Abstract

The aim of this thesis is to investigate how some civil engineering design problems, in particular structures, can be represented using evolutionary algorithms (EA) and contains two, independent experimental chapters on building layout design and geometric dome design (an introduction to EAs and design is also provided).

Civil engineering design problems are typically approached using traditional techniques i.e. deterministic algorithms, rather than via stochastic search such as EAs. However EAs are adept at exploring fragmented and complex search spaces, such as those found in design, but do require potential solutions to have a 'representation' amenable to evolutionary operators. Four canonical representations have been proposed including: strings (generally used for parameter based problems), voxels (shape discovery), trees and graphs (skeletal structures).

Several authors have proposed design algorithms for the conceptual layout design of commercial office buildings but all are limited to buildings with rectangular floor plans. This thesis presents an evolutionary algorithm based methodology capable of representing buildings with orthogonal boundaries and atria by using a 3-section string with real encoding, which ensures the initialisation and evolutionary operations are not too disruptive on column alignments encoded via the genome. In order to handle orthogonal layouts polygon-partitioning techniques are used to decompose them into rectangular sections, which can be solved individually. However to prevent the layout becoming too discontinuous, an 'adjacency graph' is proposed which ensures column line continuity throughout the building.

Dome geometric layout design is difficult, because every joint and member must be located on the external surface and not impinge on the internal void. This thesis describes a string-based representation capable of designing directly in 3D using surface area and enclosed volume as the major search parameters. The representation encodes support and joint positions, which are converted into a dome by constructing its corresponding convex hull. Once constructed the hull's edges become the structural members and its vertices the joints. This avoids many of the problems experienced by the previous approach, which suffers when restrictive constraints such as the requirement to maintain $1/8^{th}$ symmetry are removed.

Declaration

Declaration:

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ……. (candidate)
Date …….

Statement 1:

This dissertation is the result of my own independent work/ investigation, expect where otherwise stated.
Other sources are acknowledged by explicit references. A bibliography is appended.

Signed ……. (candidate)
Date …….

Statement 2:

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ……. (candidate)
Date …….

Acknowledgements

I would like to express gratitude to my supervisors Prof J.C. Miles and Prof W.A. Gray for their tireless support and guidance during this research project and the Departments of Engineering and Computer Science at Cardiff University for sponsoring me.

I must also acknowledge the continual encouragement given by my mother and brother over the years. It certainly gave me the lift I needed to complete this work.

Finally, I need to thank my incredibly long-suffering girlfriend Helen who has endured the most and deserves a break from me droning on about computers and evolutionary algorithms. You are my guiding star.

Contents

List of Figures

List of Tables

List of Abbreviations

DNA	Deoxyribonucleic Acid
DSS	Decision Support Systems
EA	Evolutionary Algorithm
EP	Evolutionary Programming
ES	Evolutionary Strategies
FSM	Finite State Machines
GA	Genetic Algorithms
GP	Genetic Programming
NFL	No free lunch theorems

1 Introduction

1.1 Aim

The aim of this work is to investigate how some civil engineering design problems, in particular structures, can be represented in evolutionary algorithms. To achieve this aim, the thesis will consider two types of structural design problem: buildings and domes, both will be investigated by reviewing existing work, proposing a new solution (including a representation with associated evolutionary operators) and providing an illustrative example to assess performance. However it should be noted that each chapter is self contained and should be considered as such. The only link between them is that the same methodology was applied to both.

Conceptual design is a particularly pertinent topic as an efficient representation is essential in effectively harnessing the search capacity of evolutionary algorithms in decision support systems for conceptual design. At the present time, conceptual design is considered to be one of the most difficult challenges facing practising engineers. This is because the lack of information limits the effectiveness of procedural techniques to assist more junior designers. Therefore only senior engineers undertake this work as they have the necessary experience.

1.2 Objectives

This work has two main objectives:

- Investigate existing and develop new knowledge for orthogonal building layout design.
- Investigate existing and develop new knowledge for geometric dome design.

1.3 Arrangement of Thesis

The remaining chapters of this thesis are arranged as follows:

1.3.1 Chapter 2: An overview of evolutionary algorithms

This chapter provides an overview of evolutionary algorithms, a family of algorithms that search problem domains using biologically inspired search operators, and is the type of algorithm used in this thesis. It starts with the topic of search and solution spaces before reviewing several categories of search techniques. Next, biological evolution is discussed

because evolutionary algorithms are inspired by nature, before the chapter focuses on the components of a basic evolutionary algorithm including: initialisation, evaluation, evolution and termination. Finally, the chapter concludes with a description of the canonical implementations: evolutionary programming; evolutionary strategies; genetic algorithms and genetic programming.

1.3.2 Chapter 3: Representing civil engineering design problems in evolutionary algorithms

Civil engineering design problems are typically approached using traditional techniques i.e. deterministic algorithms, rather than via stochastic search. Evolutionary algorithms are a type of stochastic search algorithm inspired by natural selection and a number of authors have proposed them as a design tool. This chapter discusses how solutions to civil engineering design problems, in particular structures, have been represented in evolutionary algorithms without considering implementation specific issues. The aim of this chapter is to consider representations used by other researchers.

1.3.3 Chapter 4: Conceptual layout design of orthogonal commercial buildings

The aim of this chapter is to investigate existing examples and develop new representation for orthogonal building layout design.

Conceptual layout design of commercial office buildings is a non-trivial task because the numerous variables create a large solution space. To aid designers, several decision support systems have been developed. However, all these systems are limited to buildings with rectangular floor plans.

This chapter presents an evolutionary algorithm for layout design of buildings with orthogonal boundaries and atria. To achieve this, polygon partitioning techniques are used to decompose a floor plan into rectangular sections. Also in order to prevent illegal solutions being generated, the representation ensures the initialisation and evolutionary operations are not too disruptive. The number of initial inputs has also been reduced, because this work is aimed at the conceptual design stage. Therefore the user only needs to dimension the external boundary and specify the location of any atria.

1.3.4 Chapter 5: Conceptual geometric design of 'geodesic-like' domes

The aim of this chapter is to investigate existing and develop new knowledge for geometric dome design.

Geometric dome design is a non-trivial task because every joint and member must be located on the dome's external surface and not impinge on the internal void. The only previous stochastic methodology (Shea and Cagan, 1997) tackles this by creating a 2D truss that is subsequently projected onto a predefined curved surface. Therefore the solution is a 3D object, but the search is conducted in 2D. While this 'projection' or 2.5D technique reduces the number of problem variables, by constraining the third dimension to be dependent on the planar layout, it also excludes a dome's two most important variables from the search: surface area and enclosed volume. Thus the results, while spatially innovative, are typically sub-optimal.

This chapter describes a new methodology, using an evolutionary algorithm with string representation that is capable of designing a dome directly in 3D using surface area and enclosed volume as the major search parameters. The representation contains Point3D objects that encapsulate support and joint positions, which are subsequently converted into a dome by constructing its convex hull. Once constructed, the hull's edges become the structural members and its vertices the joints. Finally, structural analysis is used to determine performance within the context of user-defined constraints. This technique avoids many of the problems experienced by the previous approach that suffers when restrictive constraints such as the requirement to maintain $1/8^{th}$ symmetry are removed.

1.3.5 Chapter 6: Summary and future work

This chapter will consider the key findings, of this thesis, in relation to its original objectives and discuss possible directions for future work.

2 An Overview of Evolutionary Algorithms

2.1 Abstract

This chapter provides an overview of evolutionary algorithms. Evolutionary algorithms are a family of algorithms that search problem domains using biologically inspired search operators and are the type of algorithm used in this thesis. The chapter starts with the topic of search and solution spaces before reviewing several categories of search techniques. Next, biological evolution is discussed, because evolutionary algorithms are inspired by nature, before the chapter focuses on the components of a basic evolutionary algorithm including: initialisation, evaluation, evolution and termination. Finally, the chapter concludes with a description of the canonical implementations: evolutionary programming; evolutionary strategies; genetic algorithms and genetic programming.

Keywords: search, evolutionary algorithms, evolutionary programming, evolutionary strategies, genetic algorithms, genetic programming.

2.2 Introduction

For every problem, a range of possible solutions must exist: with some solutions being more feasible than others. The problem's 'solution space' (Figure 2-1a) is constructed by incorporating a notional distance between solutions. To solve the problem, the solution space is 'searched' to locate the optimal values, often equivalent to finding a maxima or minima.

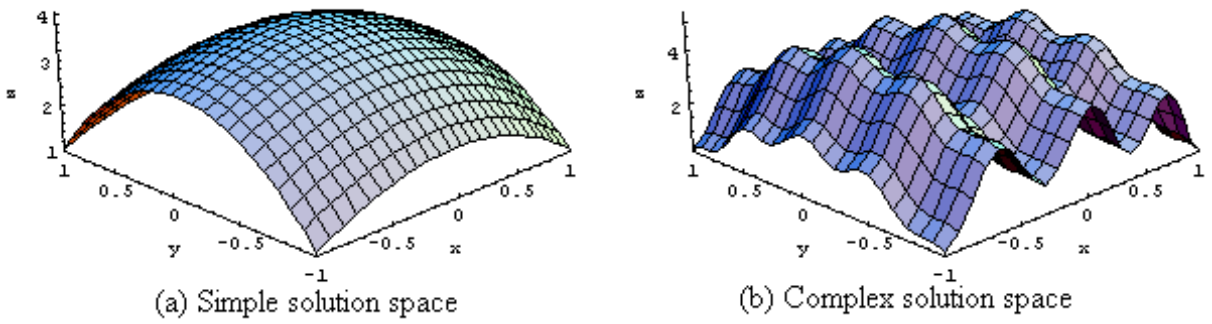


Figure 2-1 Example simple and complex solution spaces

Unfortunately, solution spaces are seldom simple. For most non-trivial problems they are ill defined (with the search process often generating new points) and contain many local or

false optima (Figure 2-1b). These complications are additional to the issues of where to start the search, how to conduct it and strategy for limiting the potential for pre-mature convergence. Consequently, search is generally a non-trivial task.

Primarily two types of search have been developed: stochastic and deterministic, although a third type 'hybrid' incorporating stochastic and deterministic elements (Figure 2-2) has also been developed.

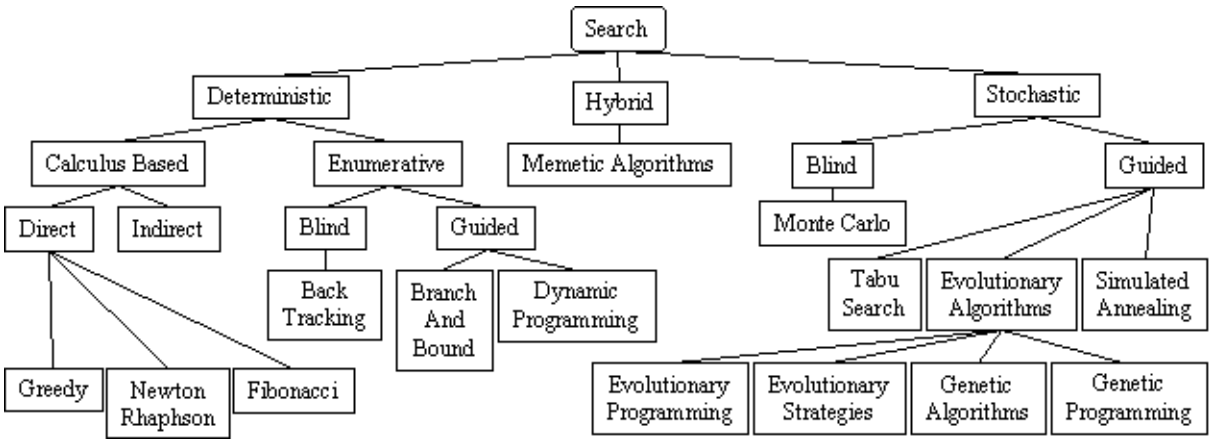


Figure 2-2 Indicative hierarchy of search (adapted from Goldberg, 1989)

Deterministic techniques are either calculus based requiring the problem to be modelled using derivatives (which may or may not be possible), or enumerative, which can suffer from the 'curse of dimensionality' if the solution space is large. However, if the solution space is a continuous smooth surface or well understood, a deterministic technique is often the most appropriate approach. Another disadvantage of deterministic algorithms is that they are often not robust enough to cope with 'noisy' data (as found in 'real world' problems) and domain knowledge maybe required to formulate and solve the problem, so this approach is less useful for conceptual design.

Stochastic algorithms, unlike deterministic techniques, are built on randomness, which improves the search for global optima by sampling random locations in the solution space. However, while this creates a more 'robust' algorithm capable of handling noisy data, it does mean that stochastic search cannot guarantee to find the global optimum solution.

All search techniques must distinguish between local and global optima. This issue is particularly pertinent if some variables are discrete, as discrete variables create a discontinuous and disjointed solution space. A simple remedy for coping with local optima is

to re-run the algorithm from another location and compare results, this is particularly important when using deterministic algorithms.

This work uses stochastic search algorithms because structural design uses a combination of related, discrete and continuous variables that can create extremely large, disjointed search spaces.

2.3 Why have so many search algorithms been developed?

Numerous search algorithms have been developed because their performance is problem dependant. This is because the algorithm's assumptions maybe incorrect or utilise a methodology that is inefficient for the given solution domain. Consequently, there is no search panacea. This is emphasised by the 'no free lunch theorems (NFL)', which consider the utility of search algorithms a priori, without assumptions and from mathematical principles alone. The NFL theorems conclude *"…any elevated performance over one class of problems is exactly paid for in performance over another class…"* (Wolpert and Macready, 1997). However, in practise, search maybe improved by incorporating additional domain specific information called 'heuristics'. For example consider the 'travelling salesman problem'[1]. The solution space is well known therefore a deterministic technique incorporating heuristics may out perform another more general, stochastic technique. However if the problem's parameters are changed, the algorithm containing heuristics may perform worse because the heuristics are invalid.

2.4 Biological inspiration for algorithms

Mankind has always been fascinated by nature's ability to create solutions to complex problem and this led to the development of a family of algorithms based on evolution, evolutionary algorithms. However, it is important to note that evolutionary algorithms are only inspired by nature, not a duplicate. For example in nature, alleles can be dominant or recessive. However this feature is not often included in EAs. For a more comprehensive description of EAs see 2.5 Evolutionary Algorithms.

[1] The 'travelling salesman problem' is a deceptively simple combinatorial problem: "A salesman spends his time visiting a number of cities. During one trip, he visits each city only once and finishes where he started. In what order should he visit the cities to minimise the total distance travelled?"

The following sub-sections contain a brief discussion of two important issues in biological evolution, from the perspective of search, Darwin's theory of natural selection and phenotype-genotype duality.

2.4.1 Darwin's theory of natural selection

Darwin's theory of natural selection (Darwin, 1859) proposes that organisms evolve over time due to environmental factors that favour certain traits. Roberts et al. (1993) summarised it into four propositions and two conclusions:

- ***Proposition 1***: individuals are different.
- ***Proposition 2***: offspring generally resemble their parents.
- ***Proposition 3***: not every offspring can survive to reproduce.
- ***Proposition 4***: fitter individuals are more likely to survive.
- ***Conclusion 1***: individuals that survive and reproduce, pass on to their offspring characteristics that have enabled them to succeed.
- ***Conclusion 2***: in time, a group of individuals that once belonged to the same species may give rise to two different groups that are sufficiently distinct to belong to separate species.

Unfortunately 'The Origin of the Species' is often reduced to a single phrase 'survival of the fittest' but this is misleading, as an individual's mortality is a relatively trivial issue in evolutionary terms. Fitness, in evolutionary terms, refers to the degree of adaptation shown by an individual to its environment. The most adapted individuals will have the best fitness and therefore pass on these beneficial characteristics to their offspring. The best individuals will often have many adaptations so it not necessarily the strongest, fastest or biggest that will prevail.

Ultimately, if a species is to be successful its population must balance two issues:

- ***Selection:*** which reduces diversity (propositions 3,4 and conclusion 1).
- ***Reproduction:*** which introduces variation (propositions 1,2 and conclusion 2).

Managing this conflict via populations represents one of biological evolution's greatest strengths, as it encourages trial and error by favouring advantageous characteristics within a species.

2.4.2 Phenotype-genotype duality

Every cell in a living organism incorporates helical strands of deoxyribonucleic acid (DNA) that encodes its phenotype (features and function). A gene is a short section of DNA that contains the instructions for a single feature e.g. eye colour. However, each gene may have several values e.g. eye colour = blue/ green/ brown, and these values are called alleles. An organism's physical characteristics (its phenotype) are determined by the DNA sequence of its genes: its genotype. Therefore, every organism can be viewed from either a genotypic or phenotypic perspective: with the genotype encoding the phenotype.

2.5 Evolutionary Algorithms

Although there are many different types of evolutionary algorithm (EA), "*…the common idea…is to evolve a population of candidate solutions to a given problem, by using search operations inspired by biology…*" (Dumitrescu et al, 2000). This section introduces the basic EA by considering every major component.

2.5.1 Why use evolutionary algorithms?

Evolutionary algorithms are very good at discovering diverse solutions to problems but are not pure optimisation algorithms (De Jong, 1993). In spite of this they have made important contributions to this field especially with regard to problems involving mixed solution spaces (containing discrete and continuous variables) and in multi-objective optimisation. However, they tend to be out-performed in combinatorial and continuous parametric optimisation by more traditional techniques (Eiben and Schoenauer, 2002). Nevertheless, EAs were considered the most appropriate technique for this work because of the following characteristics:

- EAs can investigate large numbers of inter-related parameters.
- EAs are adept at locating global optima even in discontinuous solution spaces.

- EAs are robust[2]

It should also be noted that this work is focused on using EAs for design rather than optimisation and "*…one should distinguish design problems where the goal is to find at least one very good solution once, from day-to-day optimisation where the goal is to consistently find a good solution for different inputs. In the design context, a high standard deviation is desirable provided the average result is not too bad (exploration). In the optimisation context, a good average and a small deviation are mandatory (exploitation)…*" (Eiben and Schoenauer, 2002).

2.5.2 Representation

Evolutionary algorithms are problem solvers that create solutions by applying search operators based on biological evolution. Unfortunately, most problems are not instantly amenable to biological search operators. Therefore, the potential solutions must be converted to a form that can be used by the EA. This involves developing a 'representation'. Although there is some ambiguity in literature about what constitutes a representation, in this thesis 'representation' refers to the structure and encoding of a solution so that it can be incorporated into an EA.

The primary purpose of a representation is to convert every possible solution to a form that allows it to be included in the search. It should also be a compromise between computational effort and problem abstraction e.g. machine code is computationally efficient but how can it be used to represent a house?

Many standard representations exist e.g. strings, and this determines how the EA is applied to the problem, as some components of the EA are representation dependent. Bäck et al. (1997) describe two approaches to developing a representation: "*…the first is to choose one of the standard algorithms and to design a decoding function according to the requirements of the algorithm. The second suggests designing the representation as close as possible to the characteristics of the phenotype, almost avoiding the need for a decoding function…*". Many researchers use the first method but the second generates a more efficient representation.

[2] The balance between efficiency and efficacy i.e. the more robust the algorithm, the greater the range of problems it can be applied to (Coley, 2003).

2.5.3 Representation space

Living organisms exhibit a phenotype-genotype duality because an organism's characteristics are encoded in its DNA. In the same way, individuals[3] within an EA also exhibit duality because they can be viewed from the perspective of their representation or 'natural' form. Therefore when a problem is solved by an EA, it has two problem domains, the solution space and the representation space (Figure 2-3). The solution space contains solutions in their natural form while the representation space contains solutions encoded via the representation.

