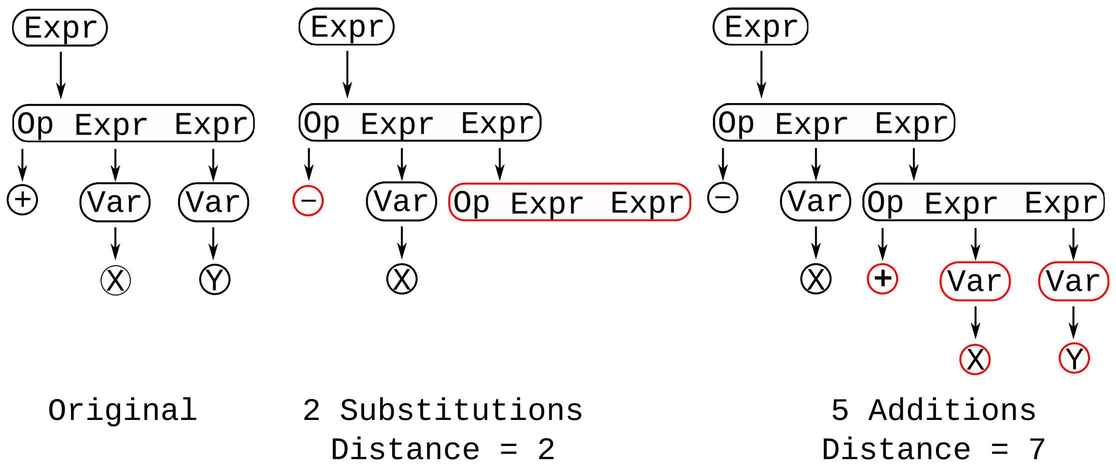


Fig. 6.3: Tree edit distance example.

6.3.2 Levenshtein Distance

Levenshtein distance is a metric for comparing two strings, or in this case, generated computer programs. This has been used as a metric to compare representations in GP [97] [89]. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. An example of calculating Levenshtein distance is given in Figure 6.4. In the study the output string is tokenised and the Levenshtein metric is applied to phenotype symbols.

1. chair $\rightarrow$ hair (deletion)
2. hair $\rightarrow$ fair (substitution)
3. fair $\rightarrow$ fairy (insertion)

Fig. 6.4: Levenshtein distance example, distance = 3.

6.3.3 Normalised Compression Distance

The normalised compression distance (NCD) is a distance metric based on the non-computable notion of Kolmogorov complexity [110]. It has been previously applied as a distance measure for linear structures within EC [67]. Kolmogorov complexity is a measure of how much information is required to fully describe an object [109]. NCD is based on normalised information distance (NID) which calculates the distance between two objects by defining objects in terms of their Kolmogorov complexity and then comparing the object descriptions.

As the Kolmogorov complexity of an object is not Turing computable, a more practical approach was taken by Cilibrasi and Vitanyi [36]. NCD approximates the Kolmogorov complexity by using off-the-shelf compression software to generate a compressed description of the objects. These descriptions are then compared using the following formula:

$$d(x, y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))}$$

where x and y are two strings, xy is their concatenation, and the function C gives the length of the compressed version of its argument. The comparison carried out in this experiment calculated the distance by applying the NCD metric to phenotype strings. The ZLIB compression library [49] was the algorithm used for computing the distance.

6.3.4 Euclidean Graph Distance

The program output generated graph representations of designs. The graphs consisted of nodes and edges in a three dimensional space. Examples of the output are shown in Figures 6.13, 6.14, and 6.15 on page 127. Euclidean graph distance uses a Euclidean distance measure to compare the nodes of the output designs. Euclidean distance is defined as the straight-line distance between two points on the same plane. The distance, d, is given by the formula:

$$d((x_1, y_1, z_1), (x_2, y_2, z_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

The graph with the most nodes is selected for iteration. Each node in the larger graph is then iterated through and the nearest node in the smaller graph is found. The distance between these nodes is then added to the total distance between designs, thus ensuring non-negativity, $d(a, b) \geq 0$.

The output graphs consist of points and the edges between them but the Euclidean distance formula only compares points in space. The edges must be taken into consideration if that $d(a, b) = 0$ if and only if $a = b$. Exhaustively checking how the nodes were connected proved to be computationally expensive. Instead, while computing the distance between node pairs, the number of outgoing edges for each pair is also compared. Any difference in the number of outgoing edges is added to the total fitness.

Two graphs, a and b, are considered identical if the distance for every node in graph a

and its nearest respective node in graph b is zero and the number of edges for each node pair is identical. While Euclidean graph distance does not fulfill all of the axioms required by a metric it does conform to the conditions of a premetric. The axioms a premetric must satisfy are: $d(a, b) \geq 0$ and $d(a, a) = 0$. Even though Euclidean graph distance is not a full metric, this does not prevent it from being a useful measure in practice [124, 71].

This section described four different metrics for comparing locality. The experiments described in the next section now use these metrics to compare the locality of nodal and structural mutation during the mapping process.

6.4 Locality During Derivation

The experiments described below examine the locality of phenotypic changes at different stages of the derivation process. The distance metrics record change at all stages of derivation. The experiment set out to investigate whether a single change at the genotypic level (Hamming distance one) would translate into a small or large change at the output stage. Traditionally different metrics are used to analyse different types of operators but as this experiment studies the different effects caused by components of the same operator, the same metric was used for both. The distance results were only recorded when both a structural and nodal mutation occurred, thus ignoring neutral mutation events.

The encoding that is examined in this chapter is a shape generating BNF grammar. This allows us to produce visual output that may be further examined by the user. In order to produce coherent yet complex shapes, a higher order function grammar for manipulating graphs was used. The next section explains the operation of the grammar in detail.

6.4.1 Higher-Order Function Grammar

Hornby [86] states that hierarchy, regularity and modularity are desirable qualities in a representation for design. McDermott et al. [126] and Yu [198] showed that grammars were capable of encoding such qualities through the use of higher-order functions (HOF).

Higher-order programming requires that functions can be passed to other functions as arguments (first-class functions) and that anonymous functions can be created (lambda expressions). The grammar used in the experiment is shown in appendix A. Lambda expressions allow the grammar to define unique functions for the program which then act as modules. Lambda expressions also allow for currying, where a pre-existing function is called except some of the arguments to that function are fixed. Passing functions to other functions allows for a level of hierarchy in the structure of the program. An example of a program that uses such a grammar is now given.

```
#path generating function
circle_path(scalar, radius, midpoint, axis):
    When given a scalar value it will return a 3D coordinate on a circular path.
    The axis is the plane that remains constant for the circle coordinates

#connecting function
connect_3(starting_point, axis):
    Generates two additional points equidistant from original and then connects them.
    The axis is the plane that remains constant for the circle coordinates

#connecting function
drop_perpendicular(point):
    When given a point it will add a point where the z-plane is zero and connect the points

#higher order function
map(scalar_point_function, scalar_list):
    Uses a list of scalar values and a function to create a list of points on a path

#higher order function
function_map(function_list, point_list):
    Higher order function that applies each function in the function list to the point list
```

Fig. 6.5: A simplified function set from the higher order function grammar. It consists of a path generating function, two connecting functions and two higher-order functions.

A simplified example of the functional set of the grammar used in this experiment is given in Figure 6.5. The geometric objects are created by combining path functions and connecting functions. Path functions are given scalar values and return points on a path. Connecting functions are given points and then generate additional points and connect them together.

An example of a program is shown in Figure 6.6. The program uses the map function in conjunction with a scalar list to generate a list of points that follow a circular path 6.7(a). Two of the arguments to `circle_path()` have been fixed or “curried”, the axis is set to z-axis and the circle radius is set to 50. Function map the applies the functions `Connect3()`

```
function_map([lambda x: connect3(x ,'y'),
              lambda x: drop_perpendicular(x)],
              map(lambda t : circle_path(t, 50, (0,0,0), 'z'),
                  [0,1,2,3,4,5])
```

Fig. 6.6: An example program generated by the grammar. `function_map()` calls `drop_perpendicular()` and `connect3()` on the point list generated by `map()` and `circle_path()`.

and `drop_perpendicular()` to the set of points generated by `circle_path()`. `Connect3()` adds triangles to the points, as shown in Figure 6.7(b), and has the axis argument curried so that it is set to the y-axis for all of the triangles. Finally `drop_perpendicular()` add a vertical line as shown in Figure 6.7(c)

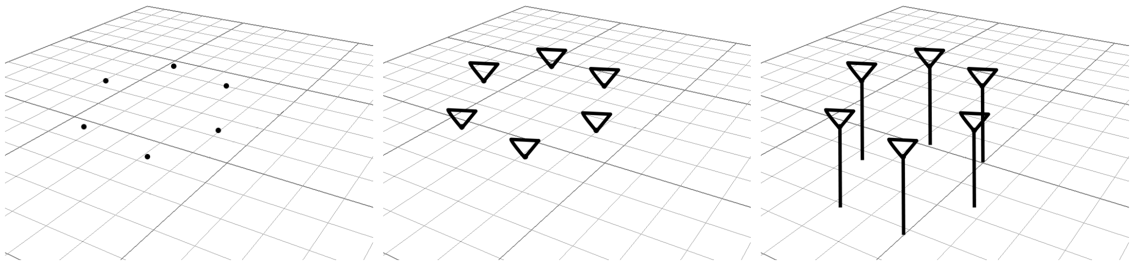

(a) `circle_path()`. (b) `connect3()`. (c) `drop_perpendicular()`.

Fig. 6.7: The output design generated by the program example.

Variations of the principles described in this example were used to create the designs upon which the locality measures were performed. The next section describes the settings used for the experiment.

6.4.2 Experimental Procedure

The experiment was implemented using GEVA version 1.2 [147]. We used a grammar which generates designs through the use of higher order functions described in Section 6.4.1. The designs are then rendered using Blender [180], an open source 3D modelling software. This allowed us to produce designs which could then be subjectively evaluated by users. The grammar used to generate the objects is shown in appendix A. It may then be used to compare similarity at the output stage of the evolved program. Instead of executing runs in the traditional sense, single mutation events were carried out (i.e.; a Hamming distance of one on the genotype) on randomly generated individuals and recorded the change in output. As the experiment required a large sample size of the mutation events where there was a phenotypic change, the experiment was allowed to run until 5000 non-neutral mutation events occurred.

Hypothesis

Given the distance results of a metric for two different mutation operators, μ_0 and μ_1, where μ_0 represents a sample of 5000 nodal mutations and μ_1 represents a sample of 5000 structural mutations, the following hypothesis is stated:

H_0 The different distributions of mutation events do not show any statistically significant difference, i.e. $\mu_0 = \mu_1$

H_1 There is a statistically significant difference between the distributions for a given metric, i.e. $\mu_0 \neq \mu_1$

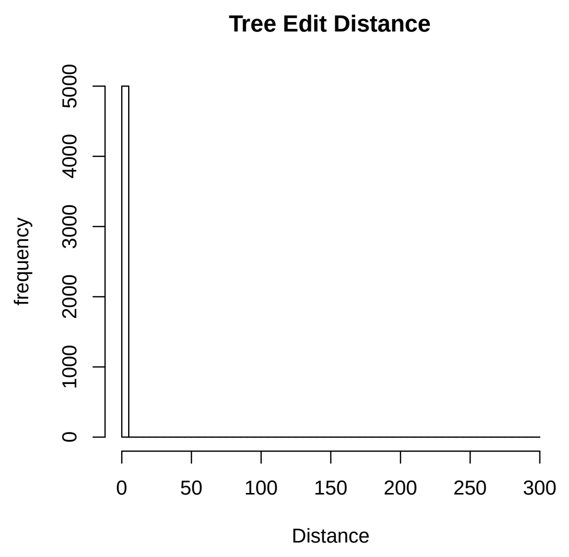
α The p-value for the Wilcoxon rank-sum is less than 0.05.

6.4.3 Results

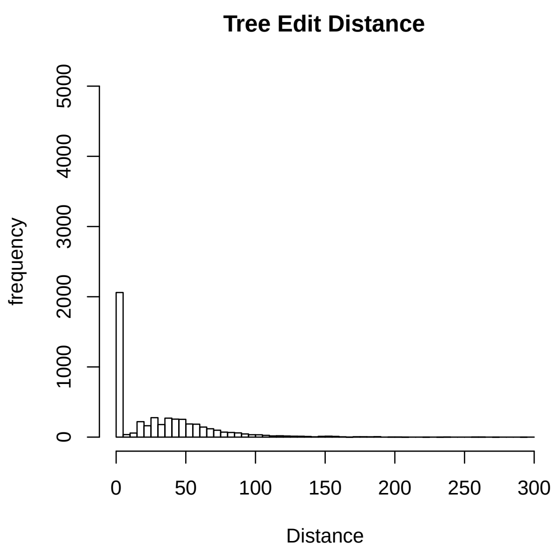
A Wilcoxon rank-sum was performed on the nodal and structural mutation results for each metric and they were shown to be significantly different at every stage of derivation. This result allows us to reject the null hypothesis. The distributions for the different distance metrics are shown in the in the histograms in Figures 6.8, 6.9, 6.10 and 6.11. The x axis in the graphed results represents the distance from the original individual. Therefore results close to the origin have high locality whereas results distant from the origin have low locality.

Tree Edit Distance

The result for tree edit distance in Figure 6.8 shows that nodal mutation only ever made a change size of distance one. This result is predicted as, by definition, nodal mutation should only replace one terminal node with another. Structural mutation had two distinct distributions. Just under 50% of the mutation events made a change of size one while the rest were distributed up to a distance range of 300. Although there was a large overlap between nodal and structural mutation, structural mutation had a second distinct distribution. The second distribution could be a result of the ripple effect that can change how a large amount of the codons are expressed.



(a) Nodal distribution.

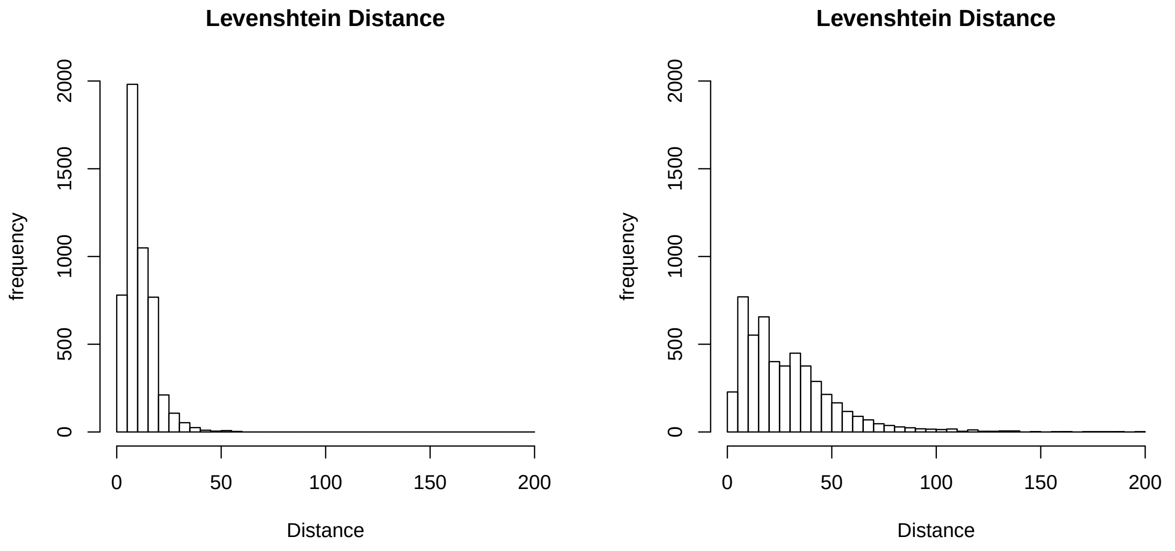


(b) Structural distribution.

Fig. 6.8: Tree edit distance.

Levenshtein Distance

The results for Levenshtein distance are shown in Figure 6.9. The nodal mutation distribution clusters close to the origin meaning that nodal events resulted in only a small change in distance. Structural mutation generated changes where the distance recorded by the metric was higher than nodal. Although structural mutation had less locality than nodal mutation, the distribution is skewed towards the origin. The result is that a portion of the distribution of structural events result in magnitudes of change similar to nodal mutation.



(a) Nodal distribution.

(b) Structural distribution.

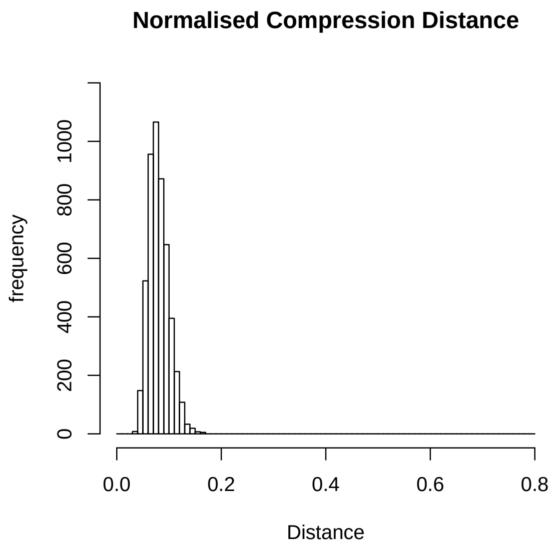
Fig. 6.9: Levenshtein distance.

Normalised Compression Distance

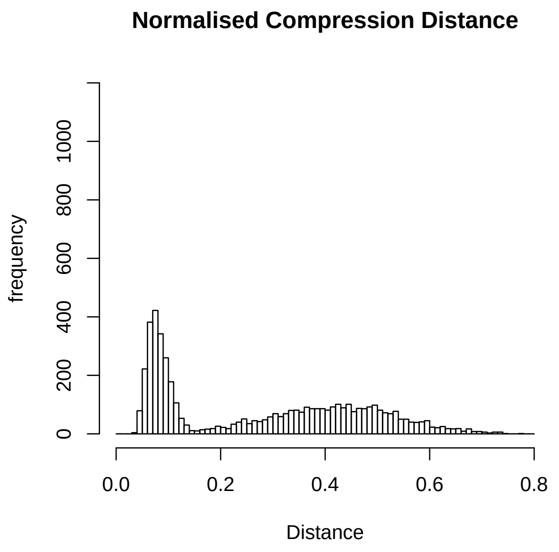
The results for normalised compression distance in Figure 6.10 show that nodal mutation events have higher locality than structural mutation events. The distribution of nodal change is still tightly grouped near the origin while structural mutation has two distinct distributions.

Euclidean Graph Distance

The results for Euclidean graph distance in Figure 6.11 show that nodal mutation has a higher locality than structural mutation. Even at the last phenotypic stage nodal mutation had a tighter distribution than structural mutation.

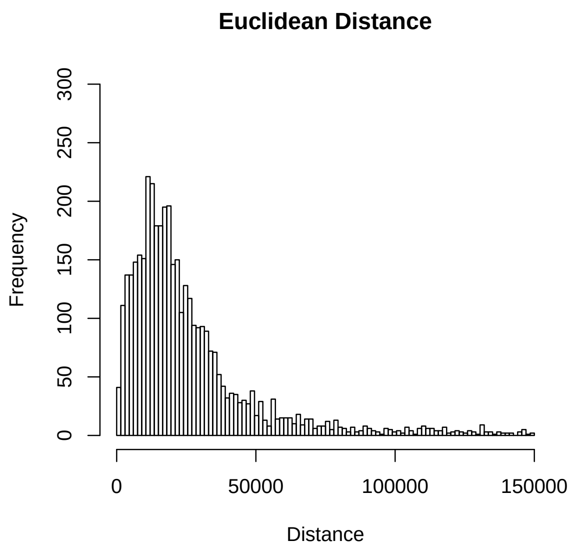


(a) Nodal distribution.