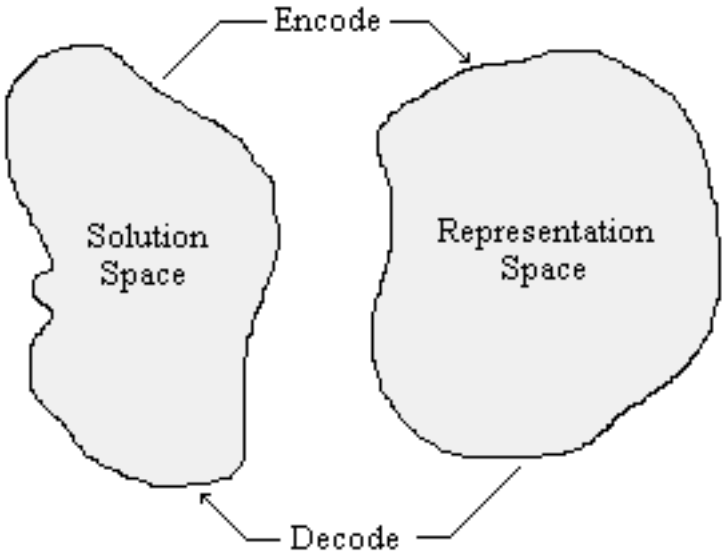


Figure 2-3 Solution and representation spaces

When solving most non-trivial problems, constraint handling is required because not every combination of variables is valid. For example, in structural engineering constraints are often applied to structural members, indicating the permissible maximum stress. Therefore, constraints define the boundaries of the feasible region. Conceptually this is equivalent to dividing the representation space into islands of feasible representations, surrounded by an infeasible region (Figure 2-4).

[3] EAs terminology has borrowed heavily from biology. A potential solution in an EA can be referred to as an individual.

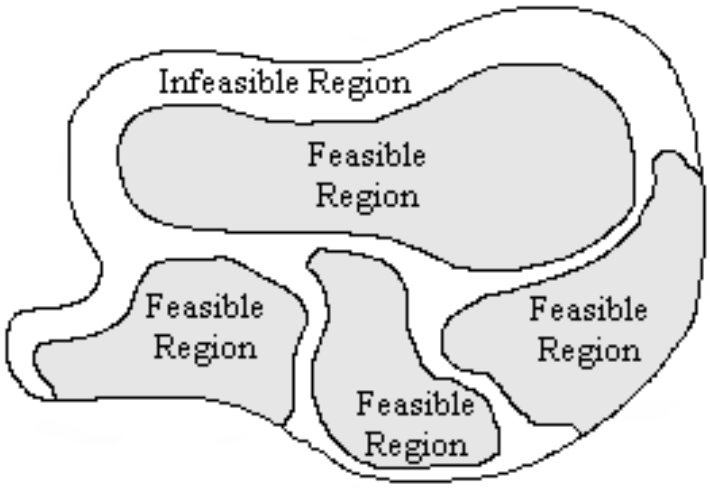


Figure 2-4 Representation space with feasible and infeasible regions

A fundamental feature of all EAs is that they operate on solutions encoded via the representation rather than directly on the solution. At first glance this may seem a disadvantage as it adds additional complexity. In reality, by converting solutions to a more abstract form, via the representation, the EA permits the use of standardised evolutionary operators.

It should be noted that while designing a representation is a vital stage in the development of an EA once complete, the representation (and its related operators) is hidden from the user allowing them to concentrate on analysis of the results (Borkowski and Grabska, 1995).

To convert between the two problem domains, a mapping must be applied. However, pleiotrophy[4] and polygeny[5] mean there are potentially five types of mapping (Figure 2-5):

- ***Illegal***: a representation decodes to form a nonsensical solution. For example, if solutions are mathematical equations, = y x + / 3 would be illegal.
- ***Infeasible***: in constrained problems, or those with discrete variables, not every combination of variables results in a feasible solution. Therefore the representation space is larger than the solution space, as it contains infeasible individuals. It should be noted that

[4] The effect that a single gene may simultaneously affect several phenotypic traits (Fogel, 1995).

[5] The effect that a single phenotypic characteristic (of an individual) maybe determined by the simultaneous interaction of many genes (Fogel, 1995).

infeasible solutions are different from illegal solutions: as infeasible solutions are invalid due to the constraints rather than being nonsensical or plain wrong.

- ***1-to-n***: one representation decodes to form *n* solutions. Obviously this is undesirable as a single representation can have multiple fitness values.
- ***n-to-1***: *n* solutions decode to form a single solution, while this scenario is applicable in practice it does enlarge the representation space increasing search difficulty.
- ***1-to-1***: one representation decodes to form a single solution. This is the ideal scenario as the solution and representation spaces are identical in size.

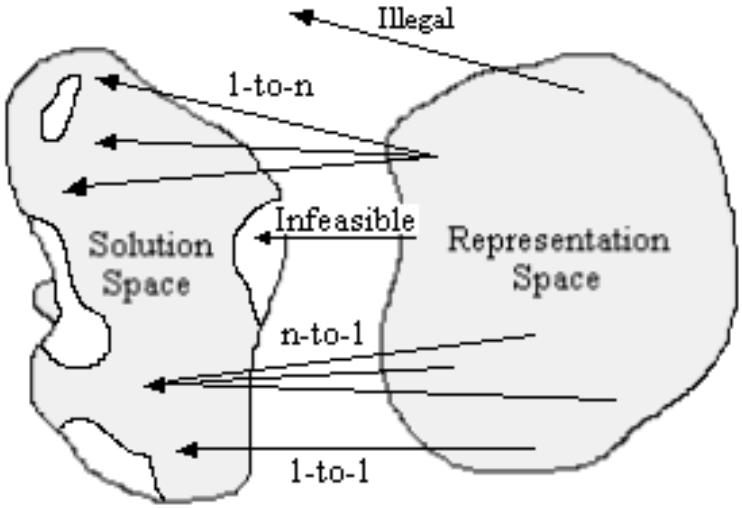


Figure 2-5 Mappings from representation to solution space

2.6 The basic evolutionary algorithm

This section describes the main components of an evolutionary algorithm, although please note this is a high-level summary avoiding implementation specific issues. The following sections contain more detailed descriptions of the canonical implementations.

The evolutionary search process commences once a problem is identified and a suitable representation is developed. For optimisation problems, the EA attempts to locate and return a single optimum solution while for design problems the EA returns a range of possible solutions that are likely to be sub-optimal. This highlights the need to determine the EA's aims and objectives from the outset so it can be appropriately implemented. In this thesis, the onus was on design and thus every EA tried to return a range of potentially sub-optimal solutions (an exploration process).

2.6.1 Overview

The basic EA (Figure 2-6) starts by initialising the first population[6] of solutions. Initialisation creates individuals from a random set of variables, based on the representation (although the initial population can be 'seeded' with known solutions but this biases the search). The population is then evaluated and assigned a 'fitness' based on how suitable it is, in the context of the problem. The algorithm then checks whether the termination criterion has been met (this usually considers whether a solution of appropriate fitness has been evolved or if a predetermined number of generations has been generated). If the algorithm terminates, it will return the best individual(s) evolved so far and if not, perform the evolutionary routine.

The next generation is evolved from the current population via selection. Selection picks individuals from the current population (based on certain criteria) and allows them to breed and pass on their genetic material (to the next generation). However because selection favours fitter individuals, those with more advantageous characteristics are more likely to be picked.

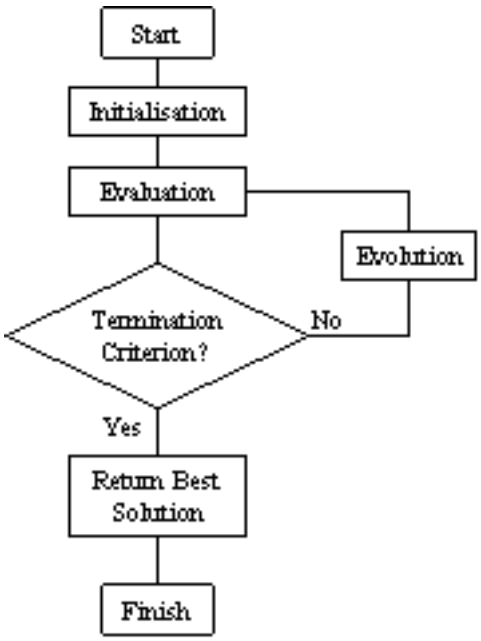


Figure 2-6 Schematic of a basic evolutionary algorithm

The following description of the basic evolutionary algorithm will indicate an advantage of this search technique, most but not all of the methodology is problem independent. Therefore, search can be conducted before a full understanding of the problem is obtained.

[6] A group of potential solutions maintained by the EA

This can be important for complex problems: in fact results generated by the EA may actually provide some insight.

2.6.2 Population- Representation independent component

EAs maintain at least one population of candidate solutions (this is one of the features that separates them from other search techniques) with each individual denoting a location in the representation space. However as in nature, each population must strike a balance between specialisation and variation. Population size is a fundamental variable in EA's as large populations accomplish a more exhaustive search (which maybe unnecessary) but at greater computational expense than a smaller population (which may not cover a sufficient set of the solution space).

2.6.3 Fitness function- Representation independent component

Evolutionary algorithms are domain independent problem solvers i.e. the same algorithm can design buildings and solve scheduling problems, but each problem requires a different solution. Therefore, how does the EA search for the best?

As with biological evolution, individuals within an EA are required to exhibit measurable phenotypic differences. In EAs, individuals then are assigned a single, numerical value that reflects how 'fit' or good it is (as a solution). Unfortunately, assigning a single numerical fitness to an individual can be problematical especially in multi-objective optimisation. In this instance, a multi-objective or multi-criteria algorithm incorporating Pareto based techniques (Pareto, 1896) can be used.

Fitness values are determined by the 'objective function', which contains user-defined information about the solution space. However, the search for the solution to all but the most non-trivial problems must consider constraints. Constraint handling may be required due to problem related issues or simply because as the evolutionary operators only manipulate the genotype while the objective function only considers the phenotype, an evolved solution maybe invalid and occupy a point in the infeasible region. Several constraint-handling techniques exist (Michaelewicz, 1999):

- ***Rejection***: individuals that violate constraints are deleted, focusing the search on the feasible region. However, the loss of potentially valuable information can hamper search especially in disjointed solution spaces and leads to premature convergence.
- ***Reparation***: individuals that violate the constraints are modified to meet the constraints. Unfortunately repairing individuals can be exceptionally complex (or impossible) and thus hinder the search. Reparation also forces solutions to conform to a preconceived notion, which might not be appropriate.
- ***Prevention***: evolutionary operators are designed to prevent the formation of illegal solutions. This can be a practical method for dealing with constraints but can also produce conservative evolutionary operators that may slow the search process.
- ***Penalisation***: individuals that violate the constraints have their fitness reduced. Penalty functions are especially suited to disjointed solution spaces or scenarios where the best solutions lie close to the feasible-infeasible boundary. This is often the case in design, where the optimum is located on the limit of what is feasible.

Once an individual has been assessed by the objective function and any constraint violations considered, its fitness can be determined. Several types of fitness measure may be used (Goldberg, 1989):

- ***Raw fitness***: objective function adjusted for constraint violations (if appropriate).
- ***Adjusted fitness***: an amended raw fitness, where a lower fitness is advantageous. The fittest individual has a fitness of 0.
- ***Standardised fitness***: an amended adjusted fitness, limited to the range $0 \rightarrow 1$.
- ***Scaling***: although not strictly a fitness measure, scaling is used to mitigate the effect of a few highly fit individuals (in early generations) by scaling down the extraordinary and scaling up the rest, or in later generations when the fitness variance tends to fall, exaggerating phenotypic differences between individuals.

2.6.4 Selection- Representation independent component

Selection is used to choose which individuals are allowed to breed and pass on their genetic material to the next generation. Competition based selection is key to EAs as it drives the search and is solely based upon an individual's fitness. However as in nature, selection does

not push the population towards a predetermined goal but merely favours phenotypic changes that have occurred randomly.

Many selection techniques have been developed and the following list indicates some of the most widely used (this list is not exhaustive):

- ***Fitness proportionate***: Compares the raw fitness of the individual against the mean, raw fitness of the population (Holland, 1975). Therefore, an individual that is three times fitter than average, will be selected three times more often. Unfortunately this has two problems:
  - *Premature convergence*: a few sub-optimal but highly fit individuals present in the current population will dominate the next generation by virtue of their large fitness, dramatically reducing the population's genetic variation.
  - *Slow convergence*: if the population only contains individuals of similar fitness, selection pressure will be low, therefore the algorithm degenerates to random search.
- ***Stochastic sampling with replacement ('Roulette Wheel')***: A predetermined number of individuals are randomly selected from the population and placed on a 'roulette wheel': with each individual's segment proportional to its fitness (Baker, 1985). Once the wheel is 'spun', the individual on the winning segment selected.
- ***Stochastic tournament***: A predetermined number of individuals are randomly selected from the population and ranked according to fitness, with the fittest individual being chosen. As the tournament size is increased, selection pressure is intensified as it magnifies the chance of a fit individual being selected.
- ***Ranking***: The population is ranked, based on raw fitness, with the fittest at position 0. Although the actual mapping from rank position to selection probability is arbitrary, in all cases individuals are selected by their rank (not raw fitness). This preserves selection pressure but reduces the effect of dominant individuals.
- ***Elitism***: Ensures that the best member(s) from the last generation are copied into the next. This can be useful because fitness proportionate selection does not guarantee the survival of any individual (Coley, 2003). Elitism is not a selection technique in its own right but can be used in conjunction with others and while it maintains the best solutions, it does increase the likelihood of premature convergence.

2.7 Termination Criterion- Representation independent component

An EA should terminate once the desired solution has been obtained. However stochastic algorithms are not guaranteed to locate the global optimum solution and in many problems, including design, the form of the optimum solution is not known. Therefore problem specific criteria cannot be specified. In this instance, the termination criterion stops the algorithm after a fixed effort has been expended e.g. a predetermined number of generations have been evolved or a maximum number of CPU cycles.

2.8 Initialisation- Representation dependent component

Ideally, initialisation should create a well-distributed spread of individuals in the solution space. Unfortunately in practise this is hard to achieve, especially if the solution space is ill defined. Therefore, individuals are usually randomly constructed from a set of variables.

The initial population often has a low fitness, but its most important feature is diversity. 'Doping' can be used to include good solutions into the population, based on the user's experience, but this can create bias (Dumitrescu et al., 2000).

2.9 Evolutionary operators- Representation dependent component

Search operators, inspired by biology, are a fundamental feature of all EAs. Evolutionary operators alter an individual's genotype (as in biology) and enable EAs to be problem independent. EAs use a mixture of the following three operators (some implementations may even omit an operator altogether):

- ***Reproduction***: copies an individual unaltered into the next generation.
- ***Recombination***: two individuals (parents) are selected and exchange genetic information to produce two new individuals (offspring). Depending on the operator, offspring are either inserted directly into the next generation or inserted after some additional selection. Recombination is referred to as a conservation operation as it "*…is used to exploit and consolidate what has already been obtained by the individuals in the population…*" (Dumitrescu et al., 2000).
- ***Mutation***: a single individual is selected and altered by deleting and randomly rebuilding a section of its genetic information. Mutation is referred to as an 'innovation' operation

because it "*…ensures that new aspects of the problem are taken into account…*" (Dumitrescu et al., 2000).

2.10 Exploration vs. Exploitation

By employing a competition based selection process EAs encourage fitter individuals to pass on their genetic material, which focuses the search (exploitation). However, these individuals may not lie in the most productive region. By contrast, evolution injects new genetic material into the population, which encourages variation and thus the algorithm to consider another area of the solution space (exploration). EAs manage this conflict by allowing the user to set the probability of reproduction, recombination and mutation during a run.

2.11 Implementing Evolutionary Algorithms

The previous section introduced the basic EA without considering specifics. This will discuss the canonical forms of the principle implementations in more detail: Evolutionary Programming; Evolutionary Strategies; Genetic Algorithms; Genetic Programming (for a more comprehensive history of EA development see Fogel (1998)). However, these implementations should not be considered as discrete but rather as the most commonly used forms of evolutionary algorithm (each focusing on different aspects of the evolutionary based search). In fact the experimental chapters will only refer to evolutionary algorithms, as using more explicit descriptions encourages the reader to apply their preconceived ideas rather than focusing on what is being described.

Evolutionary search can be considered from two perspectives, top-down and bottom-up (Fogel, 1995):

- ***Top-down***: emphasises the phenotypic behaviour of individuals (Evolutionary Strategies) or populations (Evolutionary Programming).
- ***Bottom-up***: emphasises the genotypic mechanisms (Genetic Algorithms and Genetic Programming) of evolution.

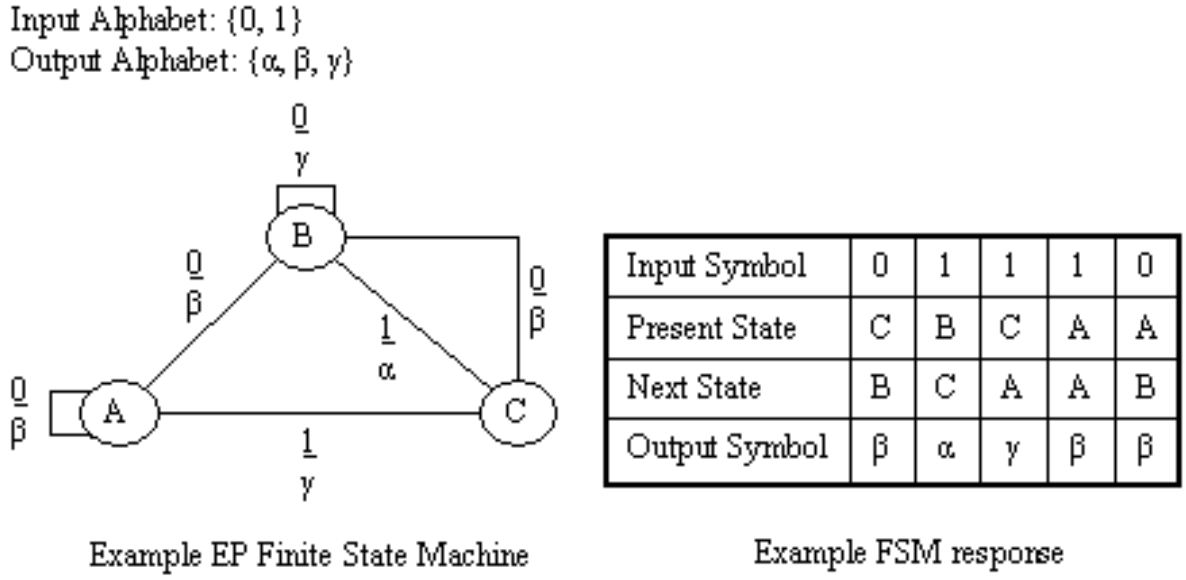
2.11.1 Evolutionary Programming

Developed by Fogel (1962) as an attempt to create artificial intelligence that can predict future events based on historical information, Evolutionary Programming (EP) is used in

continuous parameter optimisation problems. As a representation, EP uses 'Finite State Machines (FSM)' (Figure 2-7) that transforms an input sequence into an output sequence. FSM are composed of at least one state, one or more state transitions (these specify the FSM response to an input, based on its current state) and have a predetermined input and output alphabet.

During a run, a population of FSM receive an identical input sequence and process it. Fitness is assigned based on the accuracy of the response, with a more accurate response receiving a higher fitness. Individuals are then mutated (EP only incorporates mutation) to produce a single offspring. There are five mutation operators: mutate an output; mutate a state transition; insert a new state; delete an existing state; change the initial state. After mutation, the new offspring are evaluated against the initial input sequence. If the offspring is fitter than its parent, it survives, otherwise it is deleted and the parent survives. This process continues until the termination criterion is met.

A classic EP task is to predict the next character in a sequence, when given the last one. For example, consider the response of the three-state machine shown in (Figure 2-7) to the following series of inputs: 01110.



Input Symbol	0	1	1	1	0
Present State	C	B	C	A	A
Next State	B	C	A	A	B
Output Symbol	β	α	γ	β	β

Example FSM response

Figure 2-7 Example EP representation (adapted from Fogel, 2000)

2.11.2 Evolutionary Strategies

Developed by Rechenberg and Schwefel in 1964 (Beyer and Schwefel 2002), Evolutionary Strategies (ES) are a continuous parameter optimisation tool. To encode potential solutions, ES use a representation based on a pair of real-valued vectors *v* (Figure 2-8): the first vector *x* encodes a point in the search space while the second σ is a vector of standard deviations.

$$v = (x, \sigma) = ((10.9, 8.7), (1.0, 1.0))$$

Figure 2-8 Example ES representation

Although created independently, ES shares many similarities with EP including only using mutation as an evolutionary operator. In ES mutation, a vector randomly selected from a Gaussian or Normal distribution with a mean of 0 and variance of σ can mutate each component of the representation. Therefore, the value of σ controls the manner in which the algorithm can search the solution space. Originally, the value of σ was set to produce a fitter offspring at a ratio of 1:5 (Rechenberg, 1973). Thus this is often called the '1/5 success rule'. However, Schwefel (1975) proposed 'self adaptation strategies' that vary mutation parameters (including σ) during a run.

Several mutation-selection techniques have been devised (all ES use the same representation and mutation methodology) that are identified by a notation system unique to ES literature:

- ***(1+1)***: a single individual is present in the population, which mutates to produce a single offspring with only the fittest solution surviving to form the next generation.
- ***(μ+λ)***: μ individuals mutate to produce λ offspring (this produces a population larger than the original). If the offspring is fitter than its parent it survives, else it is deleted and the parent survives.
- ***(μ, λ)***: a population containing μ individuals evolves to produce λ offspring. But because an individual may evolve more than one offspring ($\lambda > \mu$), the next generation is only selected from the offspring. Therefore, an individual can only survive for a maximum of one generation (irrespective of fitness).

Although these mutation-selection strategies have been extensively studied using empirical experiments, ES retains a tendency to converge on local optima: this is confirmed by the only theoretical model of ES mutation (Rudolph, 2001) which suggests that the '1/5 success rule' cannot guarantee convergence during numerical optimisation.

2.11.3 Genetic Algorithms

Holland (1975) is considered to have developed Genetic Algorithms (GA) in 1975 with the publication of his seminal work. However, it is acknowledged that research had been conducted prior to this. Since then GAs have become the most widely known EA and are generally used as combinatorial optimisers although this issue is contentious (De Jong, 1993) because for design problems (as in this thesis) they are often used as search algorithms.

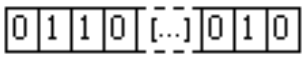


Figure 2-9 Example GA string representation

As a representation, the canonical GA uses a fixed-length, binary string (Figure 2-9) although other encoding are permitted including integers and real numbers. More advanced implementations even allow variable gene length. Other representations including voxels (Griffiths and Miles, 2004) and graphs (Borkowski et al, 2002) have also been developed. Another characteristic of the GA is their stochastic selection techniques and extensive use of recombination and mutation operators inspired by genetics.

2.11.4 Genetic Programming

Developed by Koza (1992), the Genetic Programming (GP) differs from the other EA implementations because it is pre-dominantly used for machine learning. GP is highly suited to this because its canonical tree representation can be constructed from 'LISP S-Expressions' (Figure 2-10), which are computer programmes. Therefore the GP trees can be used to evolve computer programmes and thus solve one of the fundamentals of computing: how can you make computers code themselves? Other representations based on graphs or linear structures have also been developed.

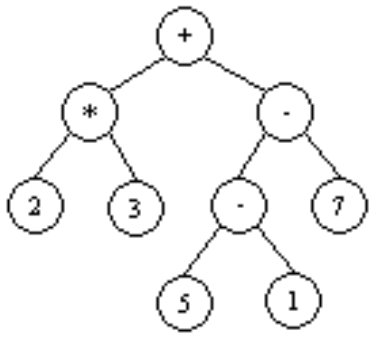


Figure 2-10 Example GP tree representation

The mechanics of the GP closely related to the GA and is even considered by some to be 'a genetic algorithm using a tree based representation'. However unlike the GA, the GP tends to either ignore the mutation operator or use it infrequently. Evolved solutions are also 'active structures' that can be executed without post-processing, while GA's typically operate on coded strings that require post-processing i.e. passive structures.

Within civil engineering the GP is a relatively new technique. Table 2-1 lists most published applications of the genetic programming in civil engineering. In general the GP is used for modelling purposes.

Table 2-1 Applications of genetic programming in civil engineering

Application	Author	Year	Details
Shear strength prediction of deep RC beams	Ashour et al	2003	Estimation of the shear strength of deep RC beams, subjected to two point loads, from 141 published experimental results.
Modelling of wastewater treatment plants	Hong and Bhamidimarri	2003	Modelling the dynamic performance of municipal activated sludge wastewater treatment plants.
Detection of traffic accidents	Roberts and Howard	2002	Detection of accidents on motorways in low flow, high-speed conditions i.e. late at night based on three years of traffic data whilst producing a near zero false alarm rate.
Flow through a urban basin	Dorado et al	2002	Construction of sewage network model in order to calculate the risk posed by rain to the basin and thus providing prior warning of flooding or subsidence.
Journey time prediction	Howard and Roberts	2002	Forecasting motorway journey times.
Estimation of design intent	Ishino and Jin	2002	Using the GP to automatically estimate design intent based on operational and product-specific information monitored throughout the design process.
Modelling of water supply assets	Babovic et al	2002	In order to determine the risk of a pipe burst, a GP is evolved to 'data mine' a database containing information about historic pipe bursts.

Identification of crack profiles	Kojima et al.	2001	Detection of cracks inside hundreds of heat exchanger tubes in a nuclear power plant's steam generator via analysis of data measured via quantitative non-destructive testing.
Modelling rainfall runoff	Whigham and Crapper	2001	Discovery of rainfall-runoff relationships in two vastly different catchments.
Improving engineering design models	Watson and Parmee	1998	Symbolic regression and Boolean induction to model engineering fluid dynamics systems.
Prediction of long-term electric power demand	Lee et al	1997	Symbolic regression via genetic programming to predict the long-term electric demand of Korea (based on training data from 1961 to 1980).
Systems identification	Watson and Parmee	1996	Symbolic regression to calibrate Rolls Royce preliminary design gas turbine cooling systems software.
Traffic light control laws	Montana and Czerwinski	1996	Develop an adaptive control system for a network of traffic signals depending on variations in traffic flow.
Identification of crack profiles	Köppen and Nickolay	1996	Agent generation to detect and track dark regions that could be cracks in greyscale images of textured surfaces.

2.12 Disadvantages of Evolutionary Algorithms

This chapter has, thus far, focused on the positive aspects of evolutionary algorithms. However as previously stated, there is no search panacea and algorithm performance is problem dependent. This section discusses some general disadvantages associated with EAs.

A major disadvantage of evolutionary algorithms is the amount of computational effort expended when solving a problem because rather than solving the problem just once, it evaluates every individual (in every population) at least once per generation. In addition, while the evolutionary operators are computationally trivial e.g. swapping elements, the fitness function tends to be more complex and thus generates a large overhead. For example, Grierson (1993) estimates that 95% of a GA's computational effort is devoted to calculating fitness. However, this figure should be considered indicative, as the actual value (of computational effort) is problem dependent. To counter this, one solution is to use a simple fitness function in early generations, when overall fitness is low.

Humans prefer to organise their conscious thinking in a parsimonious way for example in mathematics it is common practice to simplify equations. However fitness, not parsimony is the dominant factor in evolutionary algorithms. Therefore if a solution performs adequately, there is no fitness advantage and thus no selection pressure to improve it. This problem is particularly prevalent in the GP, as its representations have no fixed shape or size. Unfortunately, this means that solutions generally increase in size during a run: this is called

'programme bloat'. For example, (Jefferson et al, 1990) suggest that on average a GP tree will grow at one level per generation. A bloated solution will contain large sections of inactive code (Bhattacharya and Nath, 2001), which can slow convergence and increase the computational load. Bloat can also result in the evolution of solutions that while accurate, provide no new insight into the problem because of their complexity (Keijzer and Babovic, 1999).

2.13 Conclusions

Evolutionary algorithms are domain independent problem solvers that utilise search operators inspired by biological evolution. Historically four implementations have been developed, which incorporate different representations and are used for different tasks, evolutionary programming is typically used to predict future outcomes based on historical information, evolutionary strategies are used as continuous parameter optimisation tools, genetic algorithms can either be used for discrete parameter optimisation or as a search tool while genetic programming is often used in machine learning.

This thesis will use EAs because they are robust enough to handle issues related to civil engineering design including large numbers of inter-related parameters, discrete and continuous variables creating discontinuous search spaces.

3 Representing Civil Engineering Design Problems in Evolutionary Algorithms

3.1 Abstract

Civil engineering design problems are typically approached using traditional techniques i.e. deterministic algorithms, rather than via stochastic search. Evolutionary algorithms are a type of stochastic search algorithm inspired by natural selection and a number of authors have proposed them as a design tool. This chapter discusses how solutions to civil engineering design problems, in particular structures, can be represented in evolutionary algorithms without considering implementation specific issues.

Keywords: evolutionary algorithms, civil engineering, design

3.2 Introduction

This section considers the topic of engineering design. The following section discusses how computers can be utilized to aid the design process specifically via decision support systems.

Design is a highly complex process that has been investigated via numerous theoretical and empirical studies e.g. Lawson, 1997: Dym, 1994: Pahl and Beitz, 1996. In spite of this, a definitive design methodology remains elusive. This is because "*…design is not a simple hierarchical process where the designer is presented with a set of requirements and works steadily through a decomposition strategy, moving from abstract concepts to the final concrete product. The design problem is ill-defined and changes as the designer explores it through solutions and partial solutions…*" (Hudson and Parmee, 1995). However design problems, regardless of discipline, are generally solved iteratively: by constantly proposing and refining solutions rather than by a purely sequential methodology, but it should be noted that design does not iterate around a single solution but rather around a range of acceptable solutions (particularly in multi-disciplinary projects). Finally, it must be acknowledged that as the design progresses every partial solution will influence the final solution. Therefore, each partial solution generates "*waves of consequences*" (Moran and Carroll, 1996), so decisions made during the early stages influence the later stages (of the design). Even without a definitive model of the design process, it is generally accepted that any design involves the following stages, whether a prescriptive (Finger and Dixon, 1989) or descriptive (Dym, 1994) methodology is used:

- ***Conceptual design:*** having determined a statement of need, the most important factor in conceptual design is the consideration of alternatives while developing a working solution (phrased at a high level).
- ***Embodiment design:*** "*…the part of the design process in which, starting from the working structure or concept of a technical project, the design is developed…to the point where subsequent detail design can lead directly to production…*" (Pahl and Bietz, 1996).
- ***Detailed design:*** the final stage where the embodied design is developed. This stage is almost procedural in nature and many algorithms have been created to aid designers.

This thesis will only consider the conceptual design stage because embodiment and detailed design have been extensively studied and are suited to classical/ procedural approaches. Conceptual design is characterised by the lack of information available to the designer however evolutionary algorithms are adept at searching such solution spaces.

3.2.1 Characteristics of civil engineering design

Civil engineering design problems generally involve the construction of bespoke artefacts, as conditions are rarely identical on different projects. However, traditionally designers typically start by looking at existing solutions of similar projects and adapting them to the current specification. So, while the solution is generally unique it is often based on a previous design and so exhibits common characteristics.

It should be noted that design is different to optimisation: optimisation generally involves manipulating defined variables to achieve an optimal solution; however in design, especially conceptual design, the problem is not fully defined at the outset. To solve the problem the designer proposes and refines solutions that also define the problem. To highlight these issues, Hudson and Parmee (1995) suggest that design problems contain three issues that are not present in optimisation:

- Neither the structure of the final solution nor the design space is fixed.
- The evaluation of concepts is not a simple quantitative comparison.
- A range of feasible solutions is more important than a single 'optimal' one.

However, it is acknowledged that the differentiation of design and optimisation is not clear. Rosenmann (1997) suggests a more general hypothesis that 'design' systems should be able to generate new solutions from random initial conditions using minimal heuristics.

3.2.2 Decision Support Systems for Conceptual Design

This section discusses the need for computer based Decision Support Systems (DSS) especially for civil engineering conceptual design, before the remainder of the chapter considers how solutions can be represented using a DSS based on evolutionary algorithms.

Decision Support Systems aim to expand the user's existing skills and experience by providing a problem solving methodology, which enables them to make better decisions (Miles and Moore, 1994). DSS achieve this by providing the following functionality (Turban, 1988):

- Allowing designers to quickly and objectively assess how their chosen solution responds if inputs or assumptions are changed.
- Providing a standardised framework for decision-making.
- Allowing all interested parties to participate in the design process, enabling everyone to develop a clearer understanding of the problem and possible solutions.
- Cost savings. Although contentious, a well-designed DSS should focus a design team on more viable solutions whereby reducing the chance of costly mistakes. It should also hasten the initial design process and thus reduce the overall cost.

Finally, a DSS can improve the final design by proposing a variety of ideas early in the design process. This is vital as Ullaman et al (1987) found that within 45 minutes of starting a design, designers have settled on their proposed solution and rather than consider alternatives they adapt it when problems arise. Therefore, by suggesting solutions without preconceived ideas and prejudices, a DSS should open the designer to more novel solutions (Sisk, 1999).

3.3 Representation

Evolutionary algorithms require candidate solutions to be evolved using operators based on biological evolution. Unfortunately most problems do not have solutions that are instantly amenable to these operators. Therefore they must be converted to a form that is. This involves developing an appropriate 'representation'.

In this work, 'representation' refers to the structure and encoding that allows potential solutions to be included in the search: some exclude the encoding methodology from the representation while others include the fitness function. The primary purpose of a representation is to convert every possible solution to a form that allows it to be included in the search.

The canonical evolutionary algorithms use a variety of representations:

- ***Evolutionary Programming (EP):*** Finite state machines.
- ***Evolutionary Strategies (ES):*** Real-valued vectors.
- ***Genetic Algorithms (GA):*** String representation (with binary encoding).
- ***Genetic Programming (GP):*** Representation based on tree, graph or linear structure.

The following discussion does not consider implementation specific issues but focuses on how structures can be represented (including the advantages and disadvantages of every approach). However it should be noted that most representations discussed are commonly associated with either the GA or the GP. This is because the EP and the ES are generally used as continuous parameter optimisation tools and are therefore not particularly suited to conceptual design. Also this thesis considers labelling evolutionary algorithms as GA or GP etc as potentially misleading because it encourages people to apply their preconceived ideas rather than focusing on what is being described.

3.4 String Representation

This section considers string representations. For the purpose of this thesis, 'strings' are one-dimensional structures that do not allow cycles and in general contain a sequence of parameters.

String representations are often appropriate for parametric problems or when discrete elements are required. Strings are composed of a series of variables (in some instances variable ordering is important). In any case, there are three ways to encode a string: binary, integer and real (although a single string may include several encodings).

3.4.1 Binary-encoded string

A binary-encoded string (Figure 3-1) is often the most natural representation for Boolean variables. As binary-encoded strings formed the initial GA representation used by Holland (1975), they have become synonymous with GAs. Unfortunately, this means that they are often used irrespective of suitability. However, they do provide the most schemata[7] per bit of information of any encoding and may be extended to encode integer and real numbered variables.

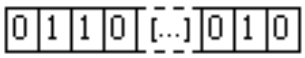


Figure 3-1 Example binary encoded string representation

Within civil engineering examples of binary encoded strings include (Table 3-1):

Table 3-1 Binary encoded strings in civil engineering design

Application	Year	Author
Optimum composite laminate design	2000	Matous et al.
Reinforced concrete biaxial column design	1998	Rafiq and Southcombe
Building layout	1999	Park and Grierson
Truss design	1995	Shrestha and Ghaboussi

3.4.2 Integer-encoded string

Integer-encoded strings are often the most appropriate representation for a finite set of discrete variables or integer based variables (Figure 3-2). For example, the diameter of steel reinforcement bars. It should be noted that an integer-based variable could be converted to a binary bit string, which will provide more schemata per bit of information. However, retaining the integer encoding ensures that two genes will remain close in both the solution and representation spaces and reduce the string's overall length.

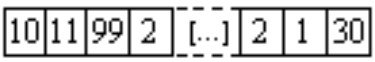


Figure 3-2 Example integer encoded string representation

[7] A sub-region of the representation space created by including an additional 'don't care' character '#' in the representation's encoding (Holland, 1975)

Within civil engineering examples of integer encoded strings include (Table 3-2).

Table 3-2 Integer encoded string representation examples in civil engineering design

Application	Year	Author
Structural building design	2003	Sisk et al.

3.4.3 Real-encoded string

A real-encoded string (Figure 3-3) is often the most appropriate representation for continuous or high precision variables e.g. the length of a structural beam. It should be noted that as with integer variables, a real-based variable could be converted to a binary bit string. However, the disadvantage of converting to a binary representation is the level of precision must be specified in advance. Therefore the string can become exceptionally long if a large quantity of high precision variables is required.

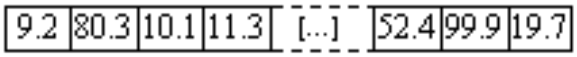


Figure 3-3 Example real encoded string representation

Within civil engineering examples of real encoded strings include (Table 3-3).

Table 3-3 Example applications of real encoded string representation in civil engineering design

Application	Year	Author
Design of reinforced concrete beams	1997	Coello et al.

3.5 Voxel Representation

This section describes ‘voxel’ representations, which are often appropriate for shape discovery problems because they decompose the solution space into discrete elements (usually square or triangular in shape) called ‘voxels’ (volume pixels). Once the solution space is decomposed, each voxel is allocated a Boolean value. If the value is true, then the voxel is considered to contain some material, and if false the voxel is empty. Therefore, this representation allows two-dimensional structures to be mapped to a binary string.

Unfortunately, because adjacent voxels are not guaranteed to remain adjacent in the genome, a disadvantage of this representation is that it is prone to “*…the development of*

small holes, isolated voxels and jagged edges […] and eliminating these deficiencies without having to apply strong guidance using heuristics poses a significant challenge…" (Griffiths and Miles, 2003). These issues can be mitigated by post-processing solutions or utilizing intelligent evolutionary operators (Zhang and Miles, 2004). A final disadvantage of voxels is the 'fineness' of the voxel grid must be determined at the outset, which significantly biases the final solution. However they are very well suited to modelling structures such as I beams.

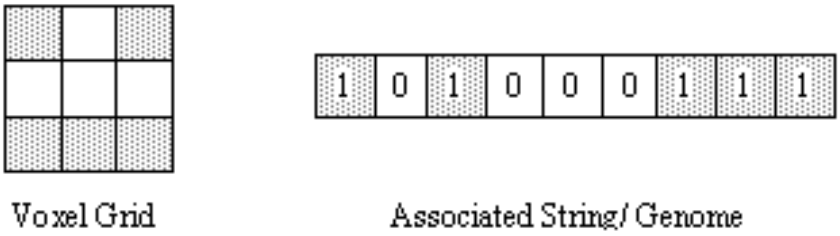


Figure 3-4 Example voxel representation

Within civil engineering examples of voxel representations include (Table 3-4).