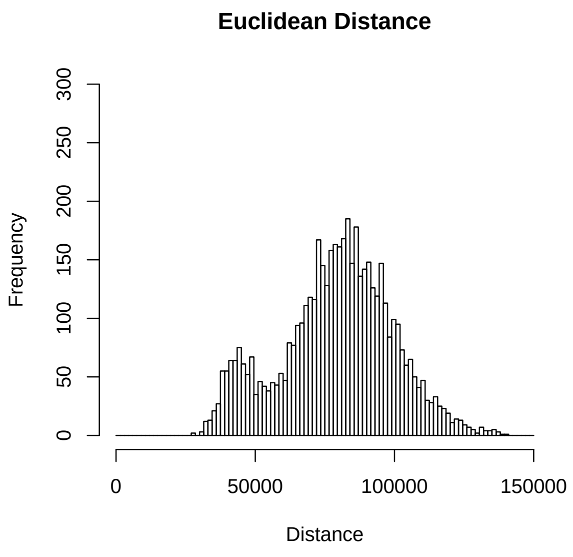


(b) Structural distribution.

Fig. 6.10: Normalised compression distance.



(a) Nodal distribution.



(b) Structural distribution.

Fig. 6.11: Euclidean graph distance.

6.4.4 Discussion

The results examined in Section 6.4.3 support the hypothesis that there is a significant difference between structural and nodal mutation at all stages of the derivation process. The metrics support the hypothesis that nodal mutation has high locality whereas structural mutation has low locality. As can be seen from these results the nodal mutation produced changes of smaller distance, i.e., higher locality, than structural mutation on both the derivation tree stage and the string stage. Structural mutation showed a non-normal distribution for both normalised compression distance and tree edit distance, as shown in Figures 6.10 and 6.11.

The occurrence of ripple mutation would explain a two peaked distribution. One distribution of small changes would be generated when structural mutation substituted one non-terminal for another thus leaving the subtree size the same. A broader distribution of large changes would occur when a substitution changed the size of that derivation subtree. The change in derivation tree size causes a "ripple" event which changes the following codons and so redefines the meaning of the remaining chromosome. A ripple event has previously been discussed in [144] and in Chapter 3.

The results highlight that there is a measurable difference in locality between nodal and structural mutation at different stages of the phenotypic output. If the operators are to be used for AUI then the difference in locality must also be perceivable by the user. While objective measures give an idea of the locality, the perceived similarity of mutation change is dependent on the subjective preferences of the user. The next section conducts a similarity survey of nodal and structural mutation events to see if user preference correlates with the metric results.

6.5 Locality Survey

This experiment examines if there is a perceivable difference in locality between nodal and structural mutation that the users can observe. The grammar used for this experiment is the same one that was described in Section 6.4.1. A random sample of 100 mutation events was taken where both a structural and nodal change occurred. These designs were then rendered in Blender and a survey was taken as to how closely the mutated designs resembled the original.

Survey participants are shown the original image and then presented with the same design after both a nodal mutation event and a structural mutation event (Figure 6.12). The user was then asked to select the image which most closely matches the target image. A randomly selected sample of the output is shown in Figures 6.13, 6.14, 6.15 and 6.16. No changes were made to the aspect or magnification of the images themselves and all were take from the same angle.

The participant evaluated 100 individuals and were given as much time as needed to complete the task. The order that the images were shown was randomised so as to avoid bias. There was also the option to pass on a selection if no clear preference was presented. Although a users interpretation of similarity is a somewhat subjective metric, the survey allowed us analyse the users perception of the effect of the different operators.

Hypothesis

Given two samples of nodal and structural mutations, μ_0 and μ_1, where the mutations are the paired results of mutating common target, the following hypothesis is stated:

H_0 The is no perceivable difference in locality between the two groups, $\mu_0 = \mu_1$

H_1 One group more closely matches the target, $\mu_0 \neq \mu_1$

α The significance level of the binomial test is 0.05.

6.5.1 Results

Tab. 6.1: Results for similarity survey.

Preference	Total	Percentage
Nodal	1313	82%
Structural	247	15.5%
Did not know	40	2.5%

The results from the survey showed a high degree of agreement that the nodal mutations produced an output more similar to the original than a structural mutation. A binomial

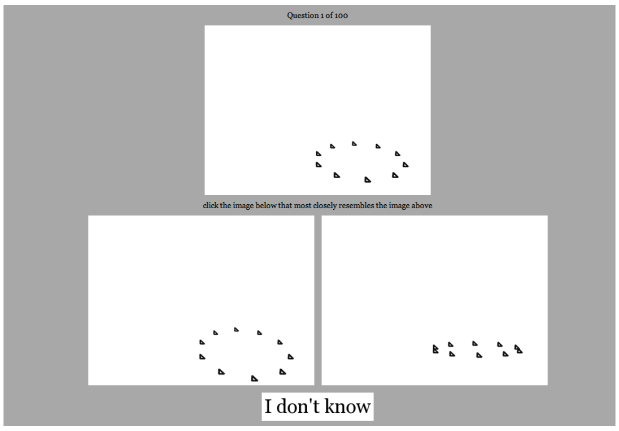
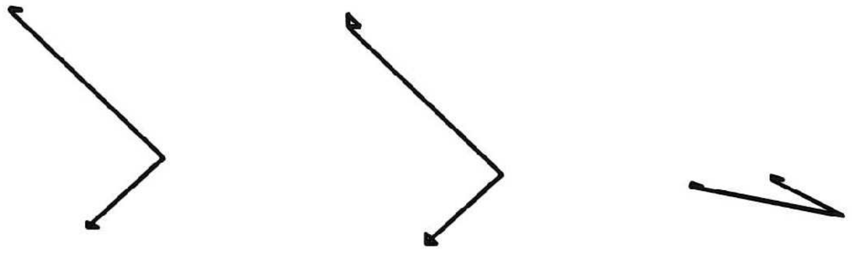
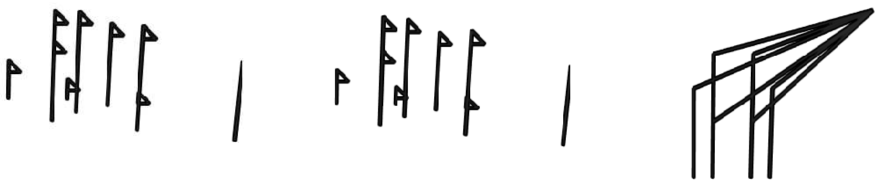

Question 1 of 100
click the image below that most closely resembles the image above
I don't know

Fig. 6.12: User screen for design survey.



(a) Original. (b) Nodal change. (c) Structural change.

Fig. 6.13: Example survey individual.



(a) Original. (b) Nodal change. (c) Structural change.

Fig. 6.14: Example survey individual.

test was performed on the results assuming that the distribution for the null hypothesis was 0.5, i.e., the users had a 50% chance of selecting nodal over structural mutation. The result for the binomial test showed that the results obtained were significant. The results for the survey are shown in Table 6.1. Despite the subjective nature of the survey, the result implies that nodal mutation is a high locality operator whereas structural mutation

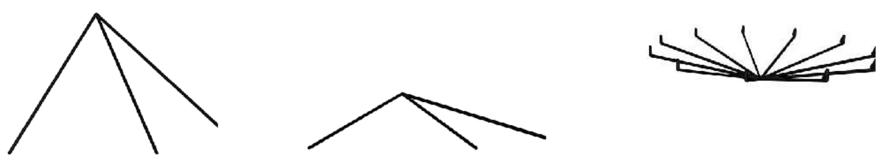

(a) Original. (b) Nodal change. (c) Structural change.

Fig. 6.15: Example survey individual.

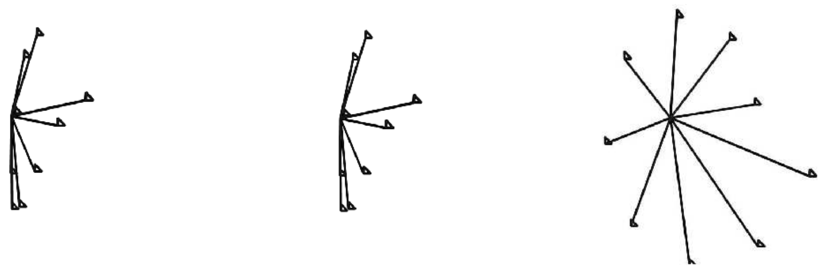

(a) Original. (b) Nodal change. (c) Structural change.

Fig. 6.16: Example survey individual.

is a low locality operator.

6.6 Conclusions

This study set out to analyse locality on the successive phenotypic output generated by GE. This chapter defined the different phenotypic stages and described the components within integer mutation that were being investigated. Experiments were carried out to show how the different mutation events compared in their locality. The results showed that it was possible to generate both high and low locality events during integer mutation that maintained their locality throughout the mapping process. The results will enable us to give users control over how much change they expect on a phenotypic level and increase the efficiency of their search. It also allows the user to search the design space exclusively with high locality operators and optimise designs that appeal to the user.

This chapter distinguished two different components of integer mutation in GE that each have different scales of change. By making this distinction it opens up the possibility of a dynamically changing operator. The operator would change the granularity of its search as it was carried out.

Chapter 7

User Interfaces for Interactive Evolutionary Design

7.1 Introduction

In traditional IEC there is a disconnect between the user and the algorithm. User evaluation through selection is only indirectly responsible for the output. The designer must feel directly involved in the design process [15]. Take the example of a user evaluating widget designs for an interactive evolutionary algorithm. Through selection they manage to find the perfect widget except that a single component is out of place. The user knows it is possible to move the component to the correct place and wants to alter the design. If evaluation is the only means of making this change then the user must select widgets where the component is in the correct place and hope the algorithm intuits their wishes correctly. Interrupt, intervene and resume (IIR) is necessary if evolutionary algorithms are to be used for design.

For IIR to work it must be possible to map changes to the phenotype back to the genotype. The complex mapping process of GE complicates this process. It is also possible the user could make changes that are not expressible in the grammar and so a reverse mapping would be impossible. The user is free to export the design to a suitable CAD format and make further changes but then the design cannot be evolved. An alternative approach is to enable the user to change the phenotype by altering the genotype. This chapter presents a novel interface that allows the user to directly manipulate the chromosome and instantly examine the resulting change.

In the preceding chapter it was demonstrated that nodal and structural mutation had a perceivable difference in the locality of their changes. Nodal mutation generated changes of high locality while structural mutation generated changes of low locality. This chapter examines if the operators can be used by the designer to explore the search space. Allowing the users to apply nodal and structural mutation would broaden interaction beyond evaluation and increase the amount of feedback and bias a user can apply to the search. Increased feedback will direct the algorithm to areas of the search space the designer finds more interesting. The experiments examine whether additional feedback from the user can be of benefit to the problem of interactive evolutionary design. The experiment finds that the interface between the user and the search space plays a vital role in this process.

Nodal and structural mutation have different degrees of locality. By giving the user the ability to selectively apply each operator the user can choose to move a small or a large distance from the current individual. Selecting which operator to apply allows the user to bias the search toward exploring a specific area of the search space, which has been shown by Whigham [194] to improve evolutionary search. User interaction with the search is increased by directly mutating individuals they find appealing. The approach taken in this chapter makes the assumption that when users find a design that appeals to them they want to see similar variations of that preferred design. The experiments described in this chapter evaluate different user interfaces to the evolutionary algorithm that allow the designer to direct the search using mutation and explore if these can aid an evolutionary design process.

This chapter is organised as follows. The initial interactivity experiment and user interface are described in Section 7.2. The results from this experiment led to a novel implementation of the interface and new series of experiments described in Section 7.3. Finally, the conclusions and future work are discussed in Section 7.4

7.2 Interactive Application of Nodal and Structural Mutation

This experiment primarily examines if user application of mutation operators can direct the evolutionary algorithm during search. The secondary goal of the experiment is to compare whether separating mutation events allows the user to navigate the search space more effectively by varying the locality of their search. The experiment compares user performance at directing search using integer mutation against a combination of nodal and structural mutation.

The subjective nature of aesthetics makes interactive evolutionary design a difficult area to quantify. Normally a user is allowed to explore the search space until they discover an appealing design. This approach is problematic as there is no way of distinguishing if the algorithm provided any benefit to the search or whether it just presented randomly generated designs to the user. In addition, there is also the problem that the interest a user has for a design is completely subjective and prevents any additional analysis of the results.

In an effort to generate measurable results, the experiment specifies a target design and the goal for the user is to match that target. The user results are then compared against a control group of randomly generated selections. The user is asked to match three targets using two different techniques. The targets are described in greater detail in Section 7.2.1. Although this is quite different to the design process, it is analogous to the user combining an interesting design with a design they had observed previously, i.e., finding a path from the current design to one that incorporates features of the previous design. Setting an objective makes the task quantifiable but it does not simplify the task. The user must change the correct codons before they are mapped to obtain the desired output. This is a non-trivial task given the complex mapping processes in GE. The experiment described in the next section examines if the user can direct search to match a target using such an approach.

7.2.1 Experimental Design

The experiment was run using Architype, an interactive design generation tool based on GE. Architype is based on PonyGE version 1.3 [80]. Rapid prototyping of interfaces is possible using Python's standard GUI package, tkinter. The grammar used in this experiment is the same bridge grammar that was described in Section 4.2.1 on page 62. The Architype interface was adapted for this experiment as shown in Figure 7.1. There is a target bridge on the right hand side and nine bridges to select from on the left hand side. When the user selects a bridge, their choice is saved in a green box in the top left frame and 8 mutated variations are made of it. The user can keep selecting bridges until they think they have matched the target or have reached the five minute time limit. Twenty four volunteers participated in this experiment and the experiment itself was approved by the Ethics Committee in UCD (Reference number: LS-E-11-05-Byrne-ONeill).

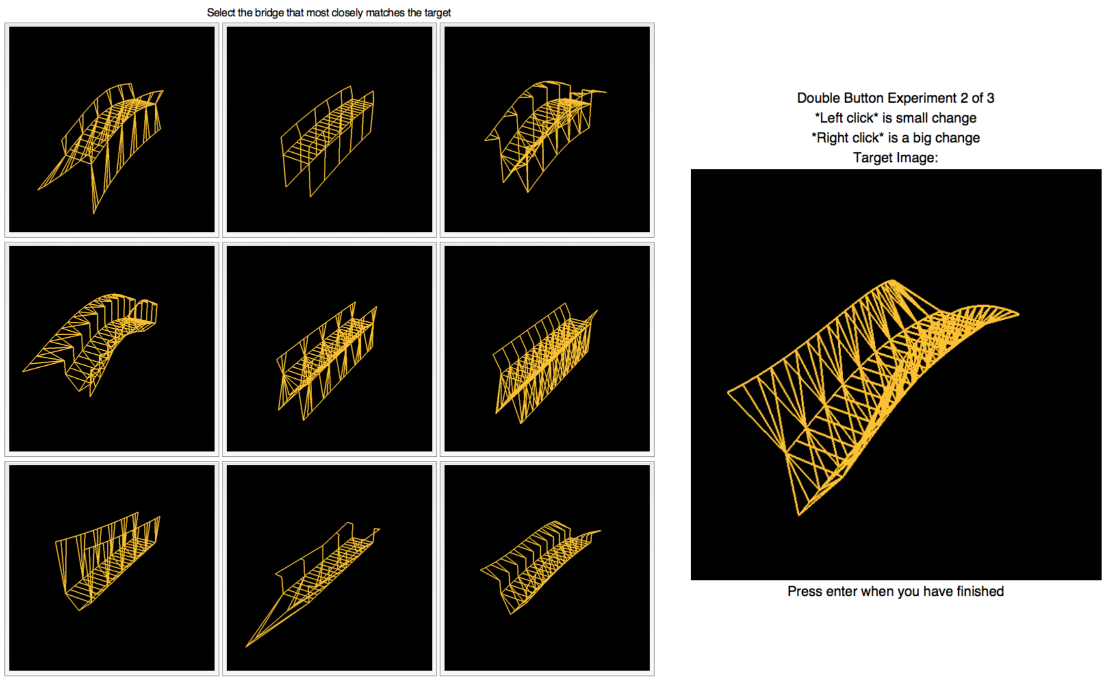

Select the bridge that most closely matches the target
Double Button Experiment 2 of 3
Left click is small change
Right click is a big change
Target Image:
Press enter when you have finished

Fig. 7.1: A screen-shot of the interactive GUI. The target is on the right and the results from the previous mutation choice are in the 9 frames on the left.

There is no crossover is used in this experiment. All variants are created exclusively by mutation events. The mutation operators do not work probabilistically, instead they select a codon from within the coding region of the genome and increment it by one. As a codon is only used when choosing the production of a rule, incrementing it by one means that it will always encode for a new production, thus ensuring a genomic change in the individual. It also means that the Hamming distance between mutation events is exactly one.

To avoid confusion it was decided to simplify the allowed user input. The user is limited to either the left mouse button (LMB) exclusively, or both the left and right mouse button (RMB). When the input is exclusively LMB, standard integer mutation is applied. When the user has a choice of both buttons, the LMB corresponds to a nodal mutation and the RMB corresponds to a structural mutation. The targets the user was required to match are shown in Figure 7.2. They were created by generating a random genotype which was then mapped using the same bridge grammar that the user interface was based upon. The idea of randomly generating the targets was to remove any human bias from the target selection process.

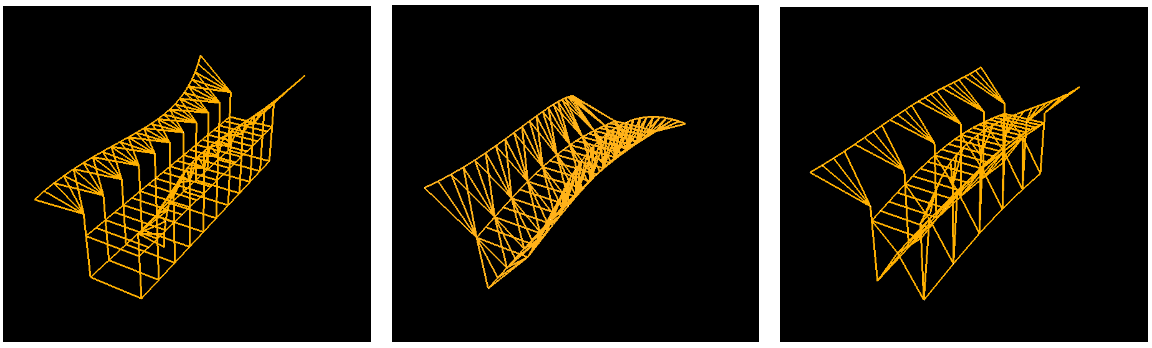

(a) Target 1. (b) Target 2. (c) Target 3.

Fig. 7.2: The targets that the users had to match.

The user was allowed two trial runs to familiarise themselves with the interface and the different effects of the mutation operators. Finally, after completion of the trial the user was asked to complete six experiments. The first three of the experiments used integer mutation to match three targets and the next three experiments used nodal and structural mutation to match the same three targets. A fixed random seed was used for each experiment so that all participants would be presented with the same initial designs. The time of each selection and the individual selected were recorded. The user had a time limit of five minutes to complete each task.

The user was presented with every selection they had made upon completion of the experiment and was asked to select the design that most closely matched the target. After completion of the experiment, the participant was asked to complete a short survey, supplied in appendix C.1. In order to see if the improvement over time was due to the choices of users, a sample of random mutation events was generated for comparison. The random sample was created by allowing a Mersenne twister random number generator make selections using the same interface that participants had used.

Hypothesis

Given a sample of randomly generated mutation events μ_0, and the results of user application of mutation operators, μ_1 and μ_2, where μ_1 allowed the user to apply integer mutation and where μ_2 allowed the user to apply nodal and structural mutation, the following hypothesis is stated.