Table 3-4 Applications of voxel representation in civil engineering design

Application	Year	Author
Optimisation of I beam cross section	1999	Baron et al.
Optimisation of I beam cross section (including shear stress)	2003	Griffiths and Miles
Optimisation of I beam cross section (including shear stress)	2004	Zhang and Miles

3.6 Tree Representation

Trees are a non-linear, hierarchical and strictly acyclical data structures constructed from nodes (Figure 3-5). Every tree starts with a 'root' node, at depth 0. The root node is unique because it does not have a parent, but it does have children[8]. Each child forms a separate sub-branch and maybe a parent for other nodes. Any node that does not have a child is called a 'leaf'. Leaf nodes generally contain inputs. The remaining nodes are 'functions'. Function nodes process leaf inputs and transfer the result to their parent.

[8] As with genealogical trees, tree representations use familial terminology when referring to other nodes

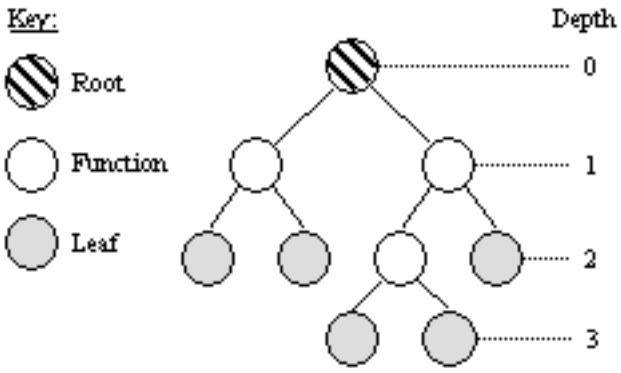


Figure 3-5 Example binary tree representation

Theoretically, every node has an arbitrary number of children. However trees are often designed with a predetermined number of children. For example, every binary tree node has a maximum of two children (Figure 3-5).

As previously stated, trees are hierarchical and strictly acyclical. Therefore, a child cannot have a higher depth that its parent (Figure 3-6).

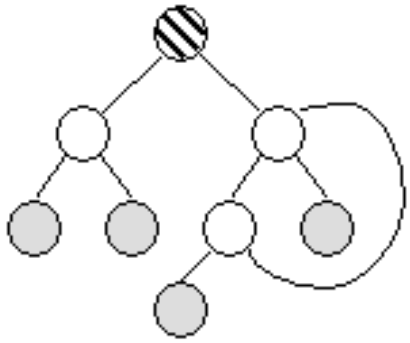


Figure 3-6 Invalid tree representation

3.6.1 Yang and Soh's (2002) tree representation

Within civil engineering design, only one set of authors has published papers incorporating a tree representation**:** Yang and Soh. This section discusses their representation while the following section discusses some of the issues related to using a tree representation (as proposed by Yang and Soh). The representation they propose incorporates a binary tree with two types of node:

- ***Function nodes:*** representing cross-sectional areas of the members A_p (p= i,j,k,l,m,n).
- ***Leaf nodes:*** representing structural joints N_i (i= 1,2,3,4).

To decode which two nodes a member spans the tree is parsed by starting at the relevant node and progressing down the connection lines until a terminal node is reached. For example member A_l spans nodes N_4 and N_3 (Figure 3-7).

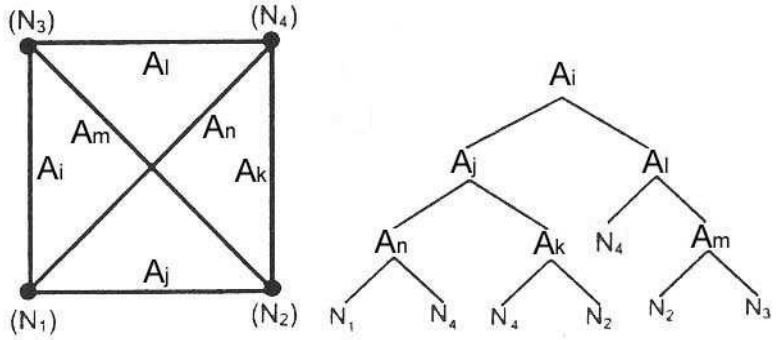


Figure 3-7 Tree representation for structural design

3.6.2 Advantages of a tree representation

Tree encoding appears very simple, when compared to the equivalent binary string e.g. when designing a truss capable of supporting six loads, the tree representation required 29 nodes (16 joint and 15 members) where as Shrestha and Ghaboussi's (1998) string representation required 25,200 bits. However this comparison is slightly unfair because tree nodes encapsulate[9] data, while the string representation does not.

3.6.3 Disadvantages of a tree representation

During evolution, especially recombination, tree representations have a tendency to develop problems: consider the following crossover (Figure 3-8) between two identical parents encoding a six-member truss:

[9] Process by which an object 'hides' data and provides methods to access it (in object-orientated programming).

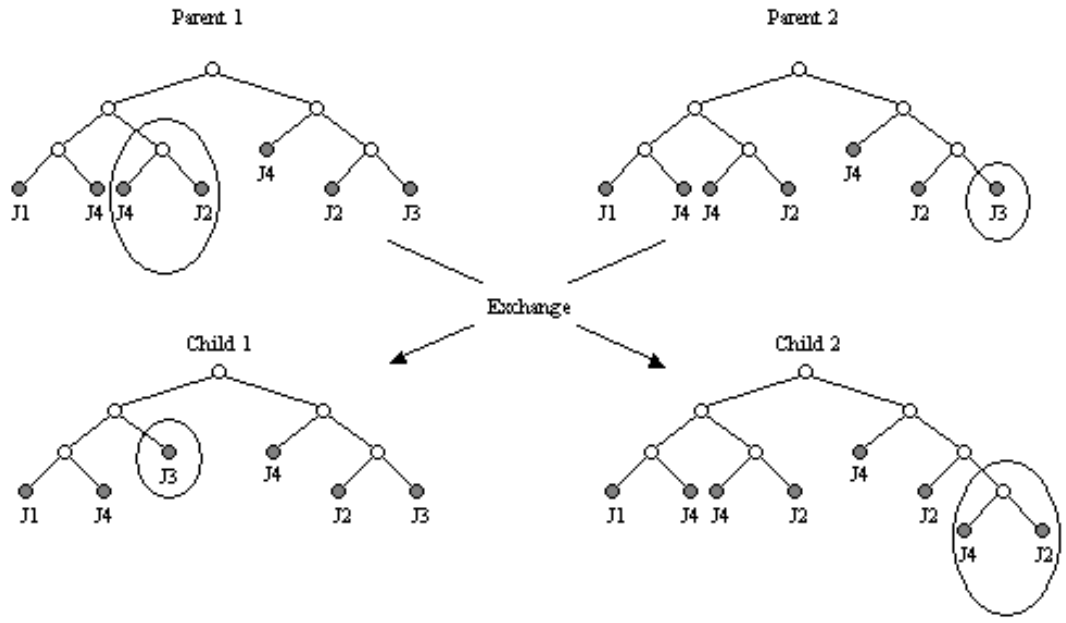


Figure 3-8 Example recombination operation between identical parents

There are three problems with these offspring:

- The ideal '1-to-1' mapping can only be assumed during initialisation, as it can degenerate during evolution (Figure 3-9). Therefore, unless the evolutionary operators are restricted or individuals are repaired, evolution will produce a 'n-to-1' mapping with all its repercussions.

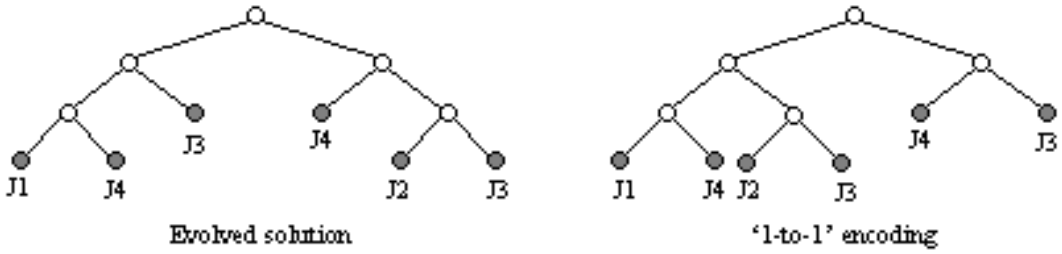


Figure 3-9 Degeneration of '1-to-1' mapping

- Evolution may produce members that span between the same joint (a null member) or create several copies of the same member (Figure 3-10). While often not fatal to the structure, it does add a computational overhead and causes the solutions to 'bloat'.

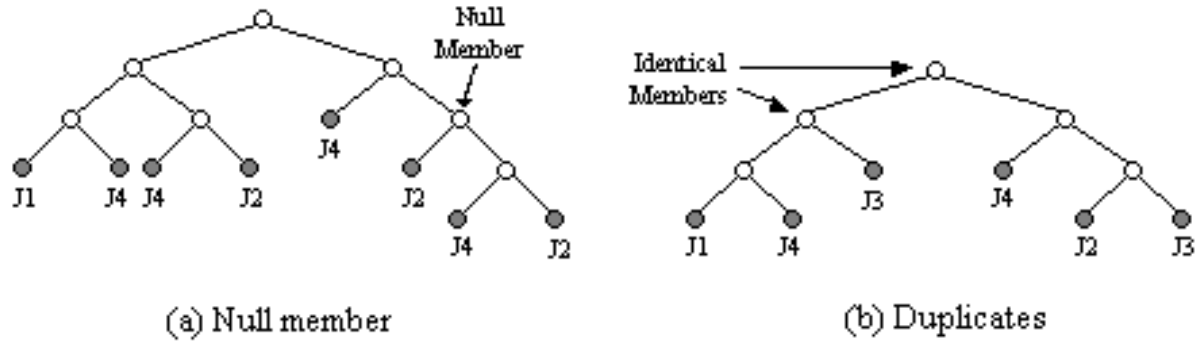


Figure 3-10 Problems after evolution for tree representation

3.7 Graph representation

Graphs are a non-linear data structure composed of nodes connected by edges. However unlike trees, graphs allow cycles and can incorporate loops and recursive commands. This is because in addition to performing a function, graph nodes determine which node will be executed next.

Graphs are often a good representation for skeletal structures e.g. trusses, because they support the adaptability required for topological design. For example strings are linear structures, therefore each element has at most two connections: left and right. Unfortunately, most physical structures contain elements that connect to an arbitrary number of elements. Therefore, a higher dimensional representation maybe required having a more appropriate form.

Graphs are often used for modelling problems in civil engineering, within design only one paper has been published: Borkowski et al (2002). The representation proposed by Borkowski et al (2002) involves two elements:

- ***Composition graphs (CP-graphs)***: A directed labelled graph (Figure 3-11) representing a structure's topological features (its genotype). CP-graphs are composed of nodes (representing joints) and edges connecting two nodes (representing members) both of which are labelled and attributed.
- ***Realisation schemes***: A mapping that assigns properties to the CP-graph to generate the phenotype.

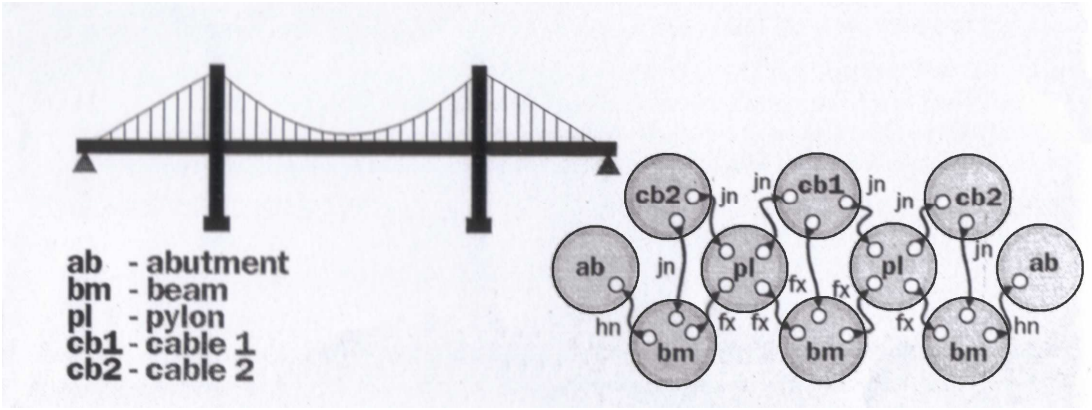


Figure 3-11 Example bridge and CP-graph representation (adapted from Borkowski and Grabska, 1995)

In addition to this, Borkowski et al (2002) suggest that a physical structure is created by a finite number of topologically identical units that they call 'panels'. For each panel, a CP-graph in evolved. This reduces the representation space's size.

3.8 Other Representations

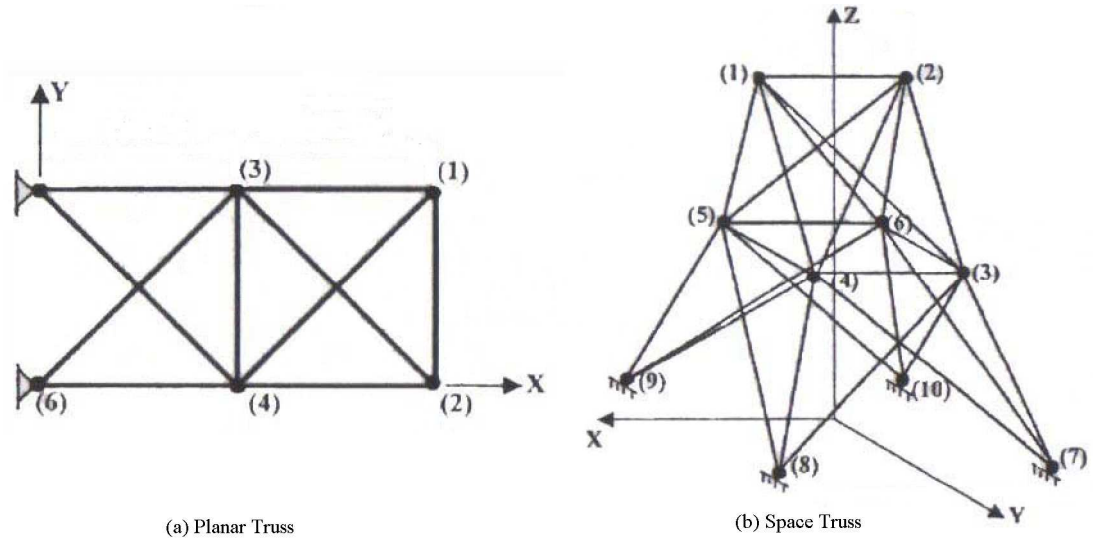
This chapter has covered the most common representations, however others do exist including:

- ***Homogenisation:*** The material (from which the structure is constructed) is considered to be 'sponge-like' containing an infinite number of micro-cells and voids, which can be assigned different densities (Bendsoe and Kikuchi, 1988).
- ***Voronoi-based:*** The structure is composed of a finite number of voronoi sites that define a voronoi diagram (Kane and Schoenauer, 1996).
- ***Shape Grammars:*** This method is often used for layout design. Shape grammars perform computations with shapes in two steps: recognition of a particular shape and possible replacements (Stiny and Gips, 1972).

3.9 Representation and truss design

This section provides an introduction to trusses before reviewing the existing approaches to truss optimisation and design. Trusses have been selected because they are the most commonly studied type of structure for civil engineering design problems. Therefore there are a number of approaches to compare and contrast.

Structural trusses Figure 3-12 are composed of at least two members (in tension or compression), which when joined together create a stable construction in either two dimensions (planar truss) or three dimensions (space truss). Trusses are a common engineering structure often used to support roofs or bridges.



(a) Planar Truss (b) Space Truss

Figure 3-12 Example planar and space truss

Structural optimisation and design problems frequently use trusses "*…this maybe attributed to the fact that trusses usually possess many nodes and elements that can be deleted or retained without affecting the functional requirements. In addition, the truss is a relatively simple, yet non trivial structure…*" (Kirsch, 1990).

3.9.1 Truss optimisation versus design

Truss optimisation involves modifying an existing design so that it is more efficient. This usually involves reducing its weight whilst ensuring it remains fit for purpose and has been a research topic since Mitchell's seminal paper in 1904 (Mitchell, 1904). When optimising a truss, there are three variables to consider:

- ***Sizing***: Modifying the size of structural members.
- ***Geometry***: Modifying the position of structural nodes.
- ***Topology***: Modifying the number and connectivity of structural members.

Most existing approaches consider a truss' sizing, geometry and topology to be independent and solve them in turn. However, sizing, geometry and topology are obviously not independent because the initial modifications will constrain those that come after. Nevertheless this approach is frequently adopted as it makes the problem more accessible.

Topological optimisation is the most difficult process to investigate because the representation must incorporate a mechanism by which member connectivity can be modelled (Deb, 2002) and this factor limits the applicability of classical/ procedural approaches. As if to highlight this, some approaches even neglect topology and concentrate on optimising sizing and geometry. Evolutionary algorithms, and in particular genetic algorithms, with their adaptive representations are more suited to this type of problem and many optimisation papers suggest using this approach Table 3-5, but all utilise a 'ground structure' first proposed by Dorn et al (1964).

Table 3-5 Topological optimisation via genetic algorithms

Author	Year
Ruy et al.	2001
Deb and Gulati	2001
Camp et al.	1998
Rajeev and Krishnamoorthy	1997
Hajela and Lee	1995
Rajan	1995

Ground structures contain a large number of highly connected nodes (Figure 3-13). To optimise the topology, an algorithm removes all non-essential members (although it is arguable that because topology is predetermined, optimisation only occurs within a limited search space). This can be accomplished by associating an extra 'flag' gene, with each member in the genome indicating whether the member is present or not. To add or remove a member, the algorithm changes it flag status. Unfortunately, this produces long genomes containing large quantities of redundant information. Therefore, the final topology is biased by the ground structure. However, this approach does simplify the issue of representation because each genome contains every possible member configuration (even if the genome is excessively long).

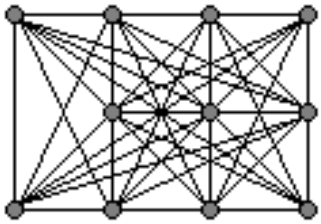


Figure 3-13 Example ground structure

Truss design is a more difficult problem than optimisation, because there is no initial structure to adapt. Therefore, for it to be effective, the design algorithm must generate at least one potential solution and modify its sizing, geometry and topology simultaneously without the need to rely on a ground structure.

The major issue with topological design (of trusses) is how to represent the 'node element diagrams' of structural analysis and in particular that a member spans between two joints (in addition to its own properties). As topological design is a difficult subject and there are only three major representations to date, all will now be reviewed.

3.9.2 Shrestha and Ghaboussi (1998)

Shrestha and Ghaboussi suggest a solution based on a fixed length, string representation, by encoding individual joints and duplicating member information. Each string genome is composed of a fixed number of sub-strings (Figure 3-15), which encode joint locations using Cartesian coordinates. In addition to this, the space around each joint is discretised into 8 regions (Figure 3-14).