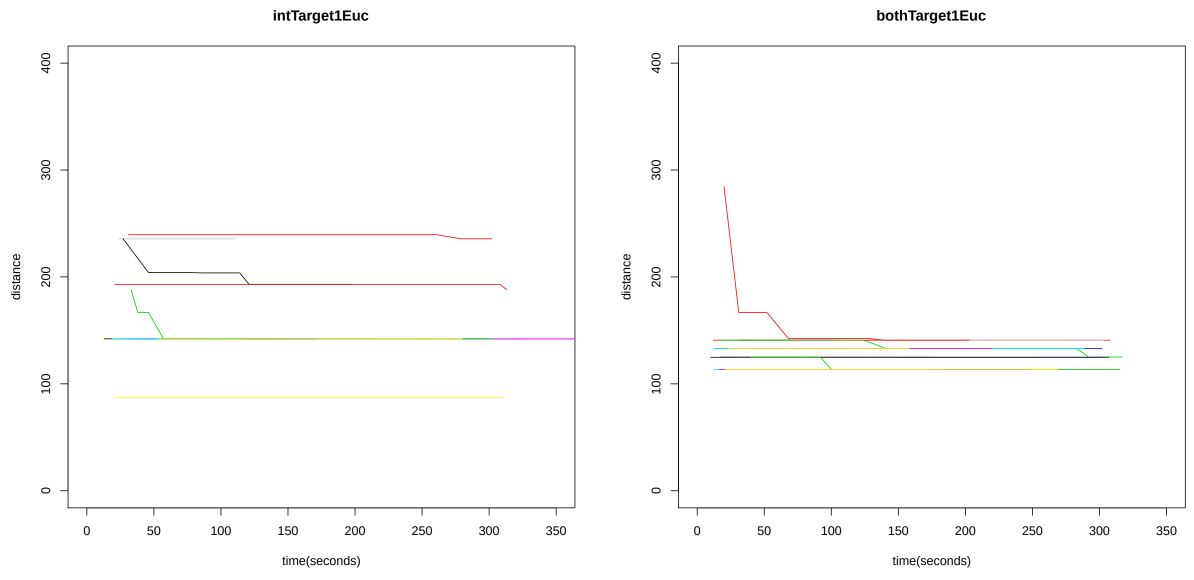
H_0 There will be no statistically significant difference between the random sample and either operator, $\mu_0 = \mu_1 = \mu_2$

H_1 There is a statistically significant difference in performance for a particular operator, $\mu_0 \neq \mu_1 \parallel \mu_0 \neq \mu_2 \parallel \mu_1 \neq \mu_2$

α The significance level of the Wilcoxon rank-sum is 0.05.

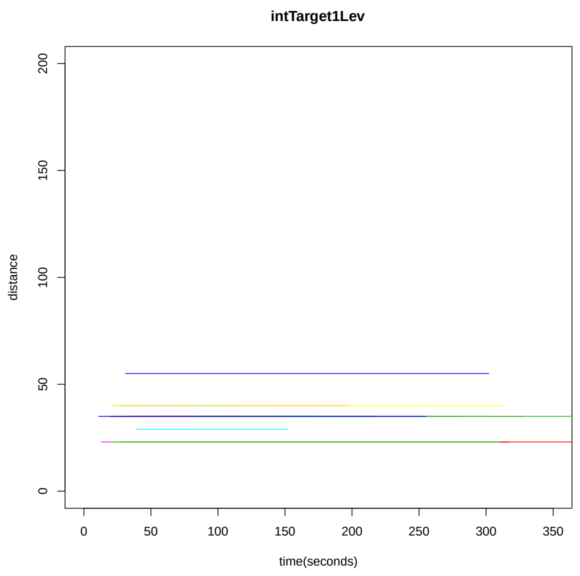
7.2.2 Results

The results for the distance metrics are shown in Figures 7.3 through 7.5. The Figures show the time taken by the user (in seconds) on the x-axis and the distance from the target on the y-axis. The smaller the value on the y-axis, the more closely the result matched the target. The graphs show the best result obtained from the user over the course of the run. As shown in the graphs, there is little or no improvement over time. This result was further confirmed by comparing the user input against the sample of random selections. A Wilcoxon rank-sum was performed and no significant difference was found between the user data and the randomly generated data. This results supports the null hypothesis. Only the results for the first target image are shown as the other target graphs show similar results. The results are discussed further in Section 7.2.3

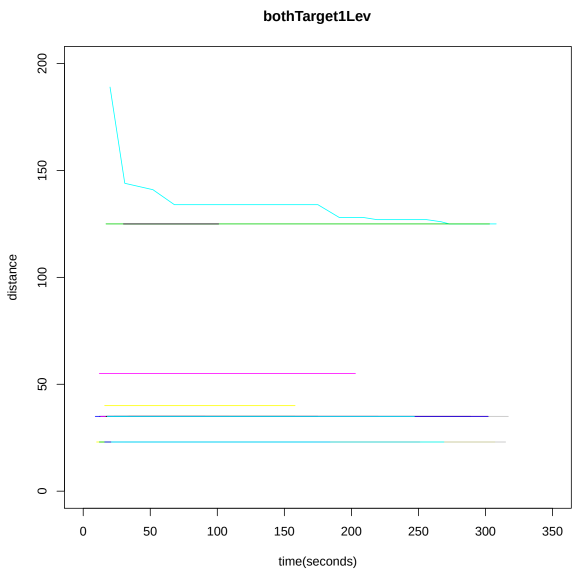


(a) Integer mutation. (b) Nodal / structural mutation.

Fig. 7.3: Euclidean distance for target 1, the x-axis is the time taken and the y-axis is the distance from the target. There is no significant improvement in distance during the course of the run.



(a) Integer mutation.



(b) Nodal / structural mutation.

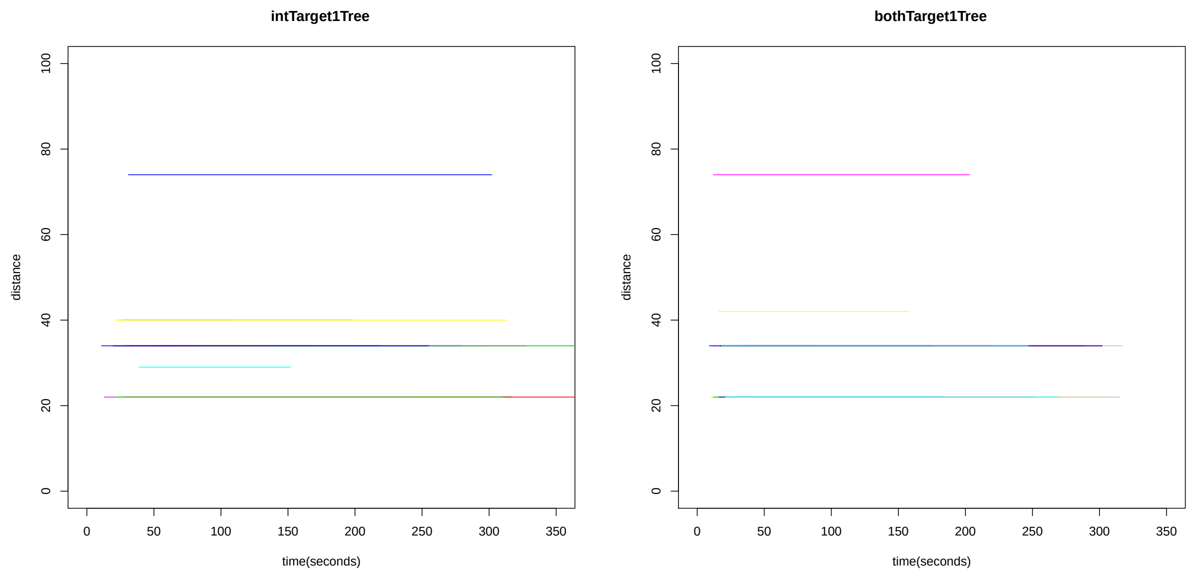
Fig. 7.4: Levenshtein distance for target 1, the x-axis is the time taken and the y-axis is the distance from the target. There is no significant improvement in distance during the course of the run.

7.2.3 Discussion

The results from Section 7.2.2 appear to support the null hypothesis, that the user is unable to direct search using interactive operators and selection. This is a highly contentious conclusion to draw as many years of interactive evolutionary computation studies have generated results that reach the opposite conclusion. If this is not the case then there are two possible causes for the results.

Choice of Metrics

How a human evaluates the similarity of two designs is a subjective measurement. The survey showed that users often based how similar they found bridges on individual features or parts of the design, such as the handrail or the curve of the walkway. While the metrics comparing bridges at earlier stages of the mapping process (tree edit and Levenshtein distance) would have great difficulty recording small changes on components, Euclidean



(a) Integer mutation. (b) Nodal / structural Mutation.

Fig. 7.5: Tree edit distance for target 1, the x-axis is the time taken and the y-axis is the distance from the target. There is no significant improvement in distance during the course of the run.

distance should have recorded some improvement. The results from Chapter 6 also showed a high level of correlation between the human perception of distance and the metrics. While the metrics are not perfect and human selection is subjective, some improvement should have been detectable.

Methodology

The experiment was constructed so as present the evolved population to the user and to facilitate precise logging of input. Pilot studies were completed to ensure the interface was usable and that certain concerns were addressed such as saving previous designs and allowing a single design to be repeatedly mutated. While these aspects of the interface are important they are essentially technical concerns. What was not addressed is the question of how to best facilitate the user's exploration of the search space.

A user made 17 selections on average during the five minutes they had to match a target. This equates to a Hamming distance of 17 from their original selection. An average of 48 codons were used to encode a design so a Hamming distance of 17 could allow for significant change to the individual. The lack of progress during search arose from the fact that each selection presented the user with 8 more images. As the participant is presented with only 136 designs (on average) during the course of a run, the result is that they only observe a small part of the search space. To assume that a significant improvement could be made in this short distance was optimistic.

The target bridges were created by randomly generating genotypes and using these to select individuals from the search space. Little thought was given to how “far” the target individuals were from the starting point. Nor was it known if the targets were reachable from the starting point. The approach taken in this experiment was intended to reduce human bias but it may have inadvertently made the task too difficult for the user.

The user did not know what change to expect between a nodal and a structural mutation, which meant that the users intention of“big” and “small” changes may not be evident in the generated individuals. What constitutes a small or a large change depends on the user’s personal preference. The nodal mutation events presented a unique problem to the user as the resulting mutations looked identical to the user. Some nodal changes fell below the threshold of a Just Noticeable Difference (JND). JND is a concept from cognitive psychology that was first described by Ernst Heinrich Weber [193]. JND is the smallest difference between two stimuli that is still capable of being perceived. The lack of what the user perceived as new variations also hindered them in completing the task.

Although the design selections were presented to the users at the end of the run, many were frustrated by the inability to go back to a good design. By forcing the user to follow a specific evolutionary path, the experiment severely limited the user and added to their frustration when they “lost” a good design. Whether being able follow different evolutionary paths would provide a benefit to search is a matter of conjecture but it degraded the user experience with the interface.

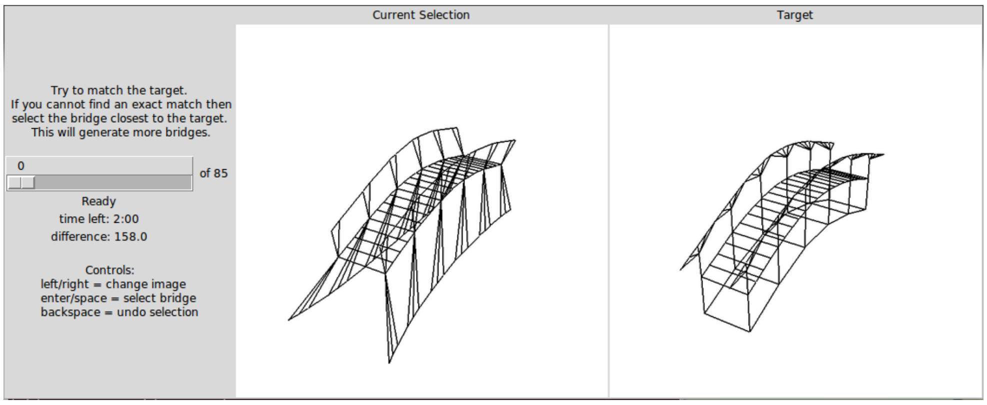
The grammar also complicated what was already a difficult task. Several participants complained of identical bridges being generated. Although most of these cases were a result of the JND described above, distance comparisons performed afterwards showed that some individuals were identical. Despite every mutation incrementing a used codon, sometimes the grammar generated anonymous functions that required codons for components that were not expressed in the output. An example of this would be codons used to generate a Bezier curve and only half the Bezier curve being used.

Due to the reasons discussed above a new approach was developed for allowing user interaction with the evolutionary algorithm. The new interface is discussed in the next section.

7.3 A Local Search Interface for Interactive Evolutionary Architectural Design

It became clear from the experiment in Section 7.2 that allowing the user to apply the mutation operators is not enough, the user has to be able to interact with the algorithm in a meaningful way and be able to process a much greater area of the search space. After analysing the results and user feedback gathered from the previous experiment, the interface was re-implemented in an attempt to allow the user to explore the search space more effectively. Whether the metrics adequately reflect similarity and locality is also examined.

The interface used in this experiment addresses the problems described in Section 7.2.3. Instead of the user choosing an operator, all possible mutations of Hamming distance one were applied to an individual. Each codon was mutated in turn, the result was rendered and then the codon was restored to its original value. The productions for each codon generated a collection of nodal and structural mutation events to choose from. This process is shown in Figure 7.7. A Euclidean graph distance comparison, described in Section 6.3.4 on page 116, was performed so that individuals identical to the original were removed from the population, thus reducing the search space presented to the user. By making no assumption about operator choice and instead presenting the user with every available option, it is now possible to examine how they navigate the search space based on their selections.



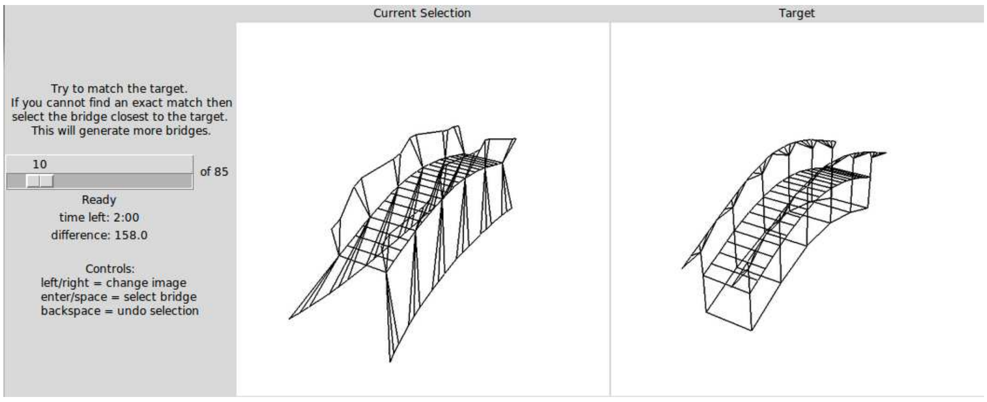
Try to match the target.
If you cannot find an exact match then select the bridge closest to the target.
This will generate more bridges.

0 of 85

Ready
time left: 2:00
difference: 158.0

Controls:
left/right = change image
enter/space = select bridge
backspace = undo selection

(a) The target is on the right and the starting individual is in the middle frame.



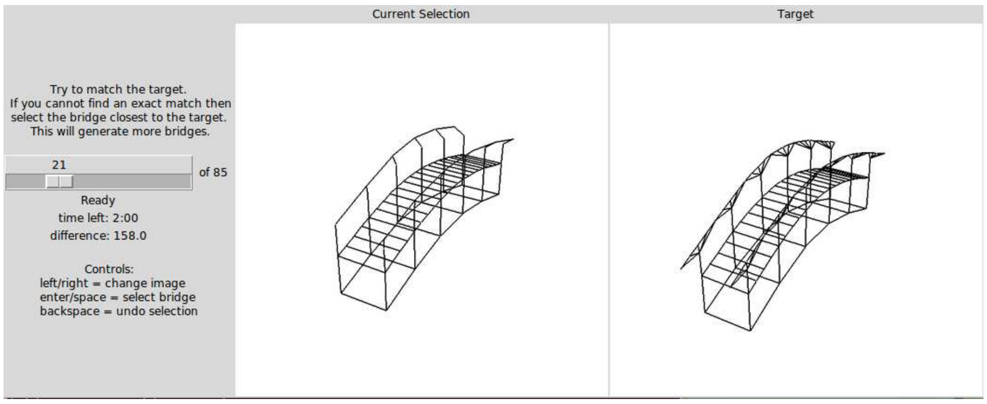
Try to match the target.
If you cannot find an exact match then select the bridge closest to the target.
This will generate more bridges.

10 of 85

Ready
time left: 2:00
difference: 158.0

Controls:
left/right = change image
enter/space = select bridge
backspace = undo selection

(b) A nodal mutation of the starting individual. The handrail is altered slightly by the mutation event.



Try to match the target.
If you cannot find an exact match then select the bridge closest to the target.
This will generate more bridges.

21 of 85

Ready
time left: 2:00
difference: 158.0

Controls:
left/right = change image
enter/space = select bridge
backspace = undo selection

(c) A structural mutation of the starting individual. The supporting struts and the handrail design are altered by a single mutation event.

Fig. 7.6: New user interface. The target is on the right and the current individual is in the middle frame. The user instructions and a slider that shows the current position in the population is on the left.

Tab. 7.1: Problems addressed by new interface.

Problem	solution
Magnitude of change unknown	Show all changes in advance
Limited user evaluations	Local search is animated for quicker evaluation
No measure of target difficulty	Each target increases in Hamming distance
No alternative paths allowed	“undo” button added
Difference below JND	Mutations animated to highlight changes
Identical Individuals	Removed by comparing Euclidean graph distance

Presenting an entire population of mutation events to the user simultaneously is not feasible. The interface instead uses a single window for exploring the population. The interface is shown in Figure 7.6. The current user selection is on the left and the target is the image on the right. The leftmost panel states the instructions, user controls, time remaining and the distance from the target (stated as the difference). The user scans through the mutation events using the left and right arrow keys and the selects the mutation change they want and that now becomes the basis for generating the next population.

The refresh rate for the window is ten frames per second. As the frame rate is below the level of “persistence of vision”, where the afterimage remains on the retina, the user is capable of perceiving the bridges distinctly. Codon changes were made sequentially so a codon's productions are grouped in their presentation to the user. Overlaying groups of changes in the same window allowed the user to pick up smaller JND changes by viewing them in rapid succession.

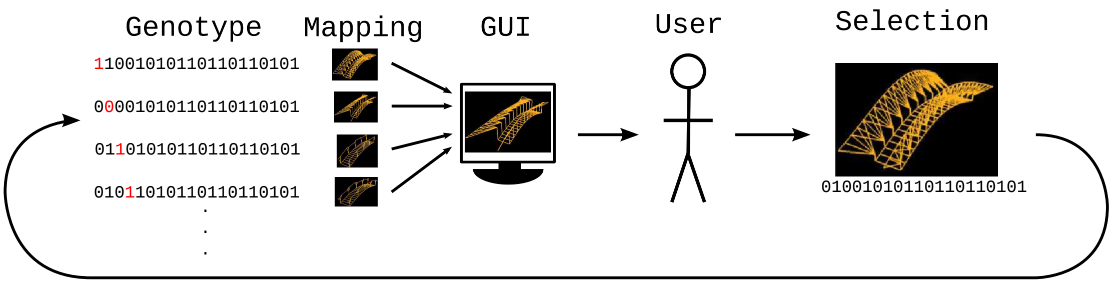


Fig. 7.7: Generating Hamming distance 1 individuals for user selection. The value of each codon is altered and the result is saved. The codon is then restored to the original value before moving on to the next codon.

7.3.1 Experimental Setup

The subjective nature of aesthetics makes evolutionary design search a difficult area to quantify. To generate measurable results a target design is specified and the user must then attempt to match it. The experiment asked the user to match ten different targets. The random seed was fixed so that interface always started from the same individual. The targets were mutated variants of the starting individual and they increased in difficulty as the Hamming distance was greater for each successive target.

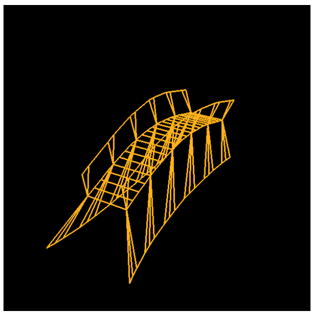
The targets are shown in the order they were presented to the user in Figure 7.8. The Hamming distance, the number of nodal and structural mutations and the Euclidean graph distance from the starting graph for each target is shown in Table 7.2. In an effort to generate a range of target difficulties, each target had a Hamming distance greater than or equal to the target that preceded it. Although this is not an absolute measure of difficulty it does mean that more input is required from the user to obtain a perfect match.

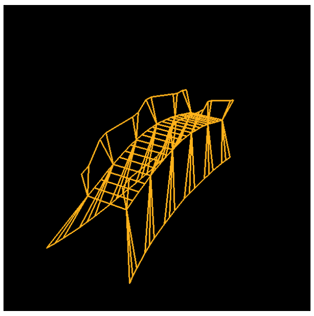
Each participant had two minutes in which to try to match the target. They were free to make as many selections within this time frame and they could undo selections if they wished. At the end of target exercise the user was asked a short survey, supplied in appendix C.2. Twenty five volunteers participated in this experiment and the experiment itself was approved by the Ethics Committee in UCD (reference number: LS-E-11-129-Byrne-ONeill).

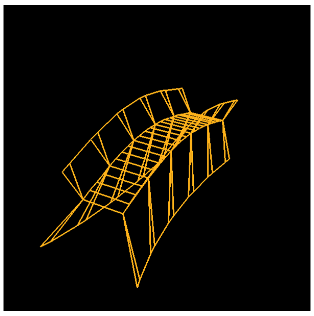
A sample of random trials were generated to examine if the users were capable of using the interface to match the target. The same interface was used except selections were chosen randomly by a Mersenne twister random number generator. Twenty five random samples were generated for each of the targets. The distribution of the random selections for each target matched the user's click average and standard deviation, as shown in Table 7.2.

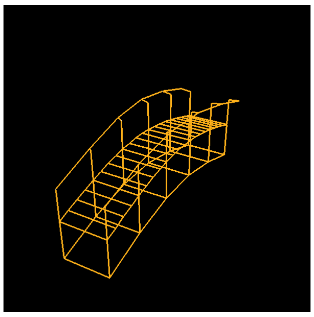
Hypothesis

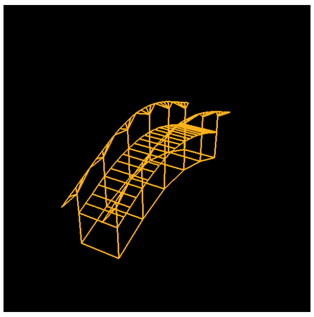
Given the distance results of a sample of randomly selected individuals from a given target, μ_0, and the distance results of the user selected individuals from a given target, μ_1, the

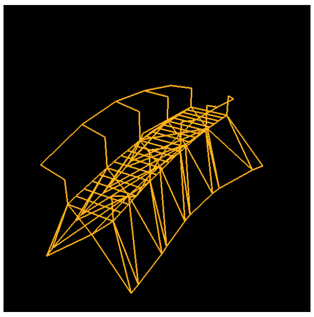

(a) Original.

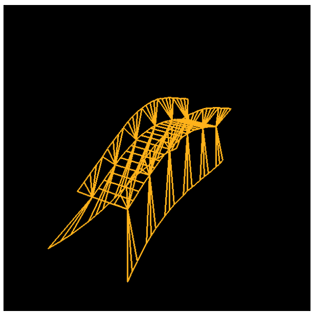

(b) Target 1.

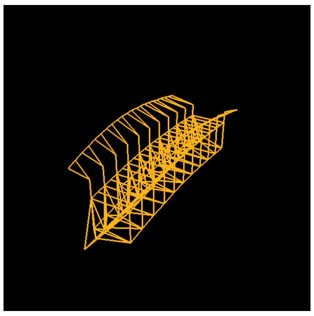

(c) Target 2.

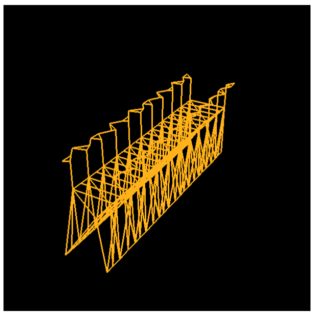

(d) Target 3.

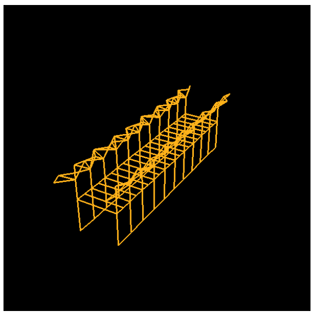

(e) Target 4.


(f) Target 5.


(g) Target 6.


(h) Target 7.


(i) Target 8.


(j) Target 9.

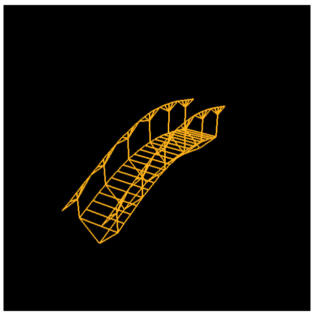

(k) Target 10.

Fig. 7.8: The targets that the user had to match, the original individual that the user started with is in the top left corner.

Tab. 7.2: The distance and change types for each target.

Target	Nodal: Structural	Hamming: Euclidean	User Clicks	User Evaluations	User Matched	Random Matched
Target 1	1:0	1:10	1.0 ± 0.0	22.4 ± 34.9	100%	4.0 %
Target 2	2:1	3:92	3.8 ± 2.1	190.3 ± 80.6	64.6%	11.8%
Target 3	2:2	3:184	2.7 ± 1.3	202.4 ± 61.6	64.9%	12.1%
Target 4	4:1	5:158	4.1 ± 2.2	245.2 ± 126.4	74.1%	20.3%
Target 5	5:1	6:214	3.3 ± 1.1	325.6 ± 130.2	36.6%	31.8%
Target 6	7:0	7:188	3.5 ± 1.3	196.0 ± 61.4	58.9%	31.3%
Target 7	6:2	8:465	3.5 ± 1.2	262.3 ± 83.2	57.6%	15.9%
Target 8	7:1	8:457	4.0 ± 1.9	364.7 ± 131.0	37.5%	27.0%
Target 9	7:2	9:273	4.8 ± 2.0	306.4 ± 86.1	28.2%	36.4%
Target 10	7:4	11:117	4.9 ± 2.0	282.4 ± 103.7	29.7%	42.5%

following hypothesis is stated:

H_0 There will be no statistically significant difference between the random sample and the user generated results, $\mu_0 = \mu_1$

H_1 There is a statistically significant difference in distance from the target for the user generated results, $\mu_0 \neq \mu_1$

α The significance level of the Wilcoxon rank-sum is 0.05.

7.3.2 Results

The number of user selections (user clicks) and the number of images presented to the user (user evaluations) are shown in Table 7.2. In the previous experiment, the users selected 17 individuals and evaluated 136 designs on average in a five minute time period, the equivalent of 27 evaluations per minute. The user click and user evaluation results show that the user made fewer selections with the new interface than in the previous experiment but they evaluated many more designs within a two minute time period. The highest number of evaluations was for target five where the user was presented on average with 325 images for evaluation, the equivalent 162 evaluations per minute.

The percentage of user mutations (User Matched) and random sampling mutations (Random Matched) that matched the target codon changes are also shown in Table 7.2. With some exceptions, the percentage of matched codon mutations decreases as the Hamming distance increases. The opposite is true for random sampling: as more codons are changed there is an increased likelihood that a random mutation would match that of the target.

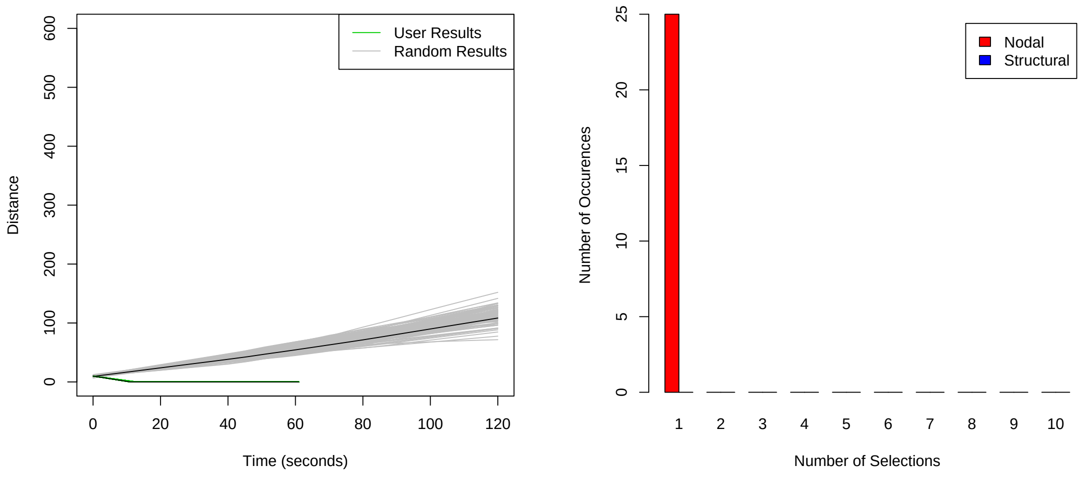
A Wilcoxon rank-sum test compared the final selections for the user and the random sampling. There was a statistically significant difference from random with exceptions of target 5 and target 10. The result rejects the null hypothesis for 8 of the 10 targets. In these cases the results show a distinct improvement over time for the user generated results while the random samples either show no reduction of distance from the target or an increase in distance from the target. The results show that users could successfully use the interface to direct search.

Scatter plots were generated to examine if there was an improvement over time regarding Euclidean graph distance from the target. Each user selection generated a data point that recorded the time, distance from the target and the mutation type. A locally weighted scatter plot smoothing (LOESS) was performed on the results to plot a smooth curve of the average values. The set of data points was then bootstrapped [54]. Bootstrapping is a resampling technique that generates an estimation of the distribution during the course of a run. The LOESS curves for each of the samples were plotted. The graphs on the left of Figures 7.9 to Figures 7.18 compares the user distribution (green) with the random sampling (grey).

The type of mutation for each selection was recorded with intention of examining if mutation operators of different locality were used at different points during the target matching process. The user was not informed of whether they were making a nodal or structural mutation. They made their selection based on the output of the mutation event. Bar charts showing the number of nodal mutations (red) compared to the number of structural mutation (blue) are shown in the graphs on the right of Figures 7.9 to 7.18. The x-axis is the number of selections made while the y-axis shows cumulative frequency for a particular type of mutation.

Target 1

The first target was used to introduce the user to the interface. As the target was a Hamming distance of one from the original, every participant successfully matched the target with one selection. The result shows that if a desired individual is in the immediate proximity of the current individual, the user is capable of finding it.



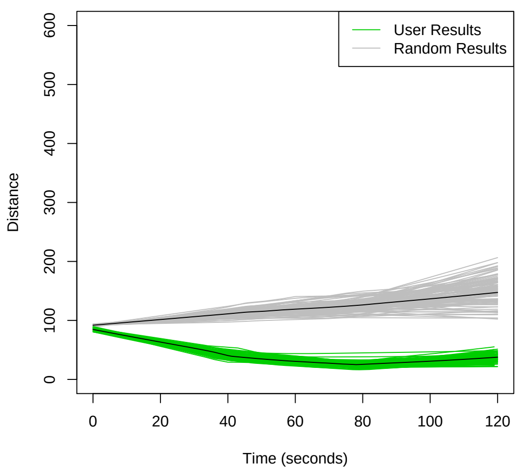
(a) LOESS and bootstrapping results.

(b) Structural / nodal selections.

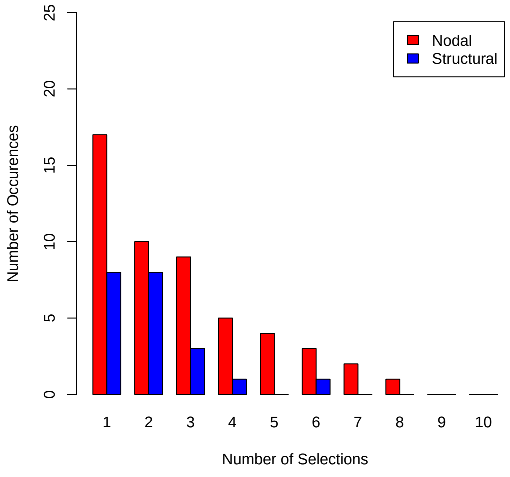
Fig. 7.9: Results for target 1.

Target 2

The second target consisted of two nodal mutations and one structural mutation and was one of the three instances where there was more nodal selections initially than structural selections. This meant the users chose a high locality mutation event instead of a low locality event. As the Euclidean distance from caused by the mutation events was only 92, meaning that none of the mutation events had a greatly changed the individual. If the locality of both nodal and structural changes was comparable then it would explain why the participants went for a nodal change initially rather than a structural change.



(a) LOESS and bootstrapping results.



(b) Structural / nodal selections.

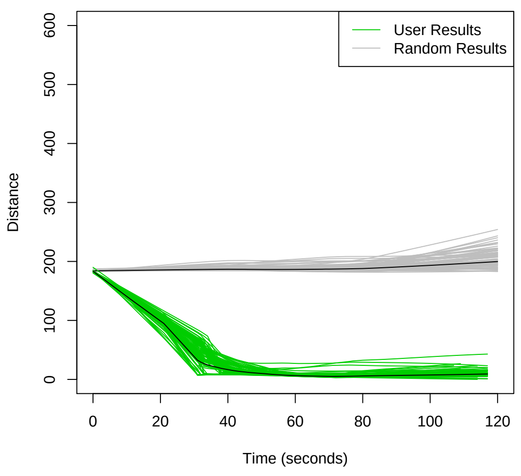
Fig. 7.10: Results for target 2.

Target 3

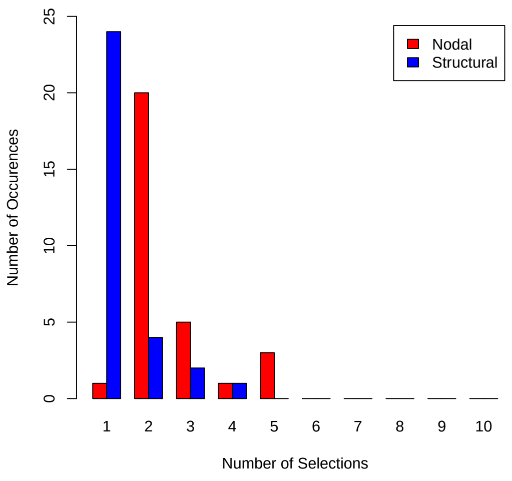
The third target consisted of two nodal and two structural mutations. The participants overwhelmingly chose a structural mutation initially and then refined their selection with higher locality nodal mutations. This is shown in the LOESS results as a steep decrease in the distance followed by a plateau where there was little additional improvement.

Target 4

The fourth target consists of four nodal mutations and one structural mutation. Participants overwhelmingly started with structural mutation and then applied nodal mutations. Despite the user's preference for a low locality event for their first selection, the fitness initially got worse according to the Euclidean distance. Additional selections made by the participants increased the fitness and there was significant improvement in distance by the end of the task.

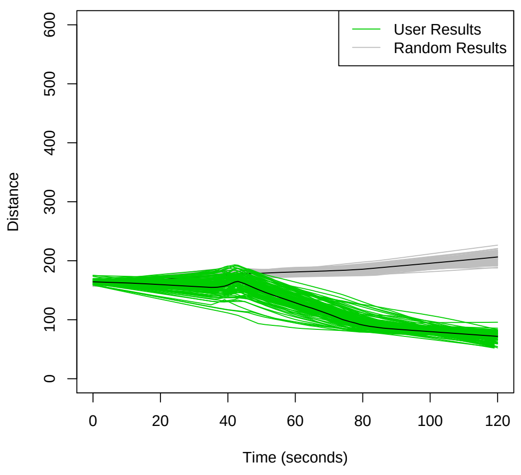


(a) LOESS and bootstrapping results.

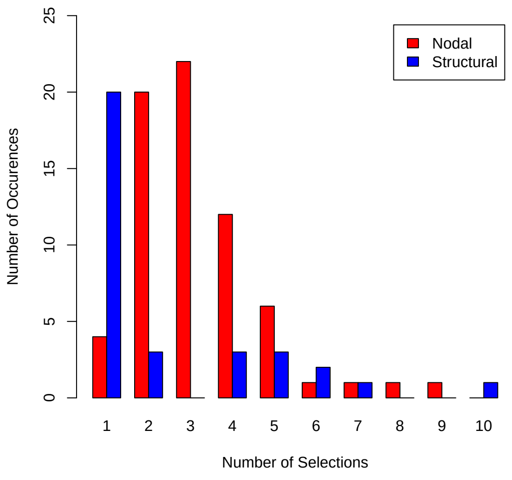


(b) Structural / nodal selections.

Fig. 7.11: Results for target 3.



(a) LOESS and bootstrapping results.

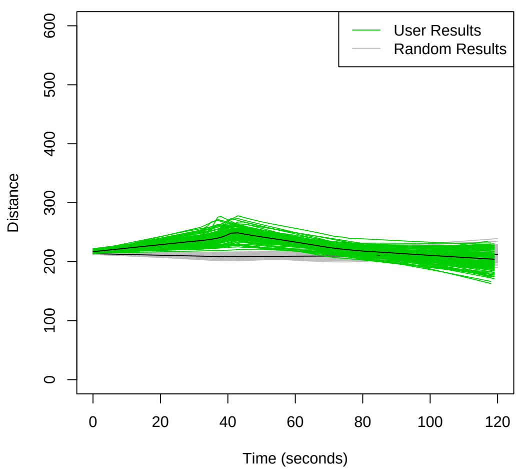


(b) Structural / nodal selections.

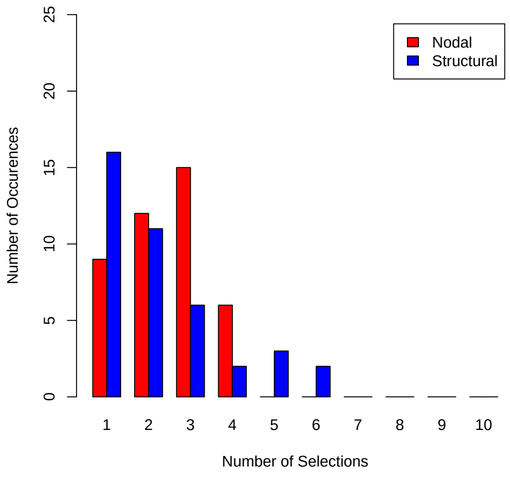
Fig. 7.12: Results for target 4.

Target 5

The fifth target consists of five nodal mutations and one structural mutation. Although more structural mutations were applied in initially, the proportion of nodal and structural mutations remains consistent. The participants mutated the same codons used to create the target only 36% of the time, resulting in a poor score. There is no improvement made by the participants for this experiment and their results are indistinguishable from random selection.



(a) LOESS and bootstrapping results.



(b) Structural / nodal selections.

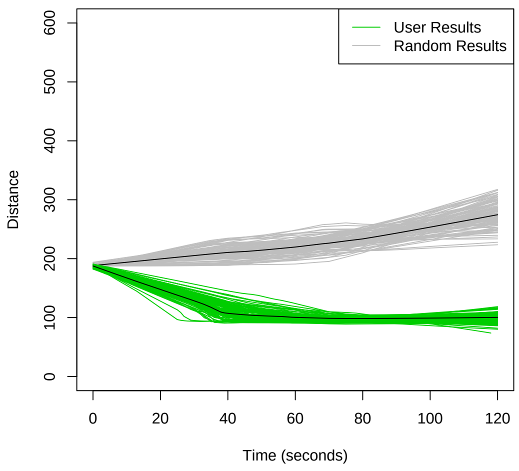
Fig. 7.13: Results for target 5.

Target 6

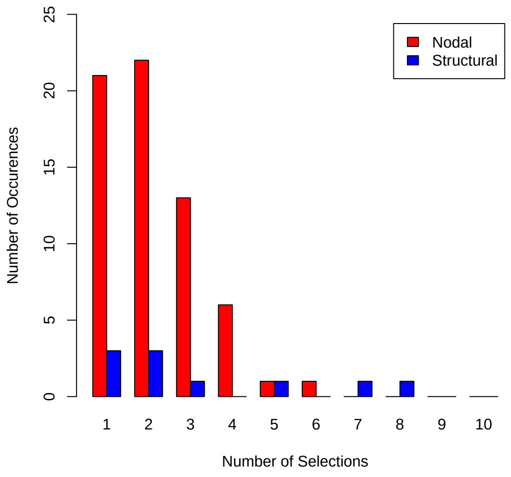
The sixth target consists of seven nodal mutations and no structural mutations. This is the second of the three instances where there were more nodal mutations made initially than structural mutations. As all the changes that generated the target were nodal changes and 58.9% of the user selections matched the mutated codons, it shows that the users were capable of following an evolutionary path similar to the one that created the target.