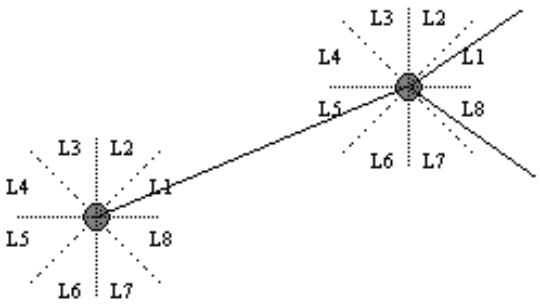


Figure 3-14 Sectorial joint representation (adapted from Shrestha and Ghaboussi 1998)

When a member is associated with a joint, the relevant joint region encodes its properties. However, because a member spans between two nodes, it can potentially have two different

sets of properties (one maintained by each region). To decide which properties to use nodes are assigned priorities with the dominant node defining the member.

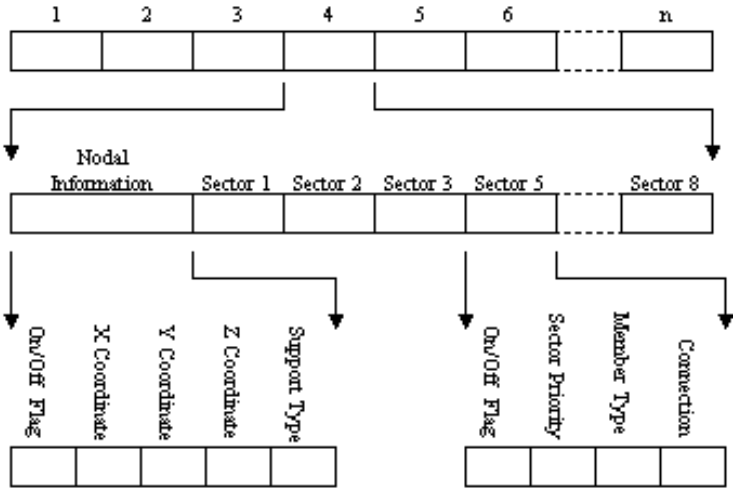


Figure 3-15 String representation (adapted from Shrestha and Ghaboussi 1998)

This indicates some of the deficiencies of a 1D string representation: because it lacks a suitable structure, topology must be encoded in addition to the geometry and sizing information and this arbitrary representation (of topology) creates redundant information in the genome increasing its size.

3.9.3 Yang and Soh (2002)

Yang and Soh suggest a solution based upon a 2D adaptive tree structure, by encoding members and duplicating joint information. They propose that the tree should compose two types of node (Figure 3-16):

- ***Leaf nodes***: representing structural joints.
- ***Inner nodes***: representing structural members.

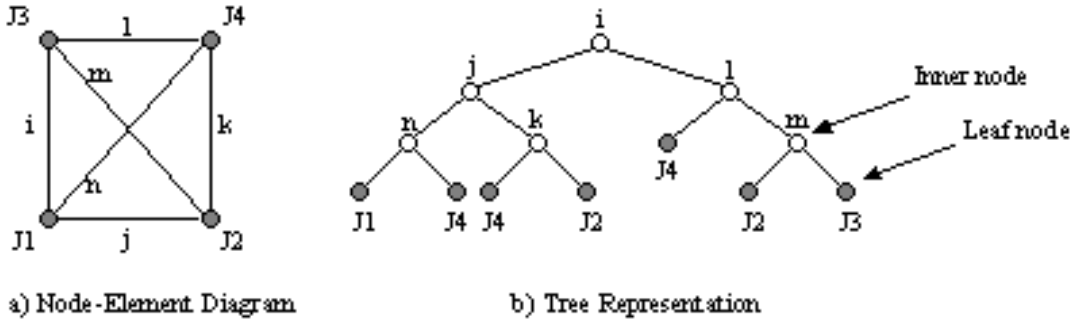


Figure 3-16 Six member truss and tree representation

They also recognise that this encoding methodology only provides a ‘n-to-1’ mapping, which means that the same truss can be represented by several different tree configurations (Figure 3-17).

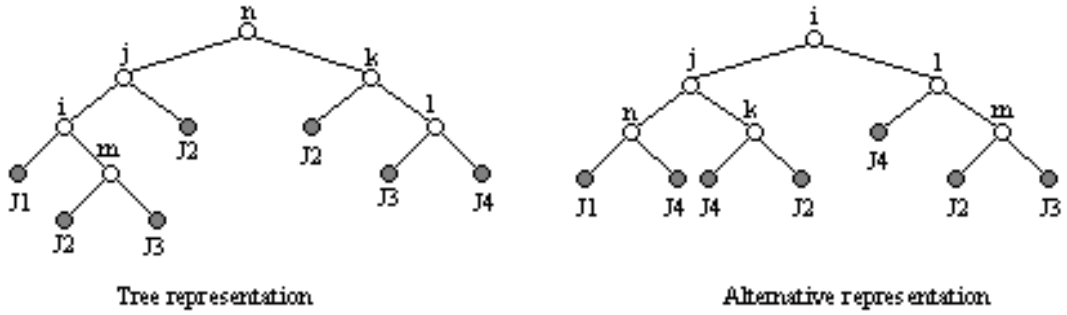


Figure 3-17 ‘n-to1’ mapping

While valid, ‘n-to-1’ mapping enlarges the solution space reducing an algorithm’s efficiency. Therefore Soh and Yang suggest an improvement to produce a ‘1-to-1’ mapping: joints and members are numbered and (without loss of generality) the lower numbered element considered first. To encode a truss, the following procedure is applied:

> *“…The lowest numbered member is selected to be the root node. This member then has its start and end joints represented by children nodes to the left and right respectively. Then, from left to right, the lowest numbered member associated with each joint is removed from the structure and inserted into the tree. This procedure continues until every member is represented in the tree…”*

It is important that the left-right relationship of offspring and parent be maintained as the tree is constructed, because the nodes to its far left and far right define every member. For example, member *i* spans between joints J_1 and J_3 (Figure 3-16). For more information regarding issues with tree representations please refer to 3.6.3.

The following paragraph is slightly esoteric, but interesting nevertheless! Soh and Yang consider that using a tree structure indicates the use of genetic programming (Koza, 1992). However, because the phenotype (the truss) has a different structure to the genotype (the tree) an additional decoding step must be incorporated into the solution procedure. Therefore, the solutions are not ‘active structures’. It is this author’s opinion that if this work is to be pigeon holed into one of the four canonical EAs their work should be considered to be a genetic

algorithm (Holland, 1975). However it is only by labelling their approach as genetic programming the authors have left themselves vulnerable to this sort of criticism. It is for this reason that this thesis will refer to any experimental work as an evolutionary algorithm using a particular representation.

3.9.4 Azid and Kwan (1999)

Azid and Kwan published an approach that allows the evolutionary operators, of a GA based system, to act directly on the phenotype rather than its genotypic representation. However they must use some form of representation (as defined in this thesis), as it is impossible to implement any computer based technique without some form of representation. Therefore because trusses naturally form graphs it is assumed that they used a graph-based representation. They also highlight the problem of using a coded string: the evolutionary operators are highly disruptive. To mitigate this, several rules are used to ensure that any offspring mimic their parents (to prevent too many infeasible solutions being generated):

- Any offspring formed by two structurally viable parents must be structurally viable i.e. not a series of discontinuous joints and bars in space.
- There must be some visual architectural resemblance between offspring and parent.

3.10 Conclusions

Conceptual design is the first stage in a highly complex process. To aid the designer, decision support system based on evolutionary algorithms maybe used because although conceptual design is characterised by the lack of information available to the designer, EAs are adept at exploring fragmented and complex search spaces. However EAs require candidate solutions to be converted to a form that is amenable to evolutionary operators. Many representations have been designed each with its own strengths and weaknesses: strings are generally used for parameters, voxels for shape discovery, trees and graphs for skeletal structures. Within civil engineering design, the most commonly studied structure is the truss and three main representations have been used, each with their own pros and cons.

4 Conceptual Layout Design of Orthogonal Commercial Buildings

4.1 Abstract

The conceptual layout design of commercial office buildings is a non-trivial task because the numerous design variables create a large solution space. To aid designers, several decision support systems have been developed. However, all these systems are limited to buildings with rectangular floor plans.

This chapter presents a evolutionary algorithm based methodology capable of designing buildings with orthogonal boundaries and atria. To achieve this the floor plan is partitioned into rectangular sections using a sweep line algorithm and to prevent unrealistic solutions being generated, the representation (a 3-section string) ensures the initialisation and evolutionary operations are not too disruptive. The number of initial inputs has also been reduced, because this work is aimed at the conceptual design stage. Therefore the user only needs to specify the external boundaries shape and location of any atria.

The aim of this chapter is to investigate existing examples and develop new representation for orthogonal building layout design.

Keywords: commercial office buildings, conceptual layout design, evolutionary algorithm, polygon partitioning, orthogonal boundary.

4.2 Introduction

Conceptual design commences once a problem has been identified and a vague description of a solution has been formulated (usually in functional terms) called the ‘project brief’. Generally, the aim of conceptual design is to generate a range of solutions that will be further developed during the subsequent design stages. Therefore although these solutions are based on limited information, they will determine most of the major design parameters. In fact it is often quoted that by the end of the conceptual design stage 70 to 80 percent of a project's resources are committed.

Conceptual design is also considered to be one of the most difficult challenges facing practising designers because of the range of possible options. For example, it is estimated that for a typical commercial building of 20 stories, even if one only considers the architectural and structural aspects, there are approximately 170 million possible design options

(Khajehpour and Grierson, 2003). Therefore only experienced engineers carry out conceptual design tasks, because the lack of initial information limits the effectiveness of procedural techniques to assist more junior designers.

4.3 Related Work

In order to aid building designers, various papers (Table 4-1) have proposed Decision Support Systems (DSS) based on evolutionary algorithms. Evolutionary algorithms are suited to this role because they are adept at exploring fragmented and complex search spaces. However, all these systems are limited to buildings with rectangular floor plans.

Table 4-1 DSS for the conceptual design of buildings

Author	Year	Method	Details
Harty and Danaher	1994	Knowledge Based System (KBS)	Produces realistic designs in structural steel and reinforced concrete for regularly shaped buildings
Grew	1995	KBS	Uses simple calculations and rules of thumb (can reuse knowledge gained from existing structures) for the design of portal framed buildings.
Fenves et al.	1995	Case Based Reasoning (CBR)	Part of the SEED system (Software Environment to Support the Early Phases in Building Design) that is user extensible.
Fuyama et al.	1997	KBS	Based on behaviour considerations and first principles this system, implemented in an object orientated programming environment, designs moment resisting steel frames.
Rajeev and Krishnamoorthy	1998	GA (String)	Design optimisation of reinforced concrete plane frames using a genetic algorithm (taking into account factors related to detailing and placement of reinforcement).
Khajehpour and Grierson	1999	GA (String)	Conceptual design of medium-rise office buildings using a multi-criteria genetic algorithm in conjunction with pareto optimisation theory.
Rafiq et al.	1999	GA (String)	Design of concrete framed buildings using a genetic algorithm incorporating a neural network for a floor plan based on regular column spacings.
Soibelman et al.	2000	CBR + GA	Structural design of tall buildings by proving designers with adapted past design solutions generated by a distributed multi-reasoning mechanism.
Miles et al.	2001	GA (String)	Design of commercial office buildings using a genetic algorithm as a search engine to determine layouts with regular and irregular column spacings.
Grierson and Khajehpour	2002	GA (String)	Cost-revenue conceptual design of high-rise buildings using a multi-criteria genetic algorithm.
Eisfeld and Scherer	2003	KBS + Descriptive Logic Reasoning	Interactive planning algorithm using an expressive description logic language to represent structural knowledge acquired from practitioners.
Sahab et al.	2005	Hybrid GA (String)	Two stage conceptual design of reinforced, concrete flat slab buildings.

4.3.1 BGRID

The work published by Miles et al. (2001) called 'BGRID' will now be discussed in more depth, because this section of the thesis is a continuation of it. BGRID was developed in close collaboration with practising engineers and focuses on the design of rectangular floor plans, using a genetic algorithm (Holland, 1975) to generate column layouts. To achieve this, BGRID concentrates on a number of 'first order' design decisions:

- Dimensions of the structural, constructional, servicing and planning grids.
- Environmental strategy (for both lighting and ventilation).
- Floor-to-ceiling height including (spacing requirements for services).
- Structural depth and its impact on the building height.
- Cost

However the search within BGRID is heavily constrained, as the user is required to fix their preferred dimensions for the modular and structural grids at the start. The GA is also allowed to modify the overall building and atria dimensions to fit a potential grid. By heavily constraining the search and modifying the outline, BGRID is able to carry out a near exhaustive search of the feasible options. Unfortunately the final solutions are often only marginally better than the initial, random solutions. This lack of improvement could be due to the fact that the best solutions tend to lie on the boundary between the feasible and infeasible regions. Thus by not allowing the search to explore the infeasible region the algorithm's search is restricted. It is also a reflection on the heuristics applied during initialisation, which ensures the population is only seeded with viable options.

After its development, BGRID was assessed by about 80 practising designers including architects, building services engineers and structural engineers and 68% of them suggested that this type of tool could be useful.

4.4 OBGRID

This section provides an introduction to the OBGRID (Orthogonal Building GRID) a DSS for the conceptual design of orthogonal buildings by considering some of the key issues. OBGRID is a continuation of BGRID however it must be stressed that the aim of this work is to develop a suitable representation (capable of handling non-rectangular floor plans) rather

than a complete building design system. Therefore the fitness function and evolutionary operators used are to demonstrate the representation's flexibility rather than to optimise performance. The following sections describe how OBGRID designs rectangular and orthogonal floor plans.

4.4.1 Column Layout

One of the most important features of commercial buildings is that columns should preferably be arranged in rectangular grids. This is not to say that other arrangements are not used, but regular rectangular grids tend to be easier and more economical to construct and provide a flexible layout that can be readily adapted during the life of the structure.

4.4.2 Structural Systems

At present OBGRID contains the information for three structural spanning systems: short, medium and long (however the system is user extensible). As stated previously, the aim of this work is to develop a suitable representation for orthogonal buildings. Therefore BGRID's structural systems have been incorporated into OBGRID.

- ***Short***: Slimflor™ has an integrated steel deck (minimising the depth of the structural zone). [Economic range = 5-8m].
- ***Medium***: Composite steel beam and composite slab system. [Economic range 6-12m].
- ***Long***: Steel stub girder and composite slab system. [Economic range 18-20m].

As larger column spacing generally produce a more flexible internal environment OBGRID tends to favour longer spans, which is admittedly biases the search.

4.4.3 Environmental Strategy (Ventilation)

Ensuring the correct ventilation is a fundamental problem in building design because it is difficult to change once built. Three environmental strategies have been considered (although others maybe added by the user):

- ***Natural ventilation***: Natural ventilation is provided by the glazing system, but usually only available in non-urban environments.
- ***Mechanical ventilation***: If the building is too deep for natural ventilation then mechanical ventilation maybe suitable.

- ***Air conditioning***: In an urban environment this is often the only option as it allows the building to effectively maintain a self-contained environment.

4.4.4 Services Integration

The electrical, communication and ventilation services must be coupled with the structural system in one of three ways:

- ***Separate***: The services and structural system are accommodated in adjacent zones. This approach is characterised by short spans and a shallow construction depth.
- ***Partial***: If the structural system is deep enough, some of the services maybe accommodated within it. However, some services must be routed under the primary beams and thus out of the structural zone.
- ***Full***: The services and structural system are accommodated in the same zone. This approach is often characterised by long spans with a deep construction depth (within cillular beams).

4.4.5 Clear floor-to-ceiling height

The clear floor-to-ceiling represents the usable 'office' space. A high floor-to-ceiling height is required if the client requires natural daylight and natural ventilation. It is suggested that this should be between 2.4m => 4.0m with a recommended minimum of 2.7m.

4.4.6 Floor-to-floor height

To calculate the floor-to-floor height, the floor-to-ceiling height is added to the distance required for the floor spanning system and services (Table 4-2).

Table 4-2 Dimensional allowances for services

		Environmental Strategy		
		Air Conditioning (mm)	**Mechanical (mm)**	**Natural (mm)**
Services Integration	**Separate**	900	635	350
	Partial	650	500	325
	Full	350	350	350

4.4.7 Initial User Input

Because this work is aimed at the conceptual design stage, the number of input variables has been reduced. The user is only required to enter the dimensions of the boundary and atria in addition to specifying the total number of storeys. Other GA based DSS allow the algorithm to search for the optimum number of storeys e.g. Khajehpour and Grierson (1999) and Rafiq et al. (1999). However, during BGRID's evaluation it was suggested that the client usually fixes this parameter at the outset therefore this option has been omitted (if the designer wishes they can re-run the algorithm with different numbers of floors to investigate this variable).

4.5 OBGRID and Rectangular Buildings

This section contains a detailed description of how OBGRID handles rectangular buildings. Layout design of rectangular floor plans is fundamental in this work, because every orthogonal floor plan will be partitioned into rectangles.

4.5.1 Representation

In an efficient building layout, columns should be aligned in straight rows. Therefore, the representation should be robust enough to reflect this feature even after the disruption caused by the evolutionary operators.

Initially an attempt was made to include individual column locations in the genome using a tree or graph structure (Figure 4-1a). However, this representation proved to be slightly unstable and tended to leave some columns isolated in the floor plan particularly after evolution (Figure 4-1b). This is because by focusing on individual columns, these representations failed to incorporate the idea of rows. So if one column's location was altered, the algorithm was unable to update the remaining columns.

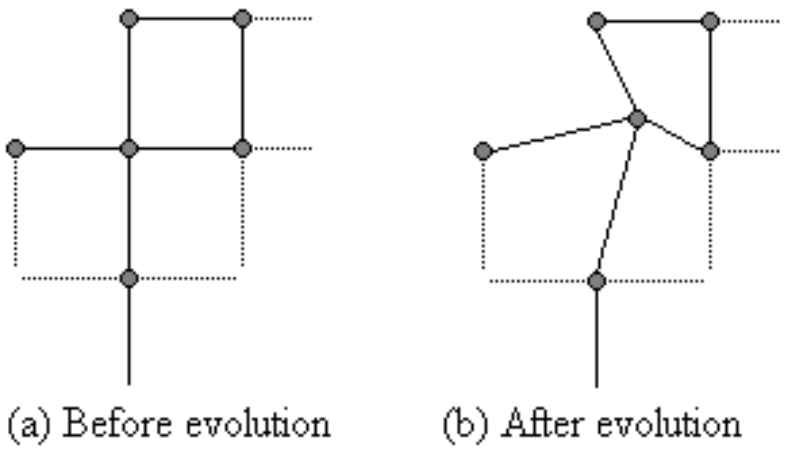


Figure 4-1 Problem using tree or graph based representation in layout design

OBGRID uses a 3-section string representation (Figure 4-2) that focuses on aligning column rows by considering a column's x and y coordinates independently, so a gene references a row of columns rather than an individual one. It should also be noted that the number of columns included in sections 1 and 2 is not fixed (and can vary during the search) thus this representation is a variable length genome.

- ***Section 1:*** contains column x spacing.
- ***Section 2:*** contains column y spacing.
- ***Section 3:*** contains the remainder of the information including: structural system, services integration, environmental strategies and the floor to ceiling height.

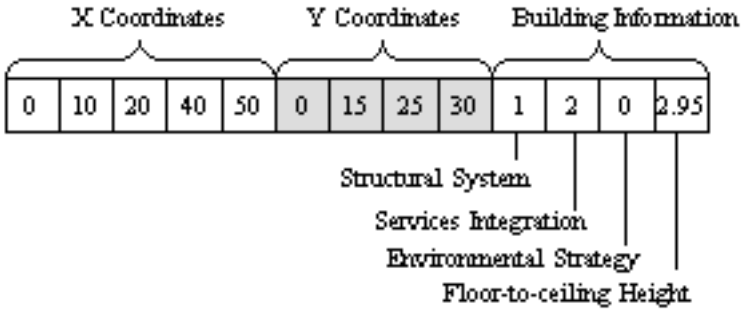


Figure 4-2 Example genome for layout design

Sections 1 and 2 of the genome contain values that always increase from left to right. This ordering is maintained because it ensures a 1-to-1 mapping between the representation and solution spaces. Span length, the distance between columns, is calculated by finding the difference between adjacent genes (as genes signify column locations).

Each gene, in section 1 or 2, references a row of structural columns rather than just a single column. Therefore any change to an individual gene will not invalidate the layout because the whole row will be altered (see 4.5.3).

4.5.2 Initialising the genome for a rectangular floor plan

The following section will describe how the genome for a rectangular floor plan is initialised. To aid understanding, an example floor plan of 50m x 30m will be initialised. Each section of the genome is considered in turn:

- ***Section 1:*** starting at the upper left hand corner of the floor plan (it is always assumed that the top left hand corner has the local coordinates (0,0)) the algorithm generates random column spacings in the x direction until the end of the floor plan is reached.

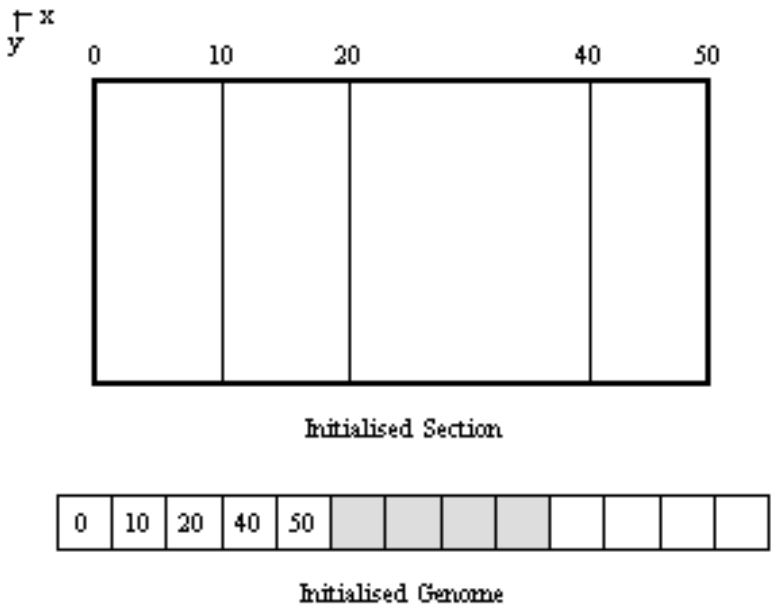


Figure 4-3 Rectangular floor plan (Section 1 Initialised)

- ***Section 2:*** restarting at the upper left hand corner (0,0), the algorithm generates random column spacings in the y direction until the end of the floor plan is reached.

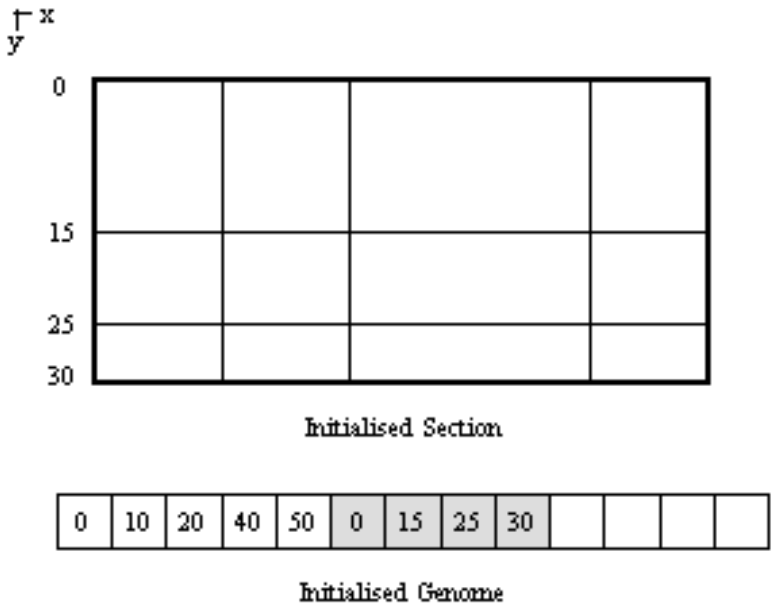


Figure 4-4 Rectangular floor plan (Section 2 Initialised)

- ***Section 3:*** The final section is initialised with randomly selected genes from the appropriate gene set. For example the basic structural system gene set contains three elements: 0 = Short, 1= Medium, 2 = Long, so this gene will either be a 0, 1 or 2.

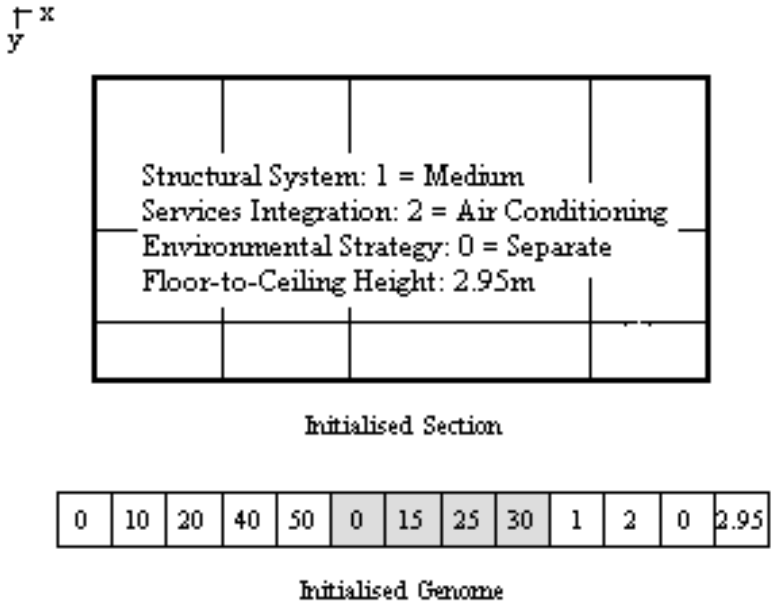


Figure 4-5 Rectangular floor plan (Section 3 Initialised)

It should be stated that unlike BGRID no effort is made to constrain column positions to 'realistic' spacing i.e. within the economical range for the selected spanning system. This is to encourage the algorithm to for solutions in both the feasible and infeasible regions. However,

the fitness function does penalise individuals that contain a wide range of column spacing. This is to encourage a degree of uniformity in column spacing, which aids 'buildability' without adding much bias.

4.5.3 Evolutionary Operators

Evolutionary algorithms search the solution space by using biologically inspired operators. However because the genome is divided into 3 distinct sections of variable length, the evolutionary operators have been amended to reflect this:

- ***Mutation:*** used to inject new solutions into the population improving the search by (hopefully) preventing premature convergence (Goldberg, 1989). Having selected an individual's genome, a new value is generated for a random gene. If the mutation operator selects a gene from sections 1 or 2 then it is replaced with a randomly generated value between 0 and the limits of the floor plan. Unlike BGRID that restricts the new spacing to a value between the two adjacent genes, OBGRID simply generates a random spacing and when it's needed sorts the genome so that the column spacing increase from left to right[10]. If a gene from section 3 is selected a random gene from the appropriate gene set is used.

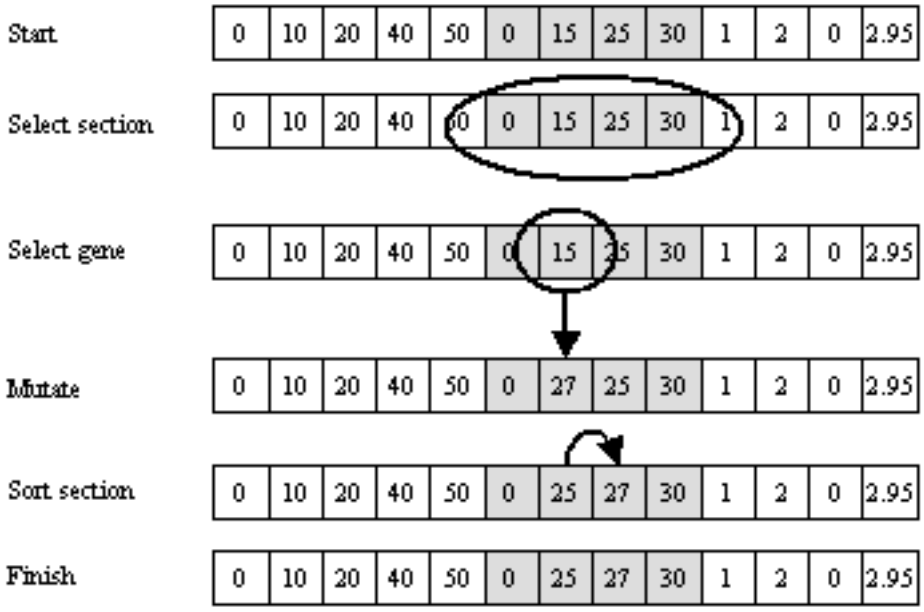


Figure 4-6 Example mutation operation

[10] The sorting algorithm is that contained in Java's native java.utils package: a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n log(n) performance.

- ***Recombination:*** used to exploit the information already in the population. OBGRID employs a single point crossover operator. Single point crossover is used because it is simple to implement even with variable length genomes (as in OBGRID). However rather than applying the crossover operator on the whole genome, it performs a separate crossover on each of the genome's three sections. Although for section 3, the cut point is always located at the same point to ensure this section of the genome remains of constant length.

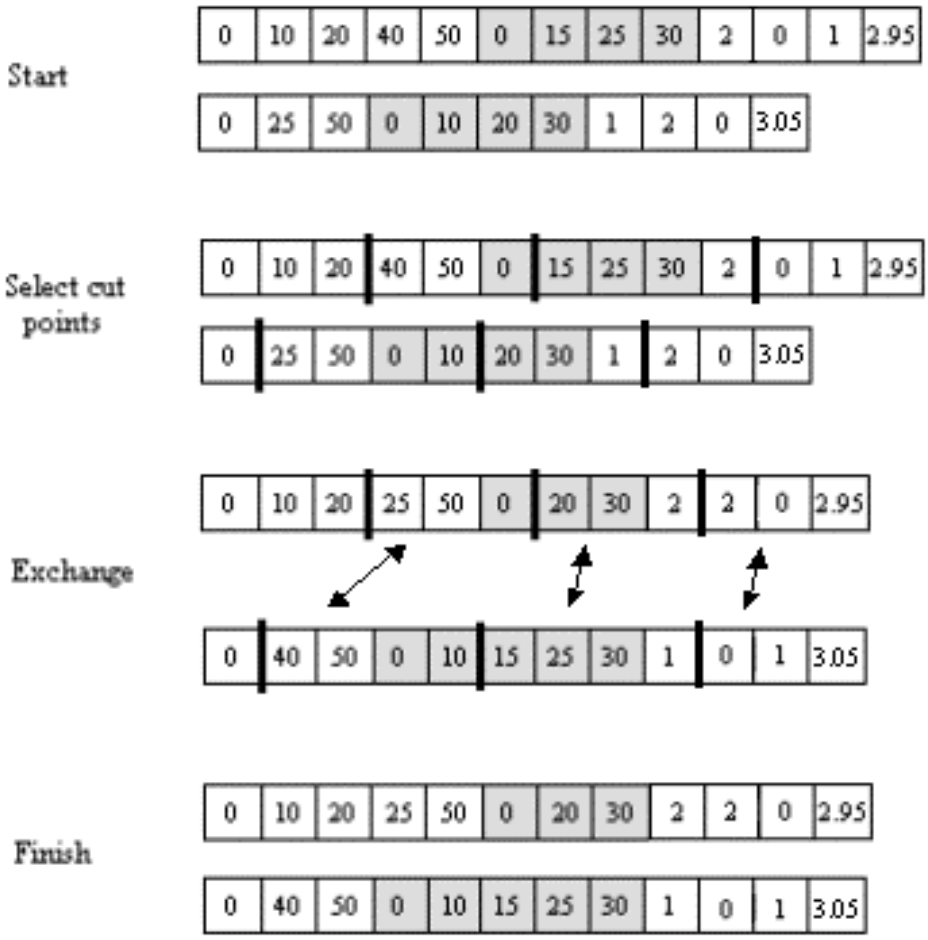


Figure 4-7 Example recombination operator

4.5.4 Selection

BGRID originally used the standard fitness method (Bradshaw and Miles, 1997) to select individuals during evolution. This technique ranks the individuals by raw fitness and then assigns a predetermined fitness to every individual according to their rank. However, OBGRID has replaced the standard fitness method with the more conventional tournament selection technique (Goldberg, 1989) to improve search performance.

4.5.5 Fitness function

The fitness function assigns a single numerical value to an individual reflecting how 'good' it is. A multi-objective fitness function might be more appropriate for this work however the goal was to develop a representation capable of handling orthogonal layouts rather than a complete building system.

OBGRID is a minimisation algorithm, which means that the optimum solution has a fitness of 0. This is because floor plans are assigned an initial fitness of 0 but during evaluation are penalised if they break the predetermined criteria. Therefore a layout with 0 fitness is not penalised and thus should be a very 'good' solution. OBGRID uses a penalty function because although this can be a conservative approach, convergence delay was considered to be less dangerous than the premature loss of material: as the optimum will typically be located on the boundary between the feasible and infeasible regions and this approach allows the EA to search from both directions. Although there are many types of penalty function OBGRID uses a quadratic penalty function, which assigns a greater penalty to a larger transgression.

OBGRID has three components to its fitness function but it is acknowledged that other factors could be added. However the following components are included to test the representation's performance using relatively 'realistic' criteria:

- ***Overall height***: The solution's overall height must not exceed the value stipulated by the user. If the solution is larger it is penalised by the penalty function.
- ***Column spacing compatibility***: Column spacing must be compatible with the economical span distance of the structural spanning system. For example, if the structural system is 'short' (specified by the first gene in section 3), the span distances should be between 5 and 8m (4.4.2 Structural Systems).
- ***Uniformity of the grid***: OBGRID attempts to evolve solutions based on regular column spacing, so the standard deviation of the spacings is used in the fitness calculation. With a lower standard deviation being preferable (indicating greater spacing uniformity).

4.5.6 Running the algorithm

There are potentially two ways to run this algorithm: with the algorithm able to vary the flooring system during a run or by preventing the algorithm varying the flooring system and

re-running for each flooring system. It was decided to use the second approach because this will typically evolve a good solution for each flooring system rather than ignoring it. This can be important in the 'real world', as practising engineers are typically sceptical of 'black box' solutions that ignore their criteria and might wish to view a particular spanning system (although OBGRID will indicate a solution's suitability via its fitness).

4.6 Illustrative Example: Rectangular Building

This section provides an illustrative example of OBGRID designing a rectangular building. The parameters in the EA tableau (Table 4-3) should be considered indicative because the aim of this work is to develop a new representation rather than a complete building design system.

4.6.1 Introduction

The following test case was designed to assess OBGRID's performance. Unfortunately, unlike structural optimisation, there are not standard test cases. This is possibly because there is no such thing as a standard building because they are multi-disciplinary structures (unlike trusses) therefore the following test case was used:

- Building dimensions: 60m x 18 m
- Height restriction: none.

Table 4-3 EA Tableau for Rectangular Building

Objective	Evolve example layout designs for a rectangular boundary of 60m x 18m with no height restrictions)
Representation	3-section string
Initialisation	Random initialisation (no seeding)
Raw Fitness	Based on: column spacing compatibility and column spacing uniformity
Selection	Tournament (size = 2)
Major Parameters*	P = 1, M = 100, G = 50
Evolutionary Operators:	
$\text{Reproduction}_{prob}$	0.1
Mutation operator	Point
Mutation_{prob}	0.3
Recombination operator	One point crossover
$\text{Recombination}_{prob}$	0.6

*P = Number of populations M = Population size G = Max number of generations

Some researchers may question why the probability of mutation is so high by comparison to a typical GA. There are two reasons for this:

- The algorithm used in this these is described as an evolutionary algorithm rather than a pure genetic algorithm, for example. Therefore by considering it as such, the researcher has a tendency to apply preconceived ideas, which may or may not be appropriate.
- The mutation operator is mechanistically very similar to that used for recombination. Therefore the algorithm is less sensitive to changes in these probabilities than other implementations. However the mutation operator has the potential to introduce a gene pattern not already found in the population, while recombination simply exchanges existing gene patterns between individuals.

4.6.2 Results

The following graphs show the best, mean and worst fitness recorded during an indicative run for the medium structural spanning system. It is important to note that because OBGRID is a minimisation algorithm a lower the fitness indicates a better solution.