Target 7

The seventh target consists of six nodal mutations and two structural mutations. There is a large drop off in distance initially as users all selected the same structural codon for mutation. The low locality of changing that structural codon is evident in the Euclidean



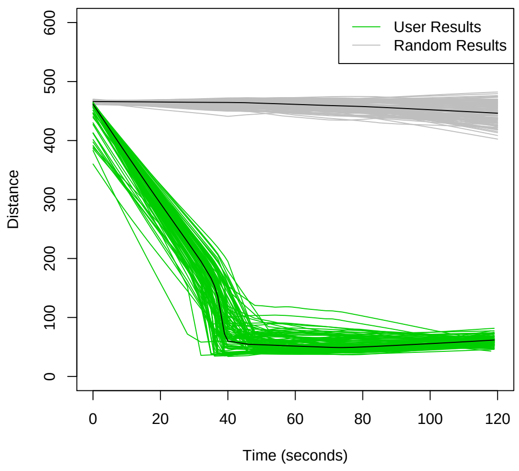
(a) LOESS and bootstrapping results.



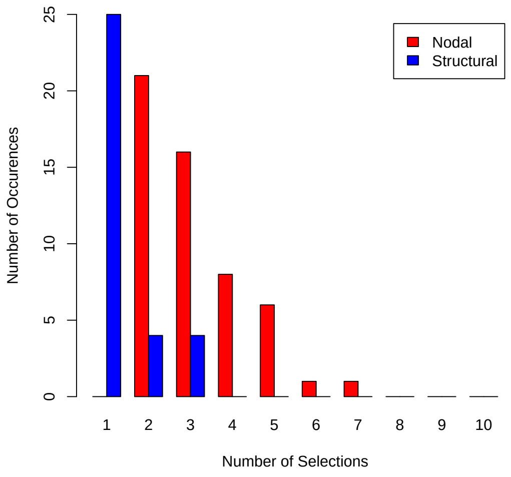
(b) Structural / nodal selections.

Fig. 7.14: Results for target 6.

distance. Once the distance had been reduced the users used nodal mutations of higher locality vary the design, resulting in smaller fitness changes.



(a) LOESS and bootstrapping results.

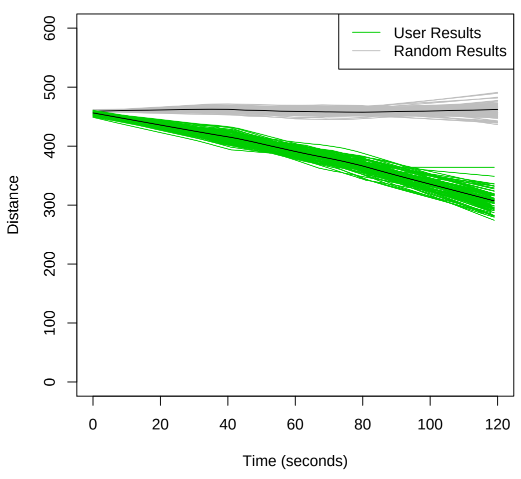


(b) Structural / nodal selections.

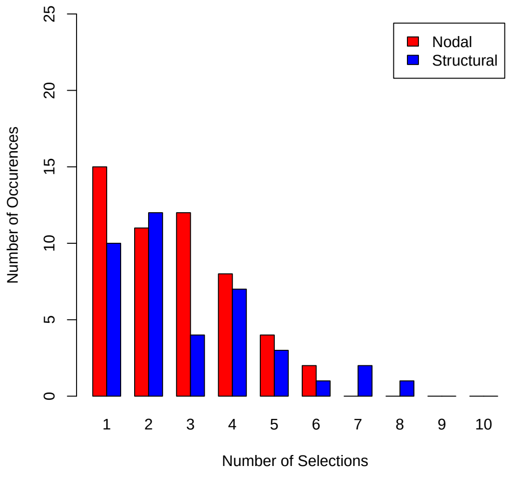
Fig. 7.15: Results for target 7.

Target 8

The eighth target consists of seven nodal mutations and one structural mutation. It is the third instance where there were more nodal mutations initially than structural mutations. Overall there was a lower percentage of matched codons. This means that the users were not mutating the same codons that changed to generate the target but there was still a significant decrease in distance. The result suggests the users could match the phenotypic output but by following a different genotypic path.



(a) LOESS and bootstrapping results.

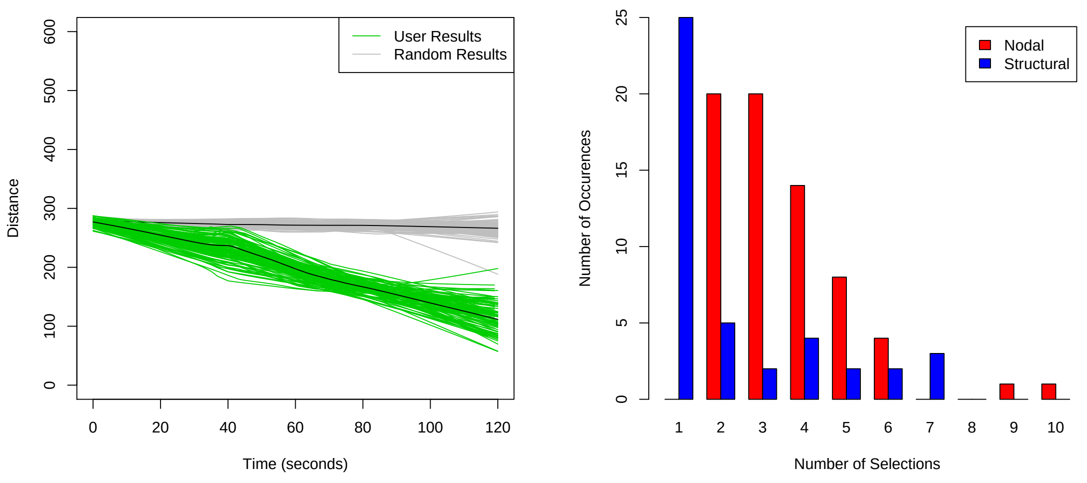


(b) Structural / nodal selections.

Fig. 7.16: Results for target 8.

Target 9

The ninth target consists of seven nodal mutations and two structural mutations. The users all applied an initial structural mutation but it does not have the same level of improvement as seen in Figure 7.15. Overall the users matched less codons than the randomly generated sample (28% and 36% respectively). Despite this there still was still a significant reduction in distance. The variance of the results is much larger than on the other targets, suggesting that while some users were able to match the target, there were several participants who had much less success. The results for the eighth and ninth targets support the hypothesis that different genotypic paths could lead to the same phenotypic output.



(a) LOESS and bootstrapping results. (b) Structural / nodal selections.

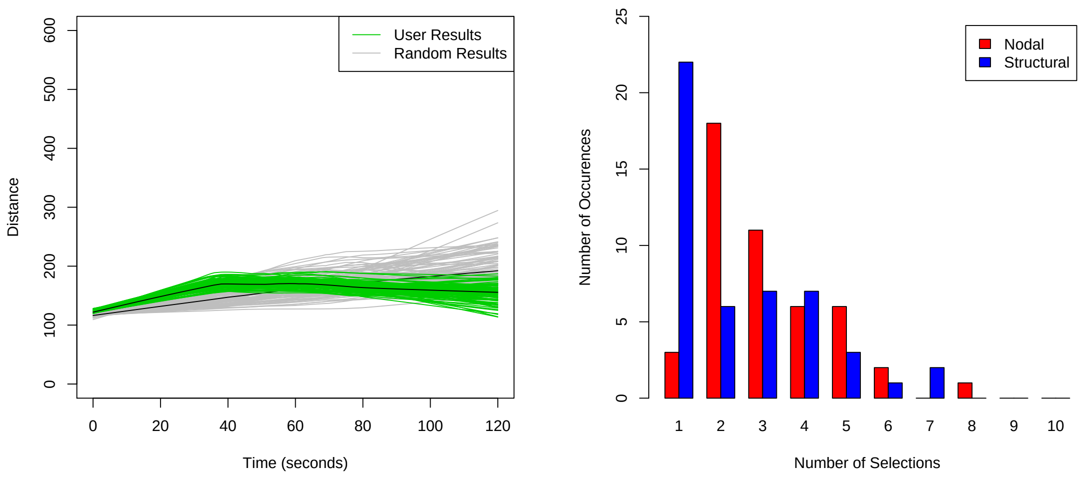
Fig. 7.17: Results for target 9.

Target 10

The tenth target consists of seven nodal mutations and four structural mutations. This was the largest Hamming distance and over one third of the expressed codons were mutated. As the user only made four selections on average it is unlikely the user would be able to match the target. There was no statistically significant difference between the user results and the randomly generated sample.

7.3.3 Discussion

The results discussed in Section 7.3.2 show that the users were able to match a target using the new interface, with the exception of target five and target ten. In the case of target five, the users matched few of the mutated codons and so followed a different mutation path that ended up in a local optima. In the case of target ten, a large percentage of the



(a) LOESS and bootstrapping results. (b) Structural / nodal selections

Fig. 7.18: Results for target 10.

expressed codons were mutated and so it presented too difficult a challenge to match all the changes within the allotted time.

This evidence supports the hypothesis that the user could use such an interface to mutate a design to incorporate features that they had previously seen in other designs. The results obtained regarding the participants use of nodal and structural mutation show that users started by predominantly applying a structural mutation and then moving to nodal mutation. The result means that the participants commenced their search by making large phenotypic changes and then fine tuning that solution with high locality mutation events. There were three exceptions to this including target six which consisted of nodal mutations exclusively.

A surprising result is that there was a definite improvement in Euclidean graph distance for targets 8 and 9 while the overall percentage of codons matched were low (37.5% and 28.2% respectively). The result shows that it is possible to get close to matching a target without following the exact same mutation path. What this also means is that the user is following a different genotypic path to arrive at the same phenotypic output.

7.4 Summary

In this chapter novel interfaces were presented that enabled the user to perform a local search on an individual. The initial experiments failed to show that the user was capable of directing search. A second interface was developed that allowed the user perform a local search on a specific individual. The experiments showed that the user was able to use the second interface to match a target by directly manipulating the genotypic representation. This result supports the hypothesis that the new interface can facilitate user directed local search toward a desired individual. Examining the user generated results showed that they moved from low locality operators to high locality operators to both explore and exploit the search space. As the user changes are made to the genotypic representation, the new individual can be reintroduced into the population and evolutionary algorithm can continue. The intention of this work is to combine the interface with existing evolutionary design tools to explore the additional benefits of combining AUI with evaluation for evolutionary architectural design.

Chapter 8

Conclusions & Future Work

Having examined different methodologies for architectural design exploration based on grammatical evolution, the conclusions are now discussed. A summary of the thesis and the contributions it has made are given in Section 8.1. The limitations of this thesis are highlighted in Section 8.2. Finally, avenues for future work are given in Section 8.3.

8.1 Thesis Summary

The goal of this thesis was to examine the use of grammatical evolution as a method for exploring the search space of architectural design. Combining the open-ended representation of GE with the structural constraints of architectural design allowed for the open ended exploration of different architectural designs. In order to incorporate the aesthetic preferences of the designer, methodologies for increasing user feedback both directly and indirectly were examined. The following questions were asked in order to examine if this goal could be accomplished:

1. *Can objective fitness functions be developed for evolving architectural designs?*
2. *Can search operators be developed that allow the user to directly interact with the design at a genotypic level?*

3. *How does the design of the interface effect the user's interaction with the evolutionary algorithm?*

4. *Can a context free grammar based representation be used to generate architectural designs?*

Initially, experiments were carried out to address the question on non-interactive fitness functions in Chapter 4 to examine if structural analysis could provide a means of evolving structures for design exploration. It was found that while GE was capable of optimising structures using multiple structural objectives, using physical constraints to restrict the search space was not a suitable approach to design exploration as it reduced the number of interesting or appealing designs presented to the user. It was then decided to explore mutation as a possible means of user directed search.

To answer the question about developing user directed operators, an analysis of mutation in GE was then conducted in Chapter 5. The results showed that there were two component behaviours in standard integer mutation. The effects were classified into nodal and structural events. Nodal events altered the leaves of a derivation tree while structural events altered the internal structure of the derivation tree. Both operators showed a markedly different impact on fitness on the benchmark problems.

The locality of changes made to the genotype were examined at different phenotypic levels in Chapter 6. Metrics and measures were chosen for comparing the different derivation levels. Nodal and structural mutation again showed to have a different magnitude of effect at all phenotypic levels. The mapping process also magnified the effect during the later stages of derivation.

An interface was then tested in Chapter 7 to examine if the user was capable of directing search by directly applying operators. Initial experiments did not show any significant difference. The results and user feedback was analysed and the interface was redesigned. The results for the new interface showed that the users could direct search by applying mutation exclusively. Analysis of the mutation events showed that users initially made large changes using structural mutation and then fine-tuned solutions with nodal mutation events.

Shape generating grammars were used throughout this thesis showing that they are a suitable representation for generating complex architectural shapes. The ability of grammars to represent designs as high level abstractions and to facilitate module definition meant that it is capable of generating complex structures and shapes. The grammars were used to generate abstract yet connected shapes, ornate bridges and electricity pylons. The designs output by these grammars were shown to be comparable to human generated designs [150].

8.1.1 Contributions

Two lines of research were carried out in this thesis, an analysis of the GE algorithm and the application of GE to architectural design. The contributions resulting from these investigations are described separately below.

8.1.2 Analysis of GE

Performed an analysis of the behaviour of mutation in GE. A detailed examination of mutation events consisting of a single change on the genotypic level was conducted in Chapter 5. The results showed that changes identical at the genotypic level caused changes of different magnitude at the derivation tree level. This finding was explored further by examining the effect of the component behaviours of integer mutation on fitness for a series of benchmark problems.

Ripple mutation. It was shown that mutation events at particular points on the chromosome can alter the meaning of following codons. The result is that different magnitudes of change can occur for a single mutation event. The experiments in Chapter 6 found evidence of two distinct distributions of change on the phenotype caused by unit changes on the genotype. The occurrence of ripple mutation during structural mutation would generate a similar result.

Examined the locality of mutation events at different phenotypic levels. The study of how mutation effects the derivation tree and fitness was extended in Chapter 6 to explore the effect of mutation on locality at different levels of the derivation process. The magnitude of change was analysed at the different phenotypic levels: the derivation tree level, the parse tree level, the string level and the final output phenotype. The results showed that there was a clear difference in the magnitude of change at the different derivation levels

Introduced nodal and structural mutation operators. The investigations of integer mutation in Chapter 5 led to the development of two component operators, nodal and structural mutation. The operators were defined mathematically and shown to explore the search space differently.

Developed a Euclidean graph distance measure for comparing similarity in graphs. A Euclidean distance measure was devised in order to have an objective measure for locality at the output stage of a design. It was shown to correlate with the user's subjective notion of similarity.

8.1.3 Application of GE to Architectural Design

Literature review. Chapter 2 presented a review of the literature regarding computer generated architecture, evolutionary design exploration, interactive evolutionary computation and design representations. An overview of the GE algorithm was also presented and the operations carried out by the algorithm explained. In addition to these contributions the conclusions that can be drawn from the experimental results is now discussed.

Evolved architectural designs using structural analysis in GE. GE had previously been used to generate designs [143, 150] but this had been done using subjective user preference. Chapter 4 introduced a form of structural analysis capable of being used to evolve structures. The results showed that the fitness function was capable of reducing the overall stress on the structure.

Developed a multi-objective fitness function for GE. Although multi-objective fitness functions have been used by genetic algorithms for optimising multiple constraints, this is the first time an NSGA2 based multi-objective fitness has been combined with Grammatical Evolution. The results showed that multiple constraints could be evolved simultaneously and that there was a statistically significant improvement for each constraint during the course of the run.

Developed an interface to allow the user to perform a local search of an individual during IEC. The investigations carried out in Chapter 7 explored the impact of the interface design. It was shown that the GUI preformed a critical role in allowing the users to explore the search space. The experiments showed that animating individual mutations successively presented more of the search space to the user and allowed them to perceive smaller changes than were noticeable when images were presented side by side.

Conducted an analysis of user interaction with an evolutionary algorithm. The mutation events selected by the users were categorised and examined during the interactivity experiments. The results showed that when the users were limited to mutation events with a Hamming distance of one, they would choose low locality mutation events initially and then refine the result with nodal mutation events.

Development of shape generating grammars for shape evolution. Grammatical evolution was shown to be a suitable methodology for producing complex shapes that exhibited hierarchy, regularity and modularity. It was shown that attributes could be added to the design to allow for structural analysis of the designs.

Conducted surveys of user preference for structurally evolved designs. Participants were asked to evaluate designs that fulfilled the design constraints versus those that did not based on their personal preference. The results in Chapter 4 showed that the users had a distinct preference for designs that did not conform with the constraints.

8.2 Limitations of Thesis

The primary focus of this thesis was to increase the amount of direct and indirect interaction available to the designer for evolutionary design exploration. As stated in Chapter 1, not all possible research avenues could be addressed.

When experimenting with any evolutionary algorithm, the choice of settings can vary the result. In EC, a number of direct and indirect choices must be considered, e.g. the choice of operators, the rates of crossover, mutation, selection and replacement. This thesis has in no way made an exhaustive search of these settings. The grammars used in our design experiments were bespoke grammars for particular designs, there been no exhaustive search of possible grammar combinations. To allow further generalisation the number of problems examined and settings used can always be increased and extended.

The basis of our interactive experiments was asking the participant to match a target image. This may seem like an artificial task for a design tool as exploratory search does not have a pre-determined outcome. Nevertheless searching for a design is a common task, especially if the user is attempting to incorporate components of a design that they had previously observed. As such it is a suitable methodology for search that allows for quantifiable results as opposed to basing experiments on subjective preference for generated designs.

A survey was conducted that showed users preferred designs that did not meet the structural design constraints of stress reduction and material usage (Chapter 4). This result does not mean that structural analysis combined with a multi-objective fitness function was incapable of evolving elegant designs. If the user initially directed the evolutionary algorithm so that the population consisted of design appealing to the user, the fitness function could be used to optimise these designs and increase functionality. This is addressed in more detail in future work.

The shape generating grammars used in this thesis were all based on context free grammars. There are a number of areas of active research for different approaches to grammars such as tree-adjoining grammars [132] and meta-grammars [81] that were not examined in this thesis.

8.3 Opportunities for Future Research

The work described in this thesis has shown that grammatical evolution is a suitable methodology for evolving architectural designs and structures. The grammar based system allows the designer to partake in the evolutionary process and directly manipulate the designs for evolution. Building upon this research, there are numerous avenues for future work.

Although structural analysis did not provide a suitable mechanism for design exploration it could be applied once a suitable design is found. The user could mark preferred designs for non-interactive optimisation. If the user wanted the topology and shaping of the design to remain similar to their selection a low mutation rate could be used to alter aspects of the design incrementally. Once the overall topology and shape has been chosen the algorithm could optimise the sizes of the structural components. This would maintain the overall aesthetic look of a design while making it more efficient.

The use of a multi-objective fitness function allows for objective aesthetic measures to be considered by the fitness function. The addition of aesthetic consideration to the fitness function could also generate more visually appealing designs.

User interaction with the pareto front is another interesting area worthy of further research. As all designs on the pareto front are considered pareto-equivalent, the user could introduce their subjective preference and direct the search by selecting the most interesting designs from the pareto front. The animated interface discussed in Chapter 8 could provide a methodology for accelerating the evaluation process.

The behaviour of integer mutation should be categorised in greater detail. The studies carried out in this thesis demonstrated that there are two clear components of integer mutation. While this research provided a detailed analysis of mutation events, it is by no means the last word on mutation in GE. It provides a starting point for further analysis of the mutation operator. The behaviour of mutation could be decomposed further and additional classification for mutation events could be possible.

The objective nature of the target matching experiments constrained the participant while they interacted with the new interface. A more unrestricted usage of the interface could be conducted in the future. The more qualitative aspects of the interface would be examined such as the benefit the user thinks the interface provides, as well as how much they feel it aids their search. The interface could be used for a real world design challenge with a control group using subjective selection exclusively.