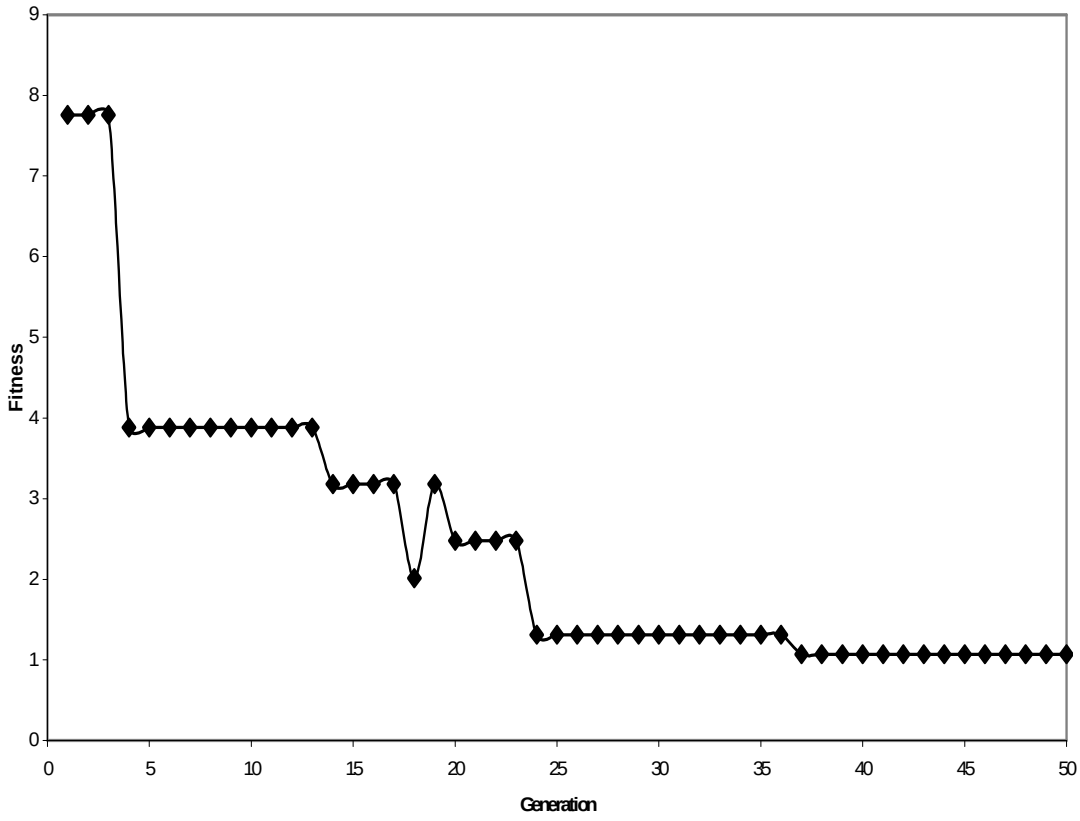


Figure 4-8 Best fitness

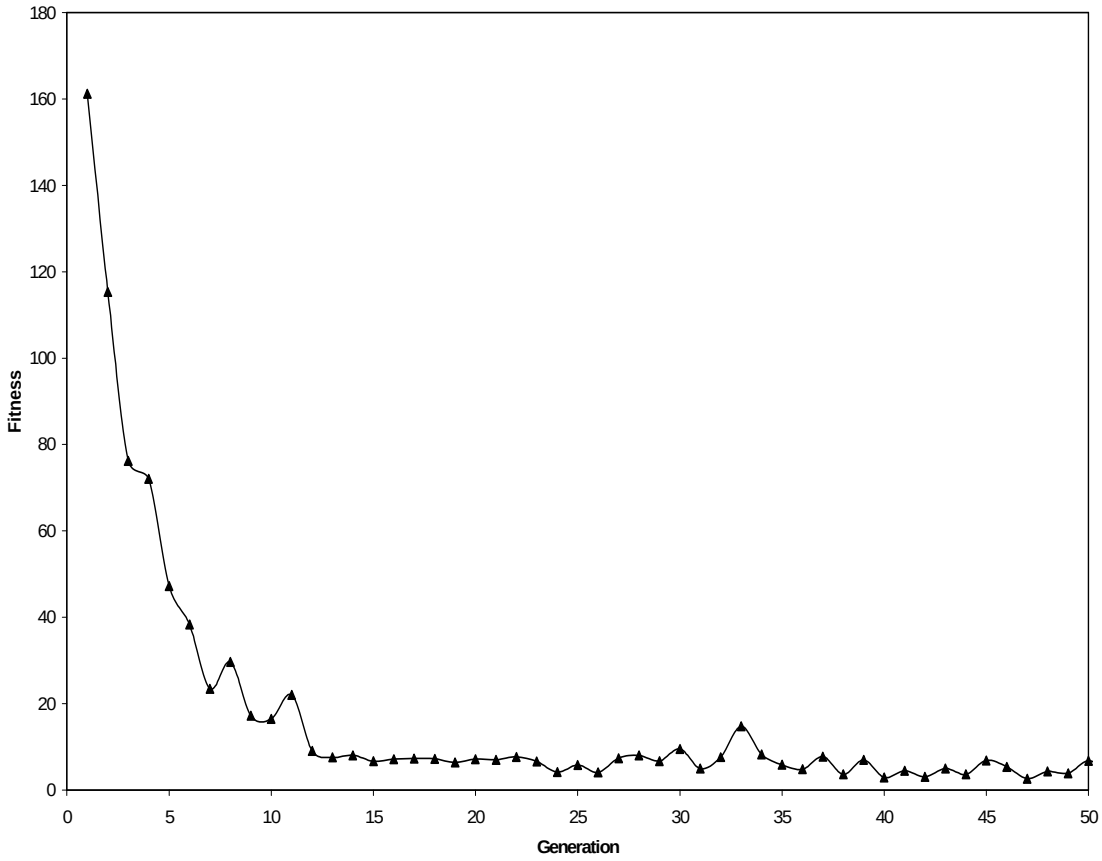


Figure 4-9 Average fitness

The best and average graphs trend downwards during the run. This indicates that the algorithm is converging towards the ‘optimum’ (although the algorithm is not guaranteed to locate it). The spread also narrows between the average and best, which suggests that the by employing fitness-based selection, the algorithm is encouraging the ‘better’ characteristics to propagate. However the same cannot be said for the worst fitness.

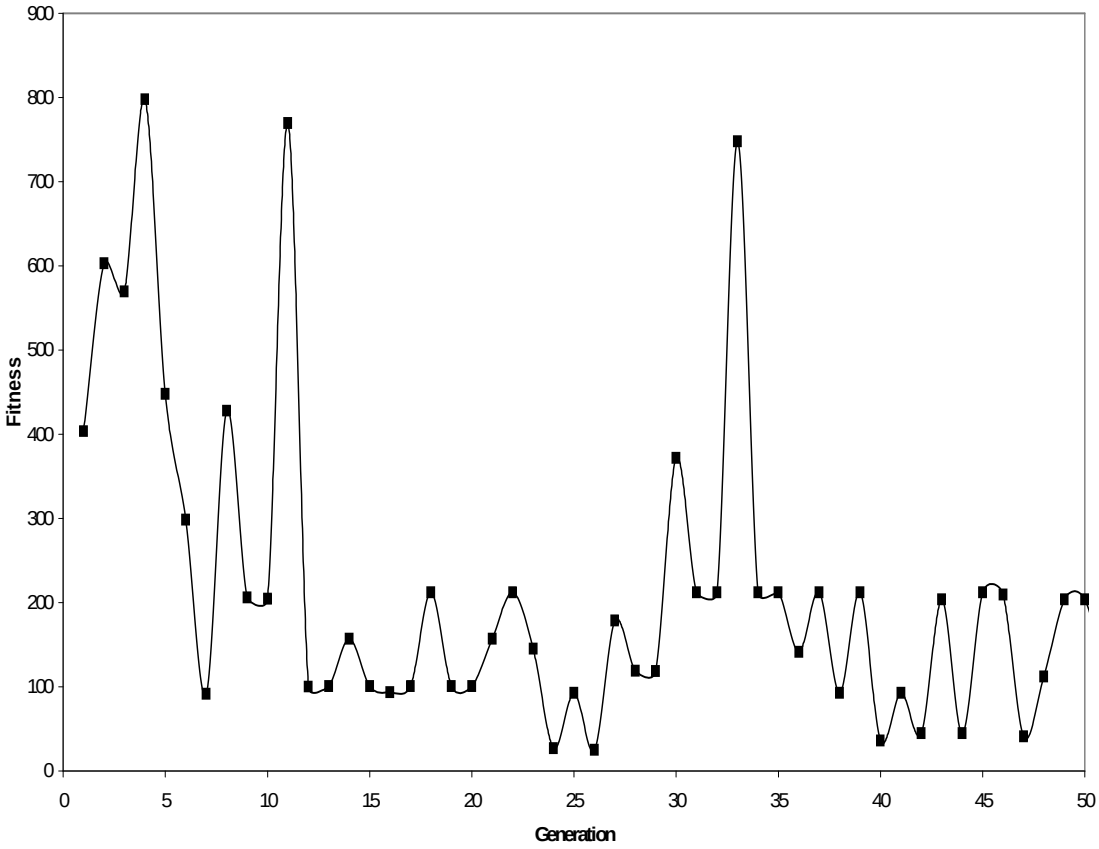


Figure 4-10 Worst fitness

Although the worst fitness 'stabilises' in the range of 50 –300 after generation 10, it never reaches equilibrium and often 'spikes' e.g. generation 33. But again, this is to be expected. The evolutionary operators are potentially very disruptive and an individual's fitness may actually be reduced afterwards. However this is why evolutionary algorithms are so powerful: although evolution may produce a harmful result for an individual it may also produce a beneficial change, which maybe be propagated throughout the whole population. This is shown in Figure 4-8 and Figure 4-9. During the run, the average fitness trends downwards in a fairly smooth manner, whilst the best proceeds in discrete steps. This is because the best individuals are formed by chance therefore they can be a huge improvement over their ancestors (this feature is especially prominent at the beginning of a run). However, as stated above, once the improvement has been found, it often spreads through the population reducing the overall fitness in a more gradual manner.

There is one final feature of Figure 4-8 worthy of mention: because the best solution is not explicitly copied into the next generation i.e. elitism is not used, the best fitness can rise between generations. For example, between generations 18 and 19 the best solution actually

decreases in fitness. But this merely demonstrates how robust evolutionary algorithms can be. Even though the algorithm has lost its best solution to date, it quickly recovers and by generation 24 has found an even better solution. Ignoring the best of generation can also help to prevent premature convergence. For example, if the algorithm is forced include the best solution to date, but this solution is simply a local optimum, then the algorithm would be hindered rather than helped. So by ignoring the effects of evolution on an individual, for example after recombination, the algorithm is free to search using all the information contained in the population and if the best solution to date is the global optimum, hopefully it will return to it!

Figure 4-11 shows the solutions returned when the algorithm is run for each structural spanning system.

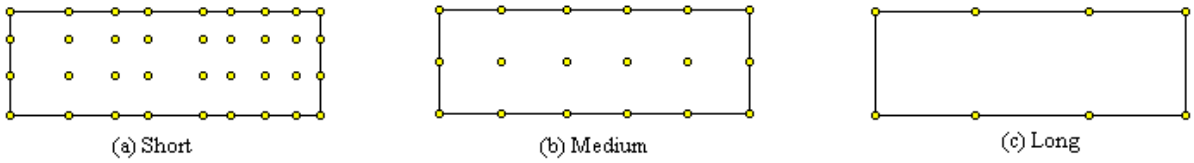


Figure 4-11 Returned solutions for rectangular building example

The final average spacings are all within the economic ranges and were as follows: short x = 7.5m y = 6m; medium x = 12m y =9m; long x = 20m y = 18m. However these averages are slightly misleading, as the column spacings returned are not necessarily uniform. In particular, as the number of columns increases, OBGRID finds it harder to retain regular spacings. Having OBGRID search explicitly for the number of rows per partition, rather than for column spacings could rectify this. However this would represent a much simpler challenge and thus was not pursued for this thesis.

4.6.3 Conclusion

Although simple, this example indicates how OBGRID solves rectangular building layouts. This is a fundamental process in this work because orthogonal layouts are decomposed into rectangular sections that are solved in this manner.

4.7 OBGRID and Orthogonal Buildings

This section contains a detailed description of how OBGRID handles orthogonal[11] buildings: by partitioning them into rectangles and using the previously described methodology to design a layout for each partition. This process is a novel feature of OBGRID that has not, so far as the author is aware, been previously used in building layout design systems and is an improvement over all existing examples that are limited to rectangular floor plans. To ensure column row continuation throughout the building an 'adjacency graph' is used.

4.7.1 Representation

OBGRID partitions an orthogonal floor plan into rectangles, using the sweep line algorithm described in 4.7.3, and associates a genome with each partition. Therefore each individual (representing an orthogonal boundary) contains a set of genomes rather than a single genome as per a rectangular floor plan (see Figure 4-12). However section 3 is considered to be standard for all genomes, as it refers to attributes applicable to the whole building rather than simply one area.

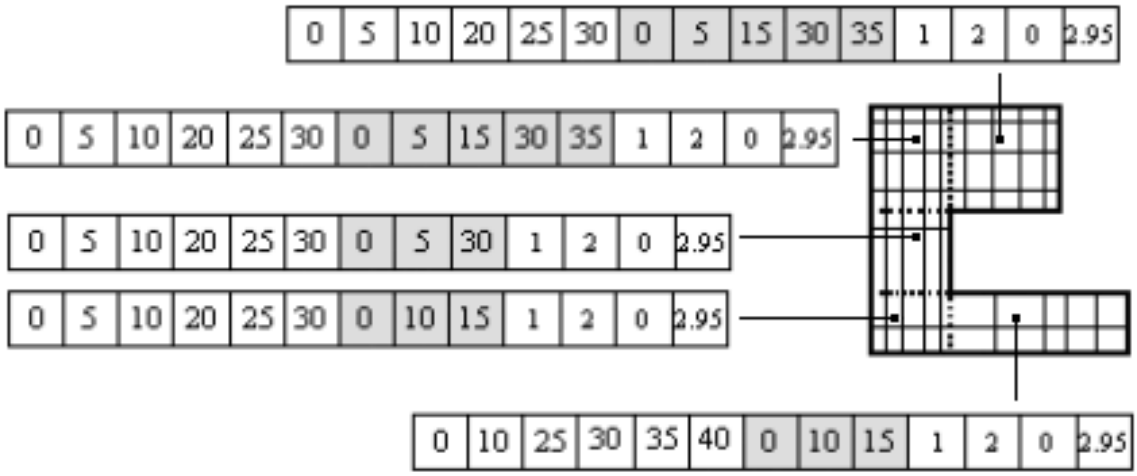


Figure 4-12 Example orthogonal representation

To ensure column line continuity throughout the floor plan each partition is linked to its neighbours via the adjacency graph (see 4.7.4). For the remainder of this section, the same 'C' shaped floor plan will be used as an example.

[11] A layout involving right angles.

4.7.2 Polygon Partitioning

Computational geometry (Shamos, 1978) is the study of efficient algorithms (usually computer based) and data structures for solving geometric problems. The partitioning of polygons is a major topic in this field and several algorithms have been developed. However a 'sweep line' approach was considered the most appropriate for column layout design because of the need to ensure column line continuation throughout the building (this issue will be discussed later).

4.7.3 Sweep Line Partitioning Algorithm

Sweep lines algorithms (O'Rourke, 1998) move an imaginary line, the 'sweep line', over a polygon from top to bottom or left to right. At predetermined points the sweep line is stopped and the polygon partitioned. These points are called 'event points'. In this work when partitioning orthogonal layouts without atria, event points are any reflex[12] vertex on the boundary (see Figure 4-13).

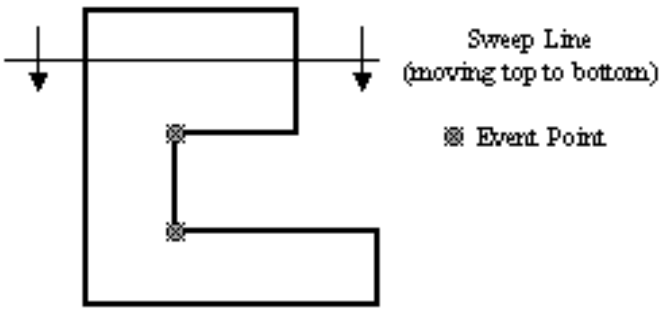


Figure 4-13 An example sweep line

Partitioning is completed in two stages:

- ***First stage***: a line is swept from top to bottom. When the line encounters an event point it extends the boundary edge horizontally across the floor plan until it encounters another edge. The encountered edge is then split at the point of intersection, which partitions the building into several, 'thin' rectangles. For example in Figure 4-14a edges *a* and *b* have been extended to edge *c*.

[12] A reflex vertex has an internal angle strictly greater than π.

- ***Second stage:*** a line is swept from left to right across the boundary, further partitioning the rectangles created by the first stage. This creates the final grid pattern. For example in Figure 4-14b edge *z* has been extended to split edges *x* and *y*.

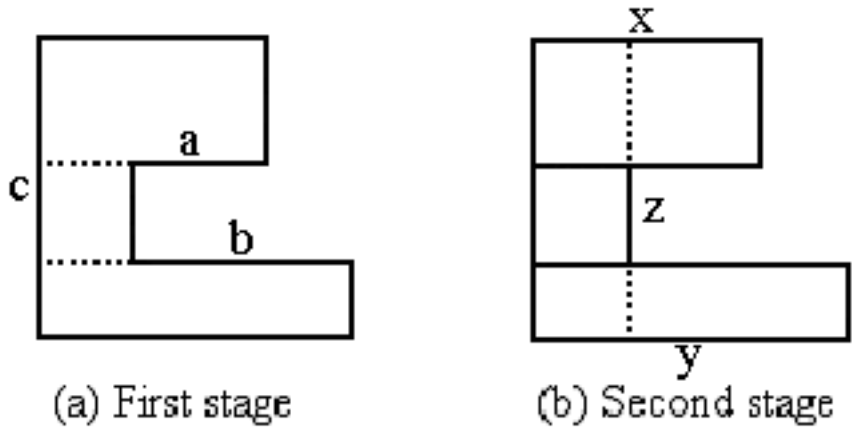


Figure 4-14 Example partitioning of orthogonal layout

It should be noted that for each floor plan, there is a unique partitioning. Therefore once it has been partitioned, no further partitioning is required during the search.

In terms of originality, as far as the author is aware, this is the first time a sweep line algorithm has been applied to building layout design. However, sweep line algorithms are commonly used in pure mathematics especially topology.

4.7.4 Adjacency Graph

This section describes the 'adjacency graph' a data structure that is used to ensure column line continuity throughout the building, which as far as the author is aware, is unique to this work.

With the floor plan decomposed into a grid of rectangles, via the sweep line algorithm, each partition must now share at least one edge with another partition (with an upper limit of four). The adjacency graph links partitions which share an adjacent edge and is used to repair individuals during initialisation or after evolution, reducing the potential for generating nonsensical solutions.

The adjacency graph is created from nodes, with each rectangular partition having a node associated with it (see Figure 4-15a). The nodes of adjacent partitions are then linked. For example in Figure 4-16b, node *a* is linked to nodes *b* and *c* but not directly to *d* because they do not share an adjacent edge. However during initialisation and evolution, any updates are applied recursively therefore changes *a*'s genome will be reflected in partition *d* too. Having

linked all adjacent partitions, the adjacency graph is complete. But how does the adjacency graph help maintain column line continuity?

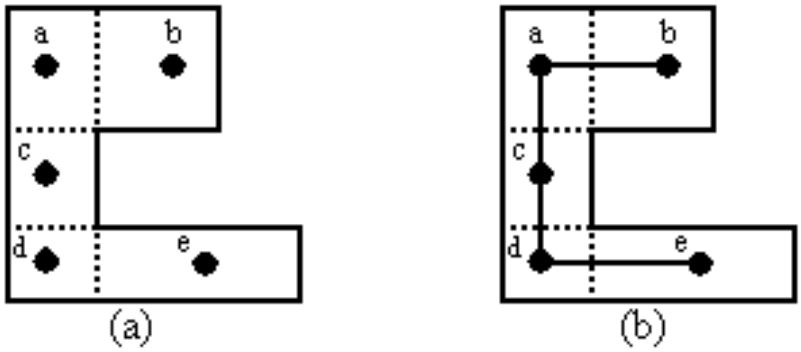


Figure 4-15Example adjacency graph of an orthogonal layout

When a partition's genome is modified, either during initialisation or evolution, it updates the corresponding section of its neighbour's genome. For example if any changes are made to the x coordinates (section 1) of partition *a*, then section 1 of partition *c* will also be updated (partition *d* will also be updated by *c*). However section 1 of partition *b* is unaffected because it does not share an edge in the x direction (they share one in the y direction). This is shown in Figure 4-16 where the column row at 10m in the x direction is deleted from partition *d* and the adjacency graph is used to ensure this gene is deleted from the genome's of partitions *a* and *c*.