The Euclidean graph premetric could become a fully fledged metric with some additional development. This would provide a useful tool for analysing the phenotype at the same output stage as is presented to the user. This would be an important development in locality analysis and would compliment the current derivation tree and sting level analysis.

The operator experiments in this thesis focused solely on integer mutation in GE. Other GE operators could be examined using the same techniques. Specifically crossover could be examined in detail using the locality operators developed for Chapter 6.

One of the most intriguing areas for further research is the automatic generation of parametric design systems using evolutionary algorithms. The most difficult part of using parametric design tools is setting up the initial model. If a repository of existing parametric systems could be encoded as a grammar then GE would be capable of evolving these component models. The designer could then use GE to generate a design similar to the model they intended without having to manually implement the design. Future work in this area would involve creating a grammar that generated models for commercial parametric design software.

List of Definitions.

Appendix A

Grammars

A.1 Grammar for Generating Bridges

```
<program> ::= <init><consts><adf><handrail><walkway><parabola>
             <walkwayEdges><handrailEdges><offset_copy>
<init> ::= g = graph.graph(); handrail_node_ids=[]; walkway_node_ids=[]
<const> ::= strut_multiple = <strut_multiple>
            npts = 20; pointA = [0, 0, 0]; pointB = [30, 0, 0]
<handrail> ::= def handrail(t):{return <handrail_curve>(t)}
<walkway> ::= def walkway(t):{retval=<walkway_curve>(t){}return(retval)}
<parabola> ::= def f(t):{return 1.0 - pow(2 * t - 1.0, 2)}
<adf> ::= def make_strut(i, t):
              n=<nbranches>
              xyz=walkway(t)
              xyz[2]=p*xyz[2]+(1-p)*handrail(t)[2]
              id=g.add_unique_node(xyz,'post')
              g.add_edge(i, id){}for j in range(n):
                  xyz=handrail(t+(j-(n-1)/2.0)/float(npts))
                  id2=g.add_unique_node(xyz,'handrail')
                  handrail_node_ids.append(id2){}g.add_edge(id, id2)
<handrailEdges>::=handrail_node_ids.sort()
                    for i in range(len(handrail_node_ids) - 1):
                        g.add_edge(handrail_node_ids[i], handrail_node_ids[i+1])
<walkwayEdges>::=walkway_node_ids.sort()
                     for i in range(len(walkway_node_ids) - 1):
                        g.add_edge(walkway_node_ids[i], walkway_node_ids[i+1])}
<offset_copy> ::= g.copy_and_offset_with_mirror((0.0, 5.5+0.1*<sx>,0), True)
<walkway> ::= for i in range(npts+1):
                  t=i/float(npts)
                  id = g.add_unique_node(walkway(t),'walkway')
                  walkway_node_ids.append(id)
# Functions which return a point, given a scalar.
<scalar_point_func> ::= <add_scalar_point_funcs> | <bezier> | <xyzcos>
                      | <xyzcos> | <expcurve>
# Given a scalar t, return a point on a cubic bezier curve.
<bezier> ::= lambda t: bezier_form(t, (<bpt>, <bpt>, <bpt>, <bpt>))
<expcurve> ::= lambda t: [<xexp>, 0.0, 0.0] | lambda t: [0.0, <xexp>, 0.0]
             | lambda t: [0.0, 0.0, <xexp>]
<xexp> ::= <sx> * exp(1.0 + 2 * <sx> * t)
#specify the height of the handrail
<zoffset> ::= lambda t: [0, 0, 4]
<walkway_curve> ::= (lambda t: pt_plus_pt((<interpolateAtoB>)(t),
                    (<z_half_cycle_sin>)(t)))
<interpolateAtoB> ::= lambda t: interpolate(t, (pointA, pointB))
<handrail_curve> ::= (<add_scalar_point_func_and_offset>)
<walkway_plus_zoffset> ::= lambda t: pt_plus_pt(walkway(t), (<zoffset>)(t))
# Given a scalar t, return a point on a diagonal between two points.
<diagonal> ::= lambda t: interpolate(t, (<pt>, <pt>))
```

```
<add_scalar_points> ::= lambda t: pt_plus_pt((<scalar_point_func>)(t),
                        (<scalar_point_func>)(t))
<add_scalar_point_offset> ::= lambda t: pt_plus_pt((<scalar_point_func>)(t),
                              (<walkway_plus_zoffset>)(t))
# allow any number (even not a multiple of 2pi) of revolutions
<xyzcos> ::= lambda t: [<xcos>, 0.0, 0.0] | lambda t: [0.0, <xcos>, 0.0]
             | lambda t: [0.0, 0.0, <xcos>]
# use 1.0 + cos() to keep it positive, avoid negative z values
<xcos> ::= <sx> * (1.0 + cos(<ndoublerevs> * 4 * pi * t))
<z_half_cycle_sin> ::= lambda t: [0.0, 0.0, 4 * <sx> * sin(pi * t)]
<pt> ::= [<sx>, <sx>, <sx>]
#<dimension> indicates x, y or z
<dimension> ::= 0 | 1 | 2
```

A.2 Higher Order Function Grammar for Generating Geometric Shapes

```
<scene> ::= <shapes>
<shapes> ::= <list_of_shapes>

# Given a list of functions and a list of points, return a list of shapes
<list_of_shapes> ::= map(<list_of_point_to_shape_funcs>, <list_of_points>)

# Only one method of creating a list of points.
<list_of_points> ::= map(<scalar_point_func>, make_scalar_list(<n>))

# Functions which return a point, given a scalar.
<scalar_point_func> ::= <unitcircle> | <ellipse> | <diagonal> | <bezier> |
                        <spiral> | <add_scalar_point_funcs> | <sinusoid>

# Given a scalar t, return a point on the spiral around a bezier curve.
# The radius, initial phase, and number of revolutions can be specified.
<spiral> ::= spiral(t, <radius>, <phase>, <revs>, <scalar_point_func>)

# Given a scalar t, return a point on a diagonal between two points.
<diagonal> ::= interpolate(t, (<pt>, <pt>))

# Given a scalar t, return a point on a circle with given radius and centre
# in the plane indicated by <dimension>.
<unitcircle> ::=  circlePath(t, <radius>, <pt>, <dimension>)
<ellipse> ::=  ellipsePath(t,<radius>,<radius>,<pt>,<dimension>)
<add_scalar_point_funcs> ::= pt_plus_pt((<scalar_point_func>)(t),
                                   (<scalar_point_func>)(t))

<sinusoid> ::= (0.0, 0.0, <xcos>) | (0.0,<xcos>, 0.0)
             | (<xcos>, 0.0, 0.0)

# use 1.0 + cos() to keep it positive, avoid negative z values
# use 4pi * t so that 2 full revolutions, for t in [0, 1]
<xcos> ::= <x> * (1.0 + cos(<ndoublerevs> * 4 * pi * t))

# Given a scalar t, return a point on a cubic bezier curve.
<bezier> ::= bezier_form(t, (<pt>, <pt>, <pt>, <pt>))

# Functions which return a shape, given a point.
<point_shape_func> ::= [connect(<pt>, x)]
                      | [dropPerpendicular(x, 2)]
                      | connect3(x, <dimension>)
<list_of_point_to_shape_funcs> ::= [<point_shape_funcs>]
<point_shape_funcs> ::= <point_shape_func>
                      | <point_shape_funcs>,<point_shape_func>
<ndoublerevs> ::= 1 | 2 | 3 | 4
# points are represented as tuples
<pt> ::= (<x>, <x>, <x>)
# <x> is used for point coordinates
<x> ::= [0, 15]
# <dimension> indicates x, y or z
<dimension> ::= 0 | 1 | 2
<radius> ::= <x>
<phase> ::= [0.0, 1.0]
<revs> ::= [1, 7]
```

Appendix B

Design Brief

B.1 Pedestrian Footbridges for the townships of Glebe & St. Mullins

B.1.1 Background

The Holy Well of St. Mullins is located on the opposite shore of the river from the historic enclave of churches and gravesite for which St. Mullins is so well known, yet forms an integral part of this religious site. Each year on the Sunday nearest to July 25 a pilgrimage, or pattern as it is known locally, is made between the ecclesiastical enclosure to the west of the site to the Holy Well on the east bank of the river. This is a significant public event in the life of this small community drawing many former residents back to the town for a week of celebration. Currently access to the well during this event, as well as throughout the year, is achieved using the road bridge which lies just north of the well itself. However historically access would have been through the river itself over a set of stepping stones which create a fording point at low water. These stones, and the stone steps on the east bank leading down to them are still intact at the south-eastern tip of the site on which the Holy Well stands. More recently there were two small timber bridges used to link the church site with the well, one located just south of the stepping stones and a second crossing the mill race to achieve access to the eastern face of the church enclosure. These bridges reinforced what was presumed to be the original Pilgrims Path from the rear of the historic churches to the well and, in addition, provided the community with access to the otherwise inaccessible, but very beautiful, lands surrounding the old mill race. In the 1990s these bridges were removed and since that time the only access to the well has been via the road bridge. As the lands surrounding the Holy Well and the millrace are now in public hands there is an opportunity to both reinstate and enhance the original Pilgrims Path with new timber bridges. The students and staff of the UCD School of Architecture, Landscape and Civil Engineering, based on a close survey of the lands in the autumn of 2009, made of proposition to Carlow County Council to undertake the design and construction of a set of small pedestrian bridges to rehabilitate the original pilgrimage route in cooperation with the Council. Based on this agreement a proposal for a renewed route has been developed which involves the installation of three new pedestrian bridges in this landscape.

B.1.2 Nature & Extent of Proposal

The intention of the proposal is to create a full circuit to and from the Holy Well site from the church enclosure. It is proposed that the route starts at the Churches and, leading down the existing stone stairway at the rear of the church enclosure, enters the landscape to the east of the millrace from where a path, to be reinstated by the Council would lead down the hill to the site of the bridge that crosses the millrace. From here pilgrims can reach the east side of the river by crossing a second bridge over the river adjacent to the historic ford created by the stepping stones. Having reached the Holy Well site there would be an opportunity to return to the church enclosure via an additional bridge over the river at the newly landscaped park, leading up to the cemetery itself. The location and configuration of the largest bridge, located in the park site, has additional rational beyond completing a full circuit of movement for the pilgrims. As the landscape surrounding the east face of the church enclosure and the mill race is very steep and often uneven or unstable, even with the addition of the two bridges this route would be difficult for anyone with mobility problems. However, as the park has a new hard surface path, recently constructed by the Council, a bridge spanning from this path to the eastern side of the river could be designed to accommodate people with disabilities, rather than leaving them to the use of the road bridge. This would require an extension of a hard surface path from the east side of the new bridge to the Holy Well itself to facilitate access, which is proposed as future work not encompassed by the current proposal. The extent of the current proposal is limited to the construction of three oak bridges, including all work required to provide stable footings, the specifics of which are detailed in the accompanying drawings. Further work on the rehabilitation or creation of related paths are outlined for future work and are not included in this proposal.

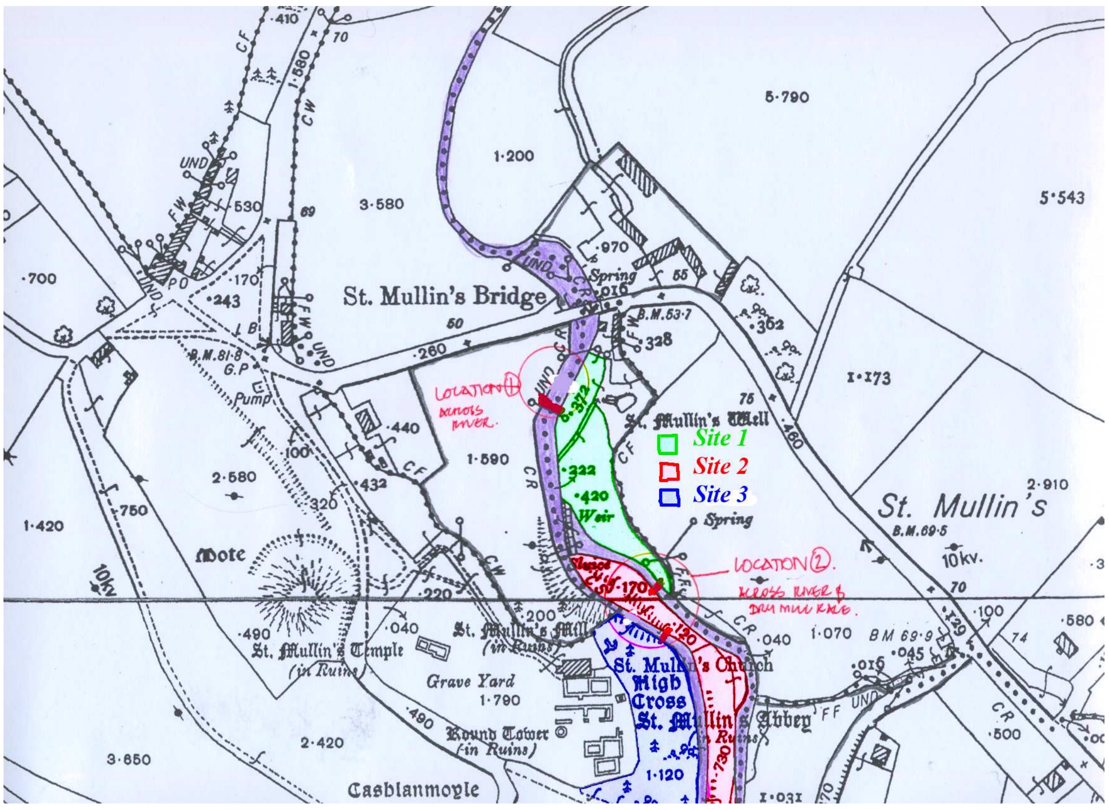
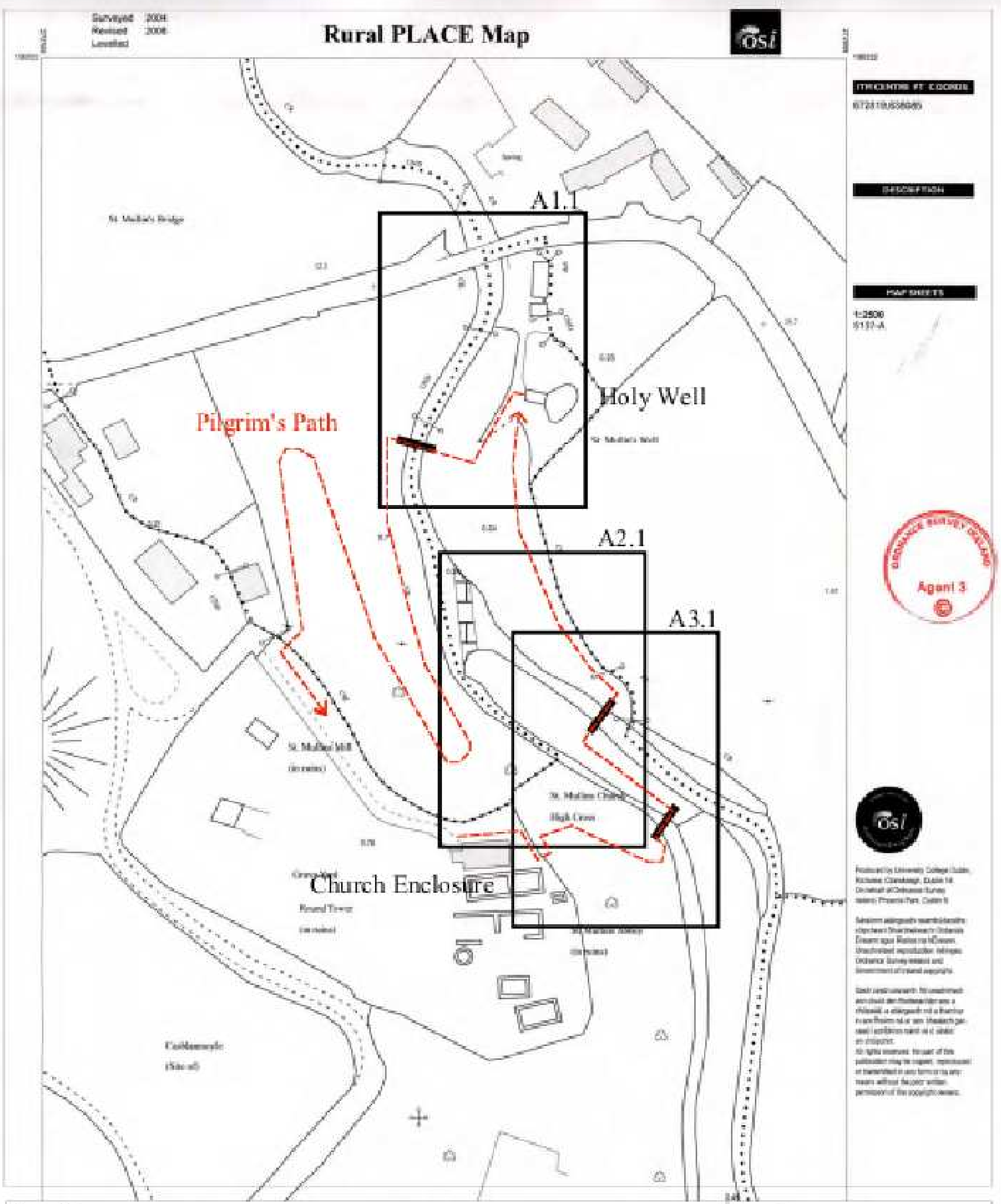
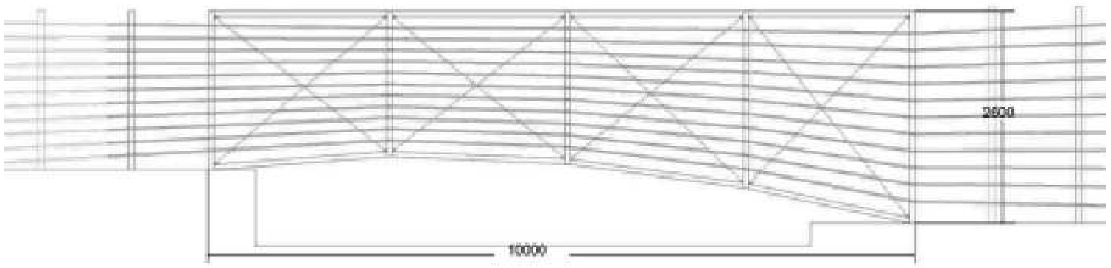
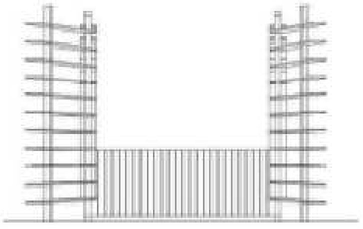


Fig. B.1: Map of St Mullins with the possible bridge locations highlighted.

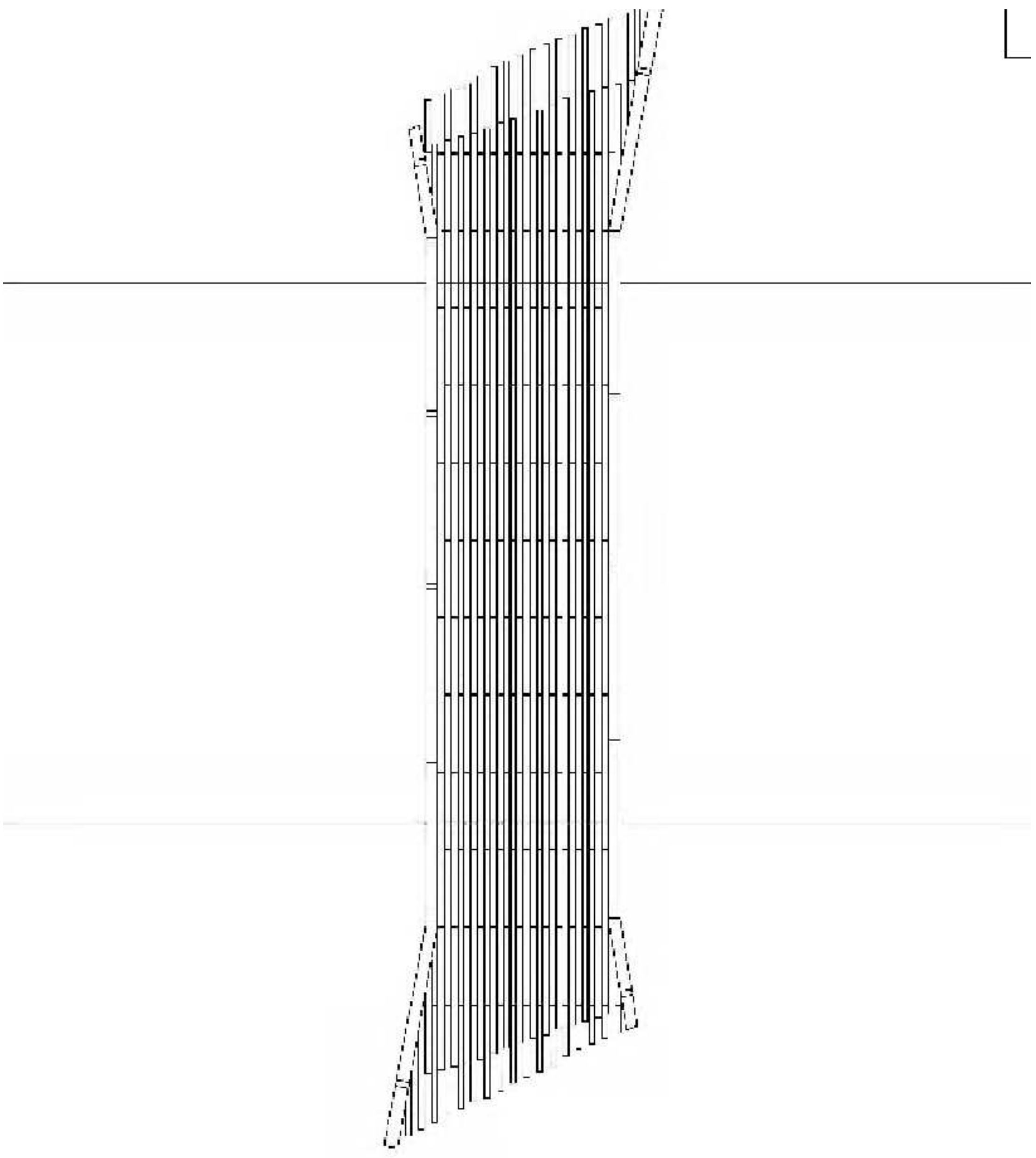


Project Title Pedestrian Footbridges for the townships of Glebe and St. Mullin's	Sheet No. A0
Project Consultants Carlow County Council in cooperation with the School of Architecture, Landscape & Civil Engineering, University College Dublin	Sheet Title Site Plan
Date of Issue February 11, 2010 for planning approval	Drawing Scale 1:1000





Project Title Pedestrian Footbridges for the townships of Glebe and St. Mullin's	Sheet No. A1.2
Project Consultants Carlow County Council in cooperation with the School of Architecture, Landscape & Civil Engineering, University College Dublin	Sheet Title Long elevation and end elevation
Date of Issue February 11, 2010 for planning approval	Drawing Scale 1:50



Project Title

Pedestrian Footbridges for the townships of Glebe and St. Mullin's

Project Consultants

Carlow County Council in cooperation with the
School of Architecture, Landscape & Civil Engineering, University College Dublin

Date of Issue

February 11, 2010 for planning approval

Sheet No.

A1.3

Sheet Title

Bridge Plan

Drawing Scale

1:50

Appendix C

Surveys

C.1 Survey for Initial Interactivity Experiment

```
Thank you for completing the experiment, please answer the questions below.
Some of the questions have a scale, just mark an X anywhere on the dotted line to show your preference.

When using single button, did you manage to complete the task successfully?
<all of the time>           <some of the time>             <none of the time>
X------------------------------------X----------------------------------------X

When using double buttons, did you manage to complete the task successfully?
<all of the time>           <some of the time>             <none of the time>
X------------------------------------X----------------------------------------X

For single button, did you see similarity between the bridges you selected and the generated bridges
<very similar>               <kind of similar>            <completely random>
X------------------------------------X----------------------------------------X

For left click on the double button experiment, did you see similarity between
the bridges you selected and the generated bridges
<very similar>               <kind of similar>            <completely random>
X------------------------------------X----------------------------------------X


For right click on the double button experiment, did you see similarity between
the bridges you selected and the generated bridges
<very similar>               <kind of similar>            <completely random>
X------------------------------------X----------------------------------------X
```

```
For single button, in terms of reaching the target, how would you describe the
outcome?
<very close>                <some similarities>            <not close at all>
X------------------------------------X--------------------------------------X

For double button, in terms of reaching the target, how would you describe the
outcome?
<very close>                <some similarities>            <not close at all>
X------------------------------------X--------------------------------------X

For single button, how would you describe the task?
<very interesting>             <indifferent>              <very frustrating?>
X------------------------------------X--------------------------------------X

For double buttons, how would you describe the task?
<very interesting>             <indifferent>              <very frustrating?>
X------------------------------------X--------------------------------------X

For double button, Did you see a difference between left and right buttons?
<very different               <some difference>               <no difference>
X------------------------------------X------------------------------------X

List some of the features you used when comparing the bridges.
    <big differences>                <small differences>

--------------------------              --------------------------

--------------------------              --------------------------

--------------------------              --------------------------

--------------------------              --------------------------


Any other comments?

-------------------------------------------------------------

-------------------------------------------------------------
```

C.2 Survey for Initial Interactivity Experiment

Fig. C.1: Survey question 1.

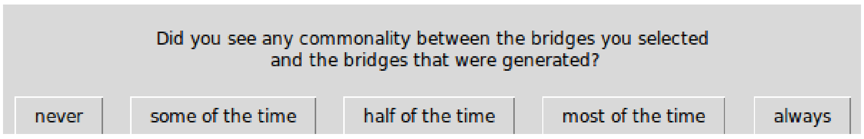

Did you see any commonality between the bridges you selected
and the bridges that were generated?

Fig. C.2: Survey question 2.

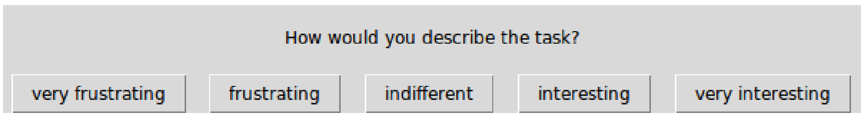

How would you describe the task?

Fig. C.3: Survey question 3.

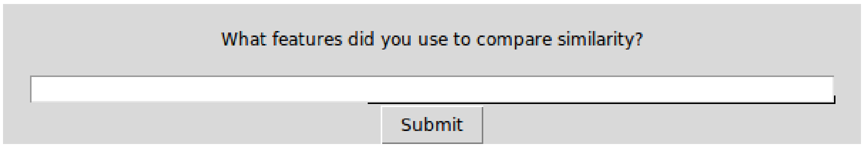

What features did you use to compare similarity?

Fig. C.4: Survey question 4.

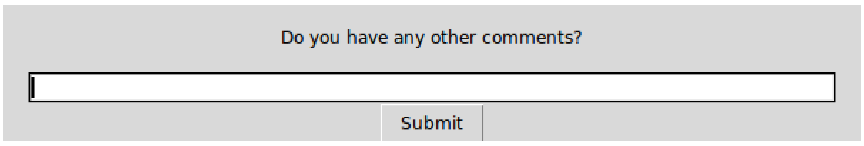

Do you have any other comments?

Fig. C.5: Survey question 5.

Bibliography

[1] M. Agarwal and J. Cagan. A blend of different tastes: the language of coffeemakers. *Environment and Planning B*, 25:205–226, 1998.

[2] ANSYS. Ansys, simulation driven product development. `http://www.ansys.com/`, 2011.

[3] K. Aoki and H. Takagi. Interactive GA-based design support system for lighting design in 3-D computer graphics. *Trans. of IEICE*, 81:1601–1608, 1996.

[4] Autodesk. Autocad. `http://docs.autodesk.com/ACD/2010/ENU/`, 2010.

[5] Autodesk. Robot structural analysis software. `http://usa.autodesk.com/robot-structural-analysis-professional/`, 2011.

[6] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, et al. Revised report on the algorithmic language ALGOL 60. *The Computer Journal*, 5(4):349–367, 1963.

[7] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6(2-3):325–354, 1994.

[8] Wolfgang Banzhaf. Genotype-phenotype-mapping and neutral variation-a case study in genetic programming. In *Proceedings of Parallel Problem Solving from Nature III*, volume LNCS 866, pages 322–332. Springer, 1994.

[9] P. Baron, R. Fisher, A. Tuson, F. Mill, and A. Sherlock. A voxel-based representation for evolutionary shape optimization. *AI EDAM*, 13(03):145–156, 1999.

[10] Bentley. Staad, structural analysis and design. `http://www.bentley.com/en-US/Products/Structural+Analysis+and+Design/`, 2011.

[11] P. Bentley. The revolution of evolution in design: From coffee tables to hospitals. *Proceedings of Recent Advances in Soft Computing*, 98:172–182, 1998.

[12] Peter Bentley and Sanjeev Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. ISBN 1-55860-611-4. URL `http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GA-329.ps`.

[13] Peter J. Bentley. Is evolution creative? In Peter J. Bentley and David W. Corne, editors, *Proceedings of the AISB'99 Symposium on Creative Evolutionary Systems (CES)*, pages 28–34, 1999.

[14] Peter J. Bentley. Exploring component-based representations-the secret of creativity by evolution? In Ian C. Parmee, editor, *Proceedings of the Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2000)*, pages 161–172, University of Plymouth, 2000.

[15] Peter J. Bentley and Una-May O'Reilly. Ten steps to make a perfect creative evolutionary design system. In *GECCO 2001 Workshop on Non-Routine Design with Evolutionary Systems*, 2001.

[16] P.J. Bentley and J.P. Wakefield. The evolution of solid object designs using genetic algorithms. *Modern Heuristic Search Methods*, pages 199–215, 1996.

[17] P.J. Bentley and J.P. Wakefield. Overview of a Generic Evolutionary Design System. In *Proceedings of the 2nd On-Line World Conference on Evolutionary Computation (WEC2)*, pages 53–56, 1996.

[18] Boris Georg Bezirtzis, Matthew Lewis, and Cara Christeson. Interactive evolution for industrial design. In *C&C '07: Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, pages 183–192, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-712-4. doi: http://doi.acm.org/10.1145/1254960.1254986.

[19] J. Biles. GenJam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*, pages 131–131. International Computer Music Association, 1994.

[20] O. Bohnenberger, J. Hesser, and R. Manner. Automatic design of truss structures using evolutionary algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 143. IEEE, 1995.

[21] K. Bollinger, M. Grohmann, and O. Tessman. Form, force, performance: Multiparametric structural design. *Architectural Design*, 78(2):20–25, 2008.

[22] T.L. Booth. *Sequential machines and automata theory.* Wiley, 1967.

[23] A. Brabazon and M O'Neill. *Natural computing in computational finance*, volume 1. Springer Verlag, 2008.

[24] O. Brandte and S. Malinchik. A broad and narrow approach to interactive evolutionary design-an aircraft design example. in et al., kd, editor, genetic and evolutionary computation–gecco 2004. In *Proceedings of the Genetic and Evolutionary Computation Conference. Part II*, pages 883–895, 2004.

[25] R. Breukelaar, MTM Emmerich, and T. Back. On Interactive Evolution Strategies. *Lecture Notes in Computer Science*, 3907:530, 2006.

[26] British Standards Institution. *BS EN 338-2003: Structural Timber Strength Classes.* BSI,London, 2003.

[27] B.J. Bush and H. Sayama. Hyperinteractive evolutionary computation. *Evolutionary Computation, IEEE Transactions on*, 15(3):110, 2011.

[28] J. Byrne, M. O'Neill, and A. Brabazon. Structural and nodal mutation in grammatical evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1881–1882. ACM, 2009.

[29] J. Byrne, M. O'Neill, and A. Brabazon. Optimising offensive moves in toribash. In R. Matousek, editor, *Proceedings of Mendel 2010 16th International Conference on Soft Computing*, pages 78–85, Brno, Czech Republic, 2010. Brno University of Technology.

[30] Jonathan Byrne, Michael O'Neill, Erik Hemberg, and Anthony Brabazon. Analysis of constant creation techniques on the binomial-3 problem with grammatical evolution. In Andy Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 568–573, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press.

[31] Jonathan Byrne, James McDermott, Edgar Galván López, and Michael O'Neill. Implementing an intuitive mutation operator for interactive evolutionary 3d design. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2010.

[32] Jonathan Byrne, James McDermott, Michael O'Neill, and Anthony Brabazon. An analysis of the behaviour of mutation in grammatical evolution. In *Genetic Programming, Proceedings of EuroGP'2010*, pages 14–25. Springer-Verlag, 2010.

[33] Jonathan Byrne, Michael Fenton, Erik Hemberg, James McDermott, Michael O'Neill, Elizabeth Shotton, and Ciaran Nally. Combining structural analysis and multi-objective criteria for evolutionary architectural design. In *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, Evo-*

MUSART, EvoSTIM, EvoTRANSLOG, volume 6625 of *LNCS*, pages 200–209, Turin, Italy, 27-29 April 2011. Springer Verlag.

[34] Jonathan Byrne, Erik Hemberg, and Michael O'Neill. Interactive operators for evolutionary architectural design. In *GECCO '11: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 43–44, Dublin, Ireland, 12-16 July 2011. ACM. doi: doi:10.1145/2001858.2001884.

[35] C. Caldwell and V. S. Johnston. Tracking a criminal suspect through face-space with a genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann, 1991.

[36] R. Cilibrasi and P. M. B. Vitanyi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[37] R. Cleary. Extending grammatical evolution with attribute grammars: An application to knapsack problems. *Master of science thesis in computer science, University of Limerick, Ireland*, 2005.

[38] J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life, See http://EndlessForms.com*, 2011.

[39] C. Coia and B.J. Ross. Automatic evolution of conceptual building architectures. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1140–1147. IEEE, 2011.

[40] Dan Costelloe and Conor Ryan. Genetic programming for subjective fitness function identification. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Applications of Evolutionary Computing*, pages 259–268, Heidelberg, 2004. Springer-Verlag. ISBN 978-3-540-21346-8.

[41] Chris Coyne. Context free art. `http://www.contextfreeart.org/`, 2010.

[42] W. Cui, A. Brabazon, and M. O'Neill. Efficient trade execution using a genetic algorithm in an order book based artificial stock market. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2023–2028. ACM, 2009.

[43] D. Cvetković and I. Parmee. Agent-based support within an interactive evolutionary design system. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(5):331–342, 2002.

[44] Richard Dawkins. *The Blind Watchmaker.* Longman Scientific and Technical, Harlow, England, 1986.

[45] K. Deb and S. Gulati. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elements in Analysis and Design*, 37(5):447–465, 2001.

[46] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002. ISSN 1089-778X.

[47] Ian Dempsey. *Grammatical Evolution in Dynamic Environments.* PhD thesis, University College Dublin, Ireland, 2007.

[48] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments.* Springer, 2009.

[49] P. Deutsch and J.-L. Gailly. Zlib compressed data format specification version 3.3, 1996.

[50] A. Devert, N. Bredeche, and M. Schoenauer. Artificial ontogeny for truss structure design. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 298–305. IEEE, 2008.

[51] N. Dorris, B. Carnahan, L. Orsini, and LA Kuntz. Interactive evolutionary design of anthropomorphic symbols. In *CEC2004: Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, 2004.

[52] K. Duncker and L.S. Lees. On problem-solving. *Psychological monographs*, 58(5):i, 1945.

[53] Economic and Social Research Institute. Cityengine, GIS modeller. `http://www.esri.com/software/cityengine/`, 2011.

[54] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Monographs on statistics and applied probability. Chapman & Hall, 1993. ISBN 9780412042317. URL `http://books.google.ie/books?id=gLlpIUxRntoC`.

[55] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124–141, 1999.

[56] Henrik Esbensen, Avant Corporation, and Ernest S. Kuh. Explorer: An interactive floorplanner for design space exploration. In *Proc. of the European Design Automation Conference*, pages 356–361, 1996.

[57] David Fagan, Michael O'Neill, Edgar Galvan-Lopez, Anthony Brabazon, and Sean McGarraghy. An analysis of genotype-phenotype maps in grammatical evolution. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 62–73, Istanbul, 7-9 April 2010. Springer.

[58] Michael Fenton. Analysis of timber structures created using a g.e-based architectural design tool. Master's thesis, University College Dublin, Ireland, 2010.

[59] D.B Fogel, N.S. Inc, and C.A. La Jolla. Imagining machines with imagination. *Proceedings of the IEEE*, 88(2):284–288, 2000.

[60] John Frazer. *An evolutionary architecture*. Architectural Association, London, 1995. ISBN 1870890477 9781870890472.

[61] Pascal J. Frey. MEDIT:interactive mesh visualization. 0 RT-0253, INRIA, 12 2001. URL `http://hal.inria.fr/inria-00069921/en/`.

[62] Chris Gathercole and Peter Ross. The MAX problem for genetic programming - highlighting an adverse interaction between the crossover operator and a restriction on tree depth. Technical report, Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN, UK, 1995. URL `http://citeseer.ist.psu.edu/gathercole95max.html`.

[63] John S. Gero. Creativity, emergence and evolution in design. *Knowledge-Based Systems*, 9(7):435 – 448, 1996. ISSN 0950-7051. doi: DOI:10.1016/S0950-7051(96)01054-4. URL `http://www.sciencedirect.com/science/article/B6V0P-3VV6NXC-9/2/bc517c1f03e1a428da7c56c38bf576a7`.

[64] John S. Gero, Vladimir A. Kazakov, Department Of Architectural, and Design Science. An exploration-based evolutionary model of generative design process. In *Microcomputers In Civil Engineering*, pages 209–216, 1996.

[65] J. Gips. Computer implementation of shape grammars. In *NSF/MIT Workshop on Shape Computation*. Citeseer, 1999.

[66] E.H. Glaylord and C.N. Glaylord. *Structural engineering handbook*. McGraw-Hill, 1979.

[67] Faustino J. Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 113–120, Montréal, Canada, 2009. ACM.

[68] Google. sketchup, 3d modeling for everyone, version 8. `http://sketchup.google.com/`, 2011.

[69] GTS. Strap, the structural analysis package. `http://www.gtscad.com/strap.htm`, 2011.

[70] Z. Gu, M. Xi Tang, and J.H. Frazer. Capturing aesthetic intention during interactive evolution. *Computer-Aided Design*, 38(3):224–237, 2006.

[71] S. Gustafson and L. Vanneschi. Crossover-based tree distance in genetic programming. *Evolutionary Computation, IEEE Transactions on*, 12(4):506–524, 2008.

[72] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[73] R. Harper. GE, explosive grammars and the lasting legacy of bad initialisation. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[74] R. Harper and A. Blair. A structure preserving crossover in grammatical evolution. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume Volume 3, pages 2537–2544. IEEE Press, 2005.

[75] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978. ISBN 0201029553.

[76] David A. Hart. Toward greater artistic control for interactive evolution of images and animation. In Mario Giacobini, editor, *Applications of Evolutionary Computing*, volume 4448 of *LNCS*, pages 527–536. Springer, 2007. ISBN 978-3-540-71804-8.

[77] N. Hayashida and H. Takagi. Visualized IEC: Interactive evolutionary computation with multidimensional data visualization. In *IECON-PROCEEDINGS-*, volume 4, pages 2738–2743, 2000.

[78] N. Hayashida and H. Takagi. Acceleration of EC convergence with landscape visualization and human intervention. *Applied Soft Computing*, 1:245–256, 2002.

[79] E. Hemberg, L. Ho, M. ONeill, and H. Claussen. A symbolic regression approach to manage femtocell coverage using grammatical genetic programming. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 639–646. ACM, 2011.

[80] Erik Hemberg and James McDermott. PonyGE, an implementation of grammatical evolution in Python. `http://code.google.com/p/ponyge/`, 2011.

[81] Erik Anders Pieter Hemberg. *An Exploration of Grammars in Grammatical Evolution*. PhD thesis, University College Dublin, Ireland, 17 September 2010.

[82] Martin Hemberg, Una-May O'Reilly, Achim Menges, Katrin Jonas, Michel da Costa Goncalves, and Steve Fuchs. Genr8: Architect's experience using an emergent design tool. In Penousal Machado and Juan Romero, editors, *The Art of Artificial Evolution*, pages 167–188. Springer-Verlag, Berlin, November 2007.

[83] Weidermann J. Hoeffler A, Leysner U. Optimization of the layout of trusses combining strategies based on Mitchels theorem and on biological principles of evolution. In *Proceeding of the 2nd Symposium on Structural Optimisation*, Milan,Italy, 1973.

[84] D. Holzer, R. Hough, and M. Burry. Parametric design and structural optimisation for early design exploration. *International Journal of Architectural Computing*, 5(4): 625–643, 2007.

[85] Gregory S. Hornby and Jordan B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 600–607. IEEE Press, 27-30 May 2001.

[86] G.S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1729–1736. ACM, 2005.

[87] J. Hugosson, E. Hemberg, A. Brabazon, and M. O'Neill. An investigation of the mutation operator using different representations in grammatical evolution. *In Proc. 2nd*

International Symposium "Advances in Artificial Intelligence and Applications", 2: 409–419, 2007.

[88] IBISWorld. Best laid plans: Growth in new building construction will pave the way for a recovery. *NAICS 54131*, 54131, 2011.

[89] Christian Igel. Causality of hierarchical variable length representations. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 324–329, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press. URL `http://www.neuroinformatik.ruhr-uni-bochum.de/PEOPLE/igel/CoHVLR.ps.gz`.

[90] Byrne J., Hemberg E., O'Neill M., and Brabazon A. A local search interface for interactive evolutionary architectural design. In *Applications of Evolutionary Computing, EvoApplications 2012: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, LNCS, Malaga, Spain, 2012. Springer Verlag.

[91] M.J. Jakiela, C. Chapman, J. Duda, A. Adewuya, and K. Saitou. Continuum structural topology design with genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):339–356, 2000.

[92] P. Janssen, J. Frazer, and T. Ming-xi. Evolutionary Design Systems and Generative Processes. *Applied Intelligence*, 16(2):119–128, 2002.

[93] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, Albuquerque, 1995.

[94] M. Keijzer, V. Babovic, C. Ryan, M. ONeill, and M. Cattolico. Adaptive logic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 42–49, 2001.

[95] M. Keijzer, C. Ryan, M. O Neill, M. Cattolico, and V. Babovic. Ripple crossover in genetic programming. *Lecture notes in computer science*, pages 74–86, 2001.

[96] M. Keijzer, M. ONeill, C. Ryan, and M. Cattolico. Grammatical evolution rules: The mod and the bucket rule. *Genetic Programming*, pages 123–130, 2002.

[97] R.E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, Stanford University, CA, USA, 1996. MIT Press.

[98] Rafal Kicinger, Tomasz Arciszewski, and Kenneth DeJong. Evolutionary design of steel structures in tall buildings. *Journal of Computing in Civil Engineering*, 19(3):223–238, 2005. doi: 10.1061/(ASCE)0887-3801(2005)19:3(223). URL `http://link.aip.org/link/?QCP/19/223/1`.

[99] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers and Structures*, 83(23-24):1943 – 1978, 2005. ISSN 0045-7949. doi: DOI:10.1016/j.compstruc.2005.03.002.

[100] H Koning and J Eizenberg. The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B*, 8:295–323, 1981. URL `http://www.envplan.com/abstract.cgi?id=b080295`.

[101] A. Kosorukoff. Human based genetic algorithm. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 5, pages 3464–3469. IEEE, 2001.

[102] S. Koulouridis, D. Psychoudakis, and J.L. Volakis. Multiobjective optimal antenna design based on volumetric material optimization. *Antennas and Propagation, IEEE Transactions on*, 55(3):594–603, 2007.

[103] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

[104] John R. Koza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, Norwell, MA, USA, 2003. ISBN 1402074468.

[105] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 222–230, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann. URL `http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.max_gp97.pdf`.

[106] B. Lawson. *How designers think: the design process demystified*. Elsevier/Architectural, 2006. ISBN 9780750660778. URL `http://books.google.ie/books?id=lPvqZJNAdG8C`.

[107] San Le. San le's free finite element analysis. `http://slffea.sourceforge.net/`, 2010.

[108] H.C. Lee and M.X. Tang. Evolving product form designs using parametric shape grammars integrated with genetic programming. *AI EDAM*, 23(02):131–158, 2008.

[109] M. Li and PMB Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer Verlag, 1997.

[110] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi. The similarity metric. *Information Theory, IEEE Transactions on*, 50(12):3250–3264, 2004.

[111] A. Lindenmayer and G. Rozenberg. Parallel generation of maps: Developmental systems for cell layers. In *Graph-grammars and their application to computer science and biology*, pages 301–316. Springer, 1979.

[112] Xavier Llorà, Kumara Sastry, David E. Goldberg, Abhimanyu Gupta, and Lalitha Lakshmi. Combating user fatigue in iGAs: partial ordering, support vector machines, and synthetic fitness. In *GECCO*, pages 1363–1370. ACM, 2005.

[113] Edgar Galván López, John Mark Swafford, Michael O'Neill, and Anthony Brabazon. Evolving a ms. pacman controller using grammatical evolution. In *EvoApplications (1)*, pages 161–170, 2010.

[114] Edgar Galvan Lopez, James McDermott, Michael O'Neill, and Anthony Brabazon. Defining locality as a problem difficulty measure in genetic programming. *Genetic Programming and Evolvable Machines*, 12(4):365–401, 2011.

[115] S. Luke. Two fast tree-creation algorithms for genetic programming. *Evolutionary Computation, IEEE Transactions on*, 4(3):274–283, 2000.

[116] P. Machado, H. Nunes, and J. Romero. Graph-based evolution of visual languages. *Applications of Evolutionary Computation*, pages 271–280, 2010.

[117] Penousal Machado and Amlcar Cardoso. All the truth about nevar. *APPLIED INTELLIGENCE*, 16:101–119, 2002.

[118] Penousal Machado, Juan Romero, Maria Luisa Santos, Amilcar Cardoso, and Bill Manaris. Adaptive critics for evolutionary artists. In G.R. Raidl et al., editors, *Applications of Evolutionary Computing: EvoWorkshops 2004*, number 3005 in LNCS, pages 437–446. Springer-Verlag, 2004.

[119] A. Machwe and I.C. Parmee. Integrating aesthetic criteria with evolutionary processes in complex, free-form design–an initial investigation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.

[120] A. Machwe, I.C. Parmee, and J.C. Miles. Integrating Aesthetic Criteria with a User-centric Evolutionary System via a Component-based Design Representation. In *International Conference on Engineering Design ICED*, volume 5, pages 15–18, 2005.

[121] Azahar T. Machwe and Ian C. Parmee. Towards an interactive, generative design system: Integrating a 'build and evolve' approach with machine learning for complex freeform design. In Mario Giacobini, editor, *Applications of Evolutionary Computing*, volume 4448 of *LNCS*, pages 449–458. Springer, 2007. ISBN 978-3-540-71804-8.

[122] Jay P McCormack, Jonathan Cagan, and Craig M Vogel. Speaking the buick language: capturing, understanding, and exploring brand identity with shape grammars.

Design Studies, 25(1):1 – 29, 2004. ISSN 0142-694X. doi: 10.1016/S0142-694X(03)00023-1.

[123] J. McDermott, M. O'Neill, and N.J.L. Griffith. Interactive ec control of synthesized timbre. *Evolutionary Computation*, 18(2):277–303, 2010.

[124] J. McDermott, U.M. OReilly, L. Vanneschi, and K. Veeramachaneni. How far is it from here to there? a distance that is coherent with gp operators. *Genetic Programming*, pages 190–202, 2011.

[125] J McDermott, J Swafford, E Hemberg, J Byrne, M Fenton, C McNally, E Shotton, and M O'Neill. An assessment of string-rewriting grammars for evolutionary architectural design. *Environment and planning B*, In Press, 2012.

[126] James McDermott, Jonathan Byrne, John Mark Swafford, Michael O'Neill, and Anthony Brabazon. Higher-order functions in aesthetic EC encodings. In *2010 IEEE World Congress on Computational Intelligence*, pages 2816–2823, Barcelona, Spain, 2010. IEEE Press. doi: doi:10.1109/CEC.2010.5586077.

[127] J.M. McDermott. *Evolutionary Computation Applied to the Control of Sound Synthesis*. PhD thesis, University of Limerick, 2008.

[128] R.I. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, and M. ONeill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11 (3):365–396, 2010.

[129] Robert McNeel and Associates. Rhino, nurbs modelling for windows. `http://www.rhino3d.com/`, 2011.

[130] G.A. Miller. The magical number seven, plus or minus two. *Psychological review*, 63:81–97, 1956.

[131] Julian Miller. Evolving a self-repairing, self-regulating, french flag organism. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation GECCO 2004*, vol-

ume 3102 of *Lecture Notes in Computer Science*, pages 129–139. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-22344-3.

[132] Eoin Murphy, Michael O'Neill, and Anthony Brabazon. A comparison of GE and TAGE in dynamic environments. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1387–1394, Dublin, Ireland, 12-16 July 2011. ACM. doi: doi:10.1145/2001576.2001763.

[133] James E. Murphy. *Applications of Evolutionary Computation to Quadrupedal Animal Animation*. PhD thesis, School of Computer Science and Informatics, University College Dublin, Ireland, March 2011. URL `http://www.james-e-murphy.ie/thesis.html`.

[134] NASA. Openvsp, open source parametric geometry. `http://www.openvsp.org/`, 2012.

[135] M. Nicolau and D. Costelloe. Using grammatical evolution to parameterise interactive 3d image generation. *Applications of Evolutionary Computation*, pages 374–383, 2011.

[136] P. Nordin. Genetic programming iii-darwinian invention and problem solving. *Evolutionary Computation*, 7(4):451–453, 1999.

[137] M. Ohsaki. An input method using discrete fitness values for interactive GA. *Journal of Intelligent and Fuzzy Systems*, 6(1):131–145, 1998.

[138] M. Ohsaki and H. Takagi. Improvement of presenting interface by predicting the evaluation order to reduce the burden of human interactive EC operators. In *1998 IEEE International Conference on Systems, Man, and Cybernetics, 1998*, volume 2, 1998.

[139] M. O'Neill and A. Brabazon. Grammatical differential evolution. In *International Conference on Artificial Intelligence (ICAI06)*, pages 231–236, 2006.

[140] Michael O'Neill. *Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution*. PhD thesis, University Of Limerick, Ireland, 2001.

[141] Michael O'Neill. Natural computing research and applications group. `http://ncra.ucd.ie/`, 2011.

[142] Michael O'Neill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462, 2006.

[143] Michael O'Neill and Anthony Brabazon. Evolving a logo design using Lindenmayer systems, Postscript and grammatical evolution. In *IEEE Congress on Evolutionary Computation 2008*, pages 3788–3794, Hong Kong, China, 2008. IEEE Press.

[144] Michael O'Neill and Conor Ryan. Crossover in grammatical evolution: A smooth operator? In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 149–162, Edinburgh, 15-16 April 2000. Springer-Verlag. ISBN 3-540-67339-3. URL `http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=1802&spage=149`.

[145] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003. ISBN 1402074441.

[146] Michael O'Neill, Conor Ryan, Martin Keijzer, and Mike Cattolico. Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4(1), 2003.

[147] Michael O'Neill, Erik Hemberg, Eliott Bartley, James McDermott, and Anthony Brabazon. GEVA - grammatical evolution in Java (v1.0). Technical Report Technical Report UCD-CSI-2008-09, Available from `http://ncra.ucd.ie/geva/`, UCD School of Computer Science, 2008.

[148] Michael O'Neill, Erik Hemberg, Eliott Bartley, James McDermott, and Anthony Brabazon. GEVA: Grammatical evolution in Java. *SIGEVOlution*, 3(2):17–22, 2008.

[149] Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1035–1042, Montreal, 8-12 July 2009. ACM. doi: doi:10.1145/1569901.1570041.

[150] Michael O'Neill, James McDermott, John Mark Swafford, Jonathan Byrne, Erik Hemberg, Elizabeth Shotton, Ciaran McNally, Anthony Brabazon, and Martin Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 2011.

[151] U.M. O'Reilly and P. Testa. Representation in architectural design tools. *Proceedings of ACDM-2000*, 2000.

[152] Una-May O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *IEEE International Conference on Systems, Man, and Cybernetics: Computational Cybernetics and Simulation*, volume 5, 1997.

[153] Una-May O'Reilly and Martin Hemberg. Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines*, 8(2):163–186, June 2007. ISSN 1389-2576. doi: doi:10.1007/s10710-007-9025-y. Special issue on developmental systems.

[154] Una-May O'Reilly and Girish Ramachandran. A preliminary investigation of evolution as a form design strategy. In Christoph Adami, Richard K. Belew, Hiroaki Kitano, and Charles E. Taylor, editors, *Proceedings of the Sixth International Conference on Artificial Life*, pages 443–447, University of California, Los Angeles, 26-29 June 1998. MIT Press. ISBN 0-262-51099-5. URL `http://citeseer.ist.psu.edu/oreilly98preliminary.html`.

[155] M. ONeill, A. Brabazon, M. Nicolau, S. Garraghy, and P. Keenan. πgrammatical evolution. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 617–629. Springer, 2004.

[156] I.C. Parmee. Improving problem definition through interactive evolutionary computation. *AI EDAM*, 16(03):185–202, 2002. doi: 10.1017/S0890060402163050. URL `http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=130661&fulltextType=RA&fileId=S0890060402163050`.

[157] IC Parmee and CR Bonham. Towards the support of innovative conceptual design through interactive designer/evolutionary computing strategies. *AI EDAM*, 14(01): 3–16, 2000.

[158] O. Pironneau, F. Hecht, AL Hyaric, and K. Ohtsuka. Freefem. *URL: http://www. freefem. org*, 2006.

[159] R. Poli, N.F. McPhee, and W.B. Langdon. *A Field Guide to Genetic Programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[160] P. Prezemyslaw and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, NY, USA, 1990.

[161] Michael Pugliese and Jonathan Cagan. Capturing a rebel: modeling the Harley-Davidson brand through a motorcycle shape grammar. *Research in Engineering*

Design, 13:139–156, 2002. ISSN 0934-9839. URL http://dx.doi.org/10.1007/s00163-002-0013-1. 10.1007/s00163-002-0013-1.

[162] I. Rechenberg. *Evolution strategy: optimization of technical systems according to the principles of biological evolution.* Frommann-Holzboog, Stuttgart, 1973.

[163] J. Romero and P. Machado. *The art of artificial evolution: a handbook on evolutionary art and music.* Springer-Verlag New York Inc, 2007.

[164] Franz Rothlauf. On the locality of representations. In *GECCO*, pages 1608–1609, 2003.

[165] Franz Rothlauf and Marie Oetzel. On the locality of grammatical evolution. Working Paper 11/2005, Department of Business Administration and Information Systems, University of Mannheim, D-68131 Mannheim, Germany, December 2005. URL http://wifo1.bwl.uni-mannheim.de/fileadmin/files/publications/workingpaper-locality.pdf.

[166] Conor Ryan and R. Azad. Sensible initialisation in chorus. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2003. ISBN 978-3-540-00971-9.

[167] J. Secretan, N. Beato, D.B. D Ambrosio, A. Rodriguez, A. Campbell, and K.O. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1759–1768. ACM, 2008.

[168] SGI. openfoam, the open source CFD toolbox. http://www.openfoam.com/, 2011.

[169] D. Shasha and K. Zhang. Fast Parallel Algorithms for the Unit Cost Editing Distance Between Trees. In *SPAA '89: Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 117–126, New York, NY, USA, 1989. ACM. ISBN 0-89791-323-X. doi: http://doi.acm.org/10.1145/72935.72949.

[170] K. Shea, R. Aish, and M. Gourtovaia. Towards integrated performance-driven generative design tools. *Automation in Construction*, 14(2):253–264, 2005. ISSN 0926-5805.

[171] K. Shea, I.F.C. Smith, et al. Improving full-scale transmission tower design through topology and shape optimization. *Journal of structural engineering*, 132:781, 2006.

[172] H.A. Simon. Rational choice and the structure of the environment. *Psychological review*, 63(2):129–138, 1956. ISSN 0033-295X.

[173] H.A. Simon. *The sciences of the artificial.* The MIT Press, 1996.

[174] Karl Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on computer graphics and interactive techniques*, pages 319–328, New York, NY, USA, 1991. ACM. ISBN 0-89791-436-8. doi: http://doi.acm.org/10.1145/122718.122752.

[175] S. Smith and J. Rieffel. A face-encoding grammar for the generation of tetrahedral-mesh soft bodies. In *Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems (ALIFE12).*, pages 414–420. MIT Press., 2010.

[176] Oasys Software. GSA, structural analysis version 8.5. `http://www.oasys-software.com/gsa-analysis.html`, 2011.

[177] Rhino Software. Grasshopper, generative modelling. `http://www.grasshopper3d.com/`, 2010.

[178] Intact Solutions. Scan and solve, in situ analysis for Rhino. `http://www.intact-solutions.com/`, 2011.

[179] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[180] Stichting Blender Foundation. Blender 3D. `http://www.blender.org/`, 2009. Last viewed 11 May 2009.

[181] G. Stiny. Introduction to shape and shape grammars. *Environment and planning B*, 7(3):343–351, 1980.

[182] Rainer Storn and Kenneth Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341 – 359, 1997.

[183] Bentley Sytems. Generative components, v8i. `http://www.bentley.com/getgc/`, 2011.

[184] M. Tacker, P. F. Stadler, E. G. Bornberg-Bauer, I. L. Hofacker, and P. Schuster. Algorithm Indepedent Properties of RNA Secondary Structure Predictions. *European Biophysics Journal*, 25(2):115–130, 1996.

[185] H. Takagi and K. Kishi. On-line knowledge embedding for an interactive ec-based montage system. In *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, pages 280–283. IEEE, 1999.

[186] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. of the IEEE*, 89(9):1275–1296, 2001.

[187] M Tapia. A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design*, 26(1):59–73, January 1999. URL `http://ideas.repec.org/a/pio/envirb/v26y1999i1p59-73.html`.

[188] A. Thompson. *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution.* Distinguished dissertation series. Springer-Verlag, 1998.

[189] BHV Topping and JPB Leite. Parallel genetic models for structural optimization. *Engineering Optimization*, 31(1):65–99, 1988. ISSN 0305-215X.

[190] Kazutoshi Tsutsumi and Keisuke Sasaki. Study on shape creation of building's roof by evaluating aesthetic sensibility. *Math. Comput. Simul.*, 77(5-6):487–498, 2008. ISSN 0378-4754. doi: http://dx.doi.org/10.1016/j.matcom.2007.11.022.

[191] T. Unemi. Simulated breeding–a framework of breeding artifacts on the computer. *Kybernetes*, 32(1/2):203–220, 2003.

[192] J. Ventrella. Disney meets darwin-the evolution of funny animated figures. In *Computer Animation'95., Proceedings.*, pages 35–43. IEEE, 1995.

[193] E.H. Weber. *De Pulsu, resorptione, auditu et tactu: Annotationes anatomicae et physiologicae.* CF Koehler, 1834.

[194] P.A. Whigham. *Grammatical Bias for Evolutionary Learning.* PhD thesis, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1996.

[195] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[196] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/882262.882324.

[197] B.G. Woolley and K.O. Stanley. On the deleterious effects of a priori objectives on evolution and representation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, 2011.

[198] Tina Yu. Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction. *Genetic Programming and Evolvable Machines*, 2(4):345–380, 2001.

[199] K. Yue, R. Krishnamurti, and F. Gobler. Computation-friendly shape grammars. *School of Architecture*, 2009.

[200] O.C. Zienkiewicz and PB Morice. *The finite element method in engineering science*, volume 7. McGraw-Hill London, 1971.

[201] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4):257–271, 1999.

[202] ZH Zuo, YM Xie, and X. Huang. Combining genetic algorithms with beso for topology optimization. *Structural and Multidisciplinary Optimization*, 38(5):511–523, 2009.