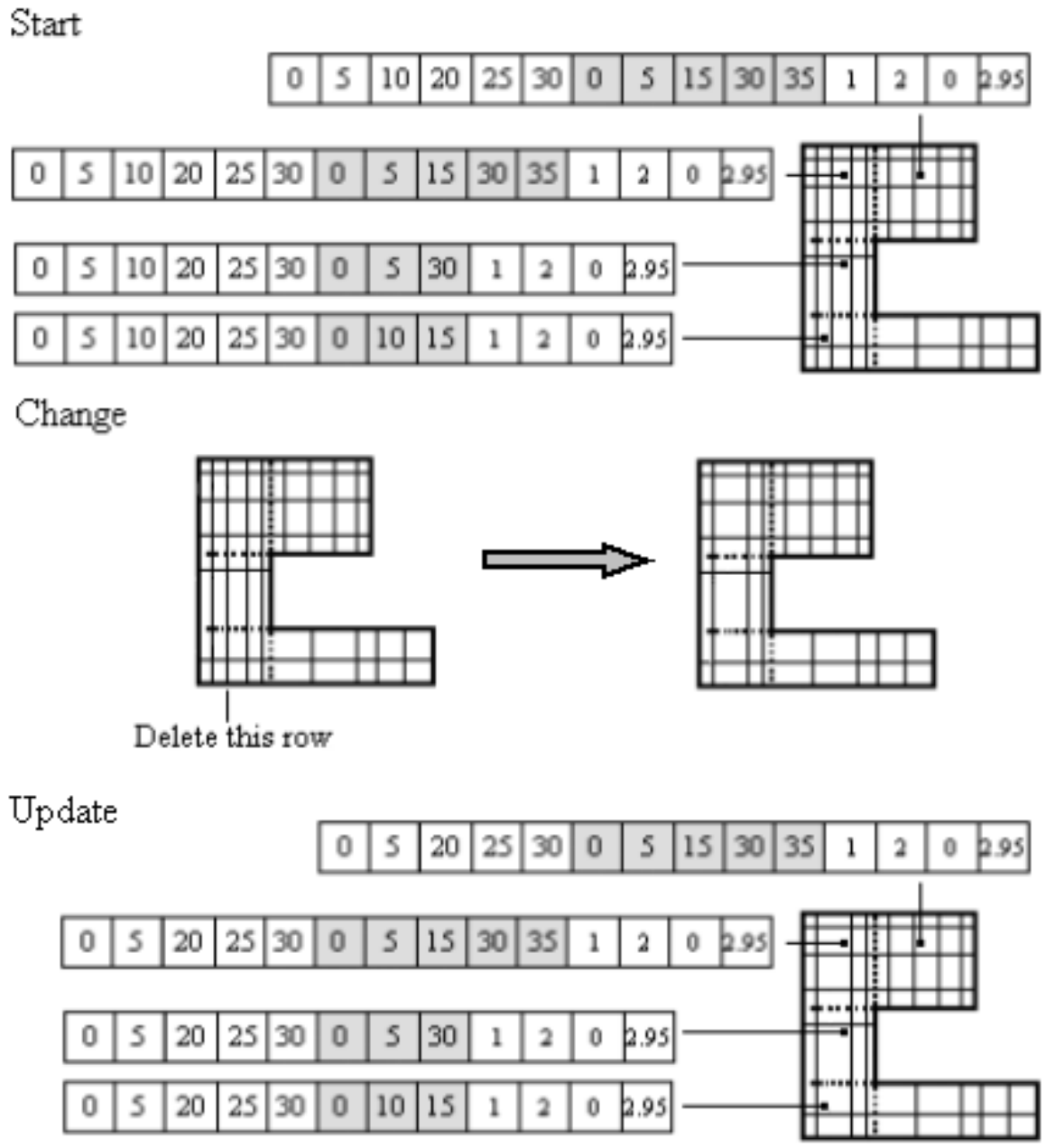


Figure 4-16 Example genome update using the adjacency graph

This example also demonstrates why this sweep line algorithm was developed as it has been, because it ensures that adjacent edges are always of the same size. For example, some sweep line algorithms are used to solve the 'least ink problem' where the goal is to partition an orthogonal polygon using the smallest number of partitions, in terms of length. This problem is illustrated in Figure 4-17a with the least ink solution shown in Figure 4-17b. However the adjacent edge *x* (between partitions *a* and *b*) is smaller than the left edge of partition *b* (see Figure 4-17c) and thus it would be much more complicated to ensure column line continuity during initialisation and evolution.

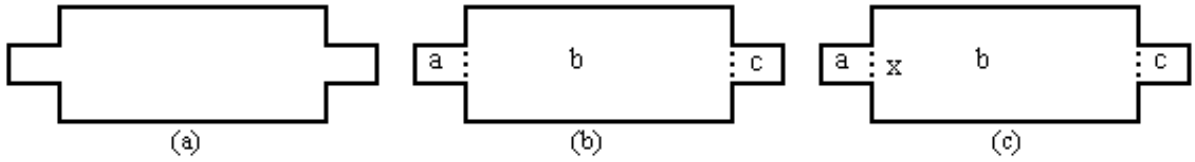


Figure 4-17 Least ink problem

4.7.5 An Alternative Partitioning Algorithm

Dr Rafiq of Plymouth University proposed the following partitioning and representation during a discussion about this work. The proposed methodology indiscriminately extends all edges across the floor plan (see Figure 4-18), allowing it to be expressed by a single genome rather than multiple genomes are proposed by this thesis.

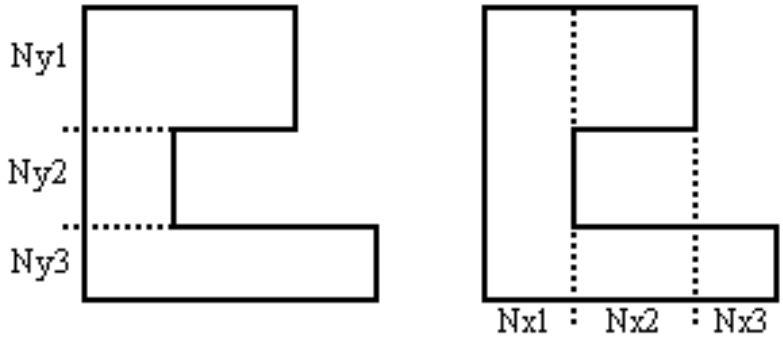


Figure 4-18 Dr Rafiq's partitioning

Unfortunately by extending edges across the whole floor plan, it has a tendency to generate superfluous partitions (not generated by this thesis' technique) and thus bias the search towards shorter column spacings creating a less flexible layout. For example in Figure 4-19 Dr Rafiq's technique generates 24 partitions (see Figure 4-19a) while this technique generates 17 (see Figure 4-19b).

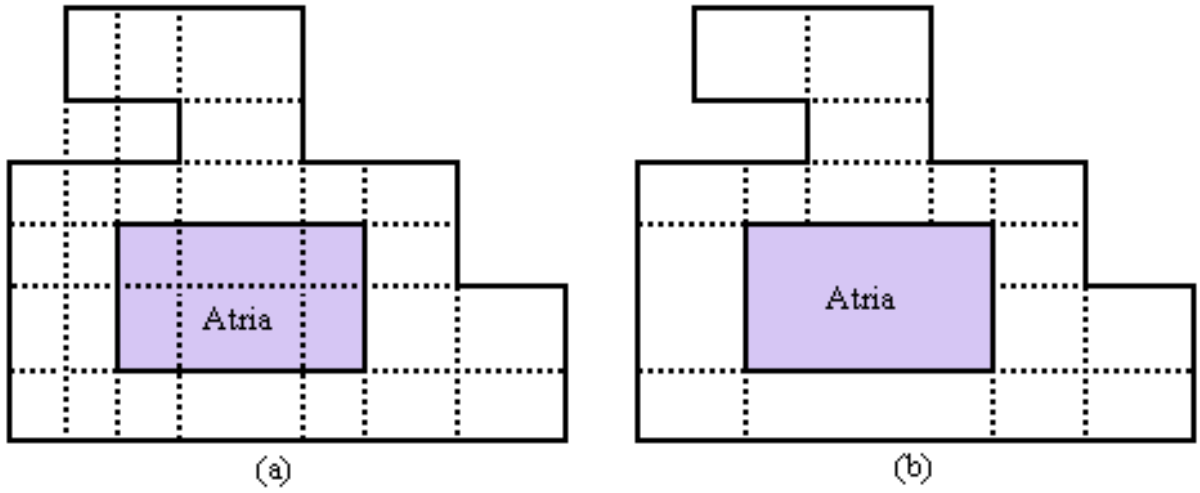


Figure 4-19 Comparison of partitioning techniques

4.7.6 Initialising an orthogonal genome

With the building layout partitioned and adjacent partitions 'monitoring' each other (via the adjacency graph), a genome is initialised for each partition.

The initialisation process starts by selecting the furthest left, upper partition. This is an arbitrary selection as the initialisation process could theoretically start at any partition, however to standardise the process it always starts at the same place. As the overall dimensions of this partition are known (and that it is a rectangle) the algorithm uses the initialisation procedure described earlier (see 4.5.2). At this stage the layout has one initialised partition (Figure 4-20a) however as frequently stated, maintaining column line continuity is essential. So an adjacent partition is initialised next. If there is more than one adjacent partition one is randomly selected. The adjacency graph is used to achieve this.

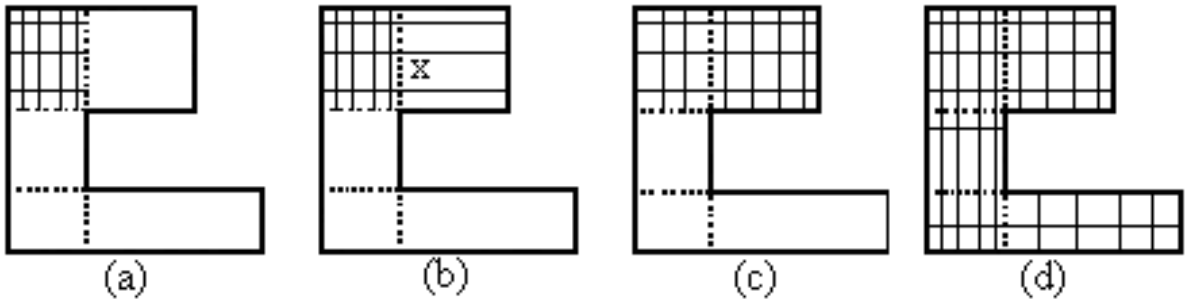


Figure 4-20 Example initialisation of orthogonal layout

Rather than initialising the adjacent partition as previously described, because the two partitions (the initial partition and its adjacent partition) must share a common edge, the

algorithm firstly copies the column spacings for this edge. For example in Figure 4-20b, edge *x* is shared between the two partitions so the y spacings from section 2 of the initial partition's genome are copied into the adjacent partition's genome. The remaining section is initialised as before, by generating new spacings in the required direction (see Figure 4-20c). This process is then repeated for another adjacent partition until the floor plan is fully initialised (see Figure 4-20d).

In complicated buildings it is possible that a partition may have been initialised 'by proxy' i.e. because all of its adjacent partitions have been initialised, it already has a complete genome. In this instance it skipped and the algorithm considers the next partition.

By constantly maintaining and updating the status of neighbouring sections, via the adjacency graph, the algorithm ensures column line continuity throughout the building. This continuity is vital to prevent the building from becoming a series of blocks that when placed together do not form a coherent solution. For example, in Figure 4-21 when considered in isolation each section is valid however, when considered as a whole, the building's layout is flawed because the columns do not align.

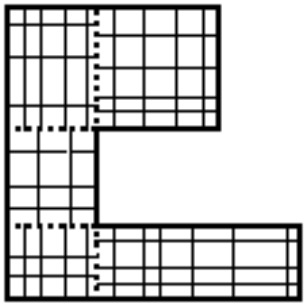


Figure 4-21 Invalid initialisation of orthogonal layout

The third section of the genome is assumed to be fixed throughout the building therefore every genome has an identical section 3 (see 4.7.1). It is acknowledged that because section 3 is constant, it could be removed from the genome. However it has been retained because it adds transparency i.e. all information pertaining to an individual is contained in the genome.

4.7.7 Evolutionary operators

The same evolutionary operators described previously are applied to each rectangular partition. However to ensure column continuity, the adjacency graph is incorporated at the end of the process to update the column line spacings in adjacent partitions:

- ***Mutation:*** Having selected the individual to mutate, the mutation operator randomly chooses (with uniform probability) one partition of the building and applies the mutation procedure discussed for a rectangular partition. Having mutated its genome, the section is placed back into the building and all adjacent sections are updated (Figure 4-22). This final step means the mutation operator is able to modify the building in only one location but the change ripples throughout the building, preventing column alignments degenerating. The adjacency graph used is during this process to determine which partitions need to be updated (for more information see 4.7.4).

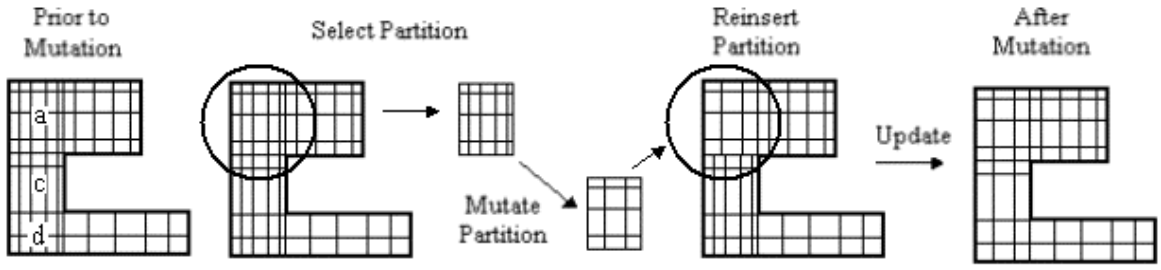


Figure 4-22 Mutation operator for layout design

Prior to mutation, partitions *a*, *c* and *d* had 6 genes within section 1 of their genome (because they share an adjacent edge in the x direction therefore they had identical genome section 1). However after mutation both the number and value of these genes had been altered. This occurs because although only partition *a* was selected for mutation, the adjacency graph recursively applies the change to all adjacent partitions (*c* and *d* in the y direction and *b* in the x direction) after reinsertion.

- ***Recombination:*** OBGRID employs a single point crossover operator (Goldberg, 1989), which exchanges part of the genomes associated with a section of the building. Recombination is as per a rectangular partition, however once recombination has been accomplished, the altered sections are reinserted into the building and all other adjacent partitions updated (as with the mutation operator described above) (Figure 4-23).

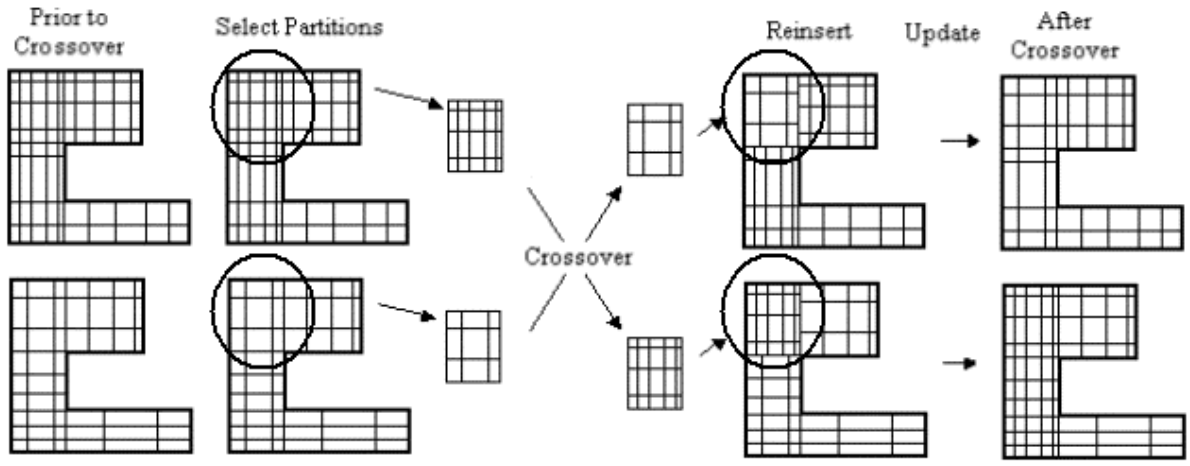


Figure 4-23 Crossover operator for layout design

It is noted that by updating adjacent partitions after reinsertion, the layout is substantially altered but this is the point. Recombination is a disruptive operator allows the algorithm to transfer spacings (or partial spacings) from one individual to another. However recombination can only transfer existing column locations between individuals, it cannot create new (although the column spacings maybe arranged in a new order).

4.7.8 Fitness function

OBGrid applies the same fitness function as previously described (see 4.5.5) to each partition in the floor plan and aggregates the results. Therefore individuals with more partitions will tend to have a numerically larger fitness, but remember, OBGRID aims to minimise this fitness.

4.8 Illustrative Example: Orthogonal Building

This section provides an illustrative example of OBGRID designing an orthogonal floor plan. The parameters in the EA tableau (Table 4-4) should be considered indicative because the aim of this work is to develop an appropriate representation rather than a complete building design system.

4.8.1 Introduction

The following test case was designed to assess OBGRID's performance. Unfortunately, unlike structural optimisation, there are not standard test cases. Therefore the 'C' shaped layout shown in Figure 4-24 was developed (no height restriction was imposed). The first

stage of the solution process involved partitioning the layout using a sweep line algorithm described above Figure 4-24.

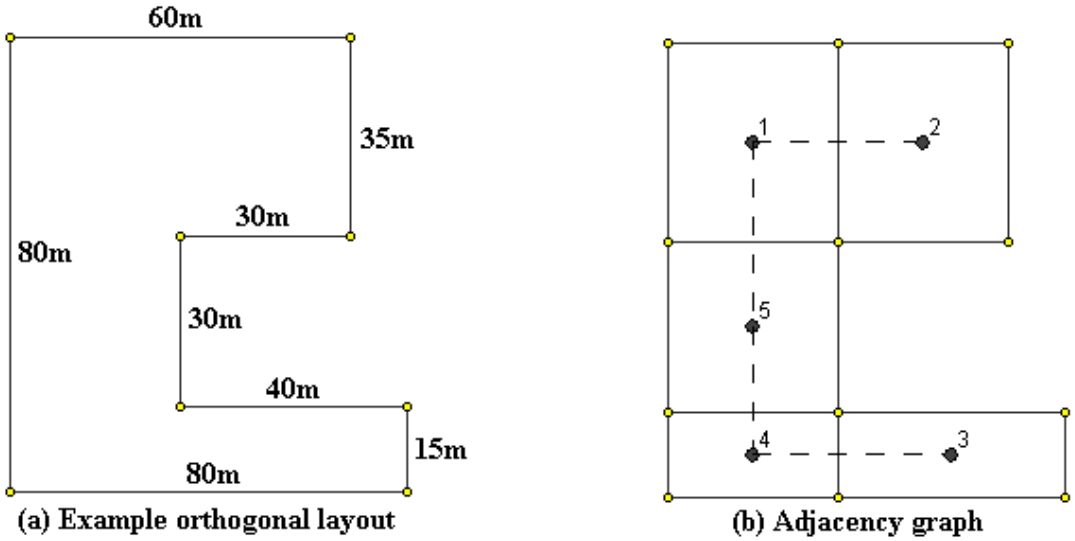


Figure 4-24 Orthogonal layout example

Table 4-4 EA Tableau for Orthogonal Building

Objective	Evolve example layout designs for a 'C' shaped boundary (with no atria or height restrictions)
Representation	3-Section string
Initialisation	Random initialisation (no seeding)
Raw Fitness	Based on: column spacing compatibility and column spacing uniformity
Selection	Tournament (size = 2)
Major Parameters*	P = 1, M = 100, G = 50, 100, 150 and 200
Evolutionary Operators:	
$Reproduction_{prob}$	0.1
Mutation operator	Point
$Mutation_{prob}$	0.3
Recombination operator	One point crossover
$Recombination_{prob}$	0.6

*P = Number of populations M = Population size G = Max number of generations

4.8.2 Results

Although this example is more complicated than the previous one, the results are actually fairly similar. For example, the best and average fitness trends downwards steeply at first before flattening off. The worst fitness does show a greater improvement that before, however it never converges and fluctuates between 30 and 85. Therefore this section will focus on how the number of generations affects a solution.

The following 4 performance graphs (see Figures 4-25, 4-26, 4-27, 4-28) each show the combined average fitness after 10 runs, for the short spanning system, with a maximum number of generations of 50, 100, 150 and 200. The final graph (see Figure 4-29) overlays all the results on one graph.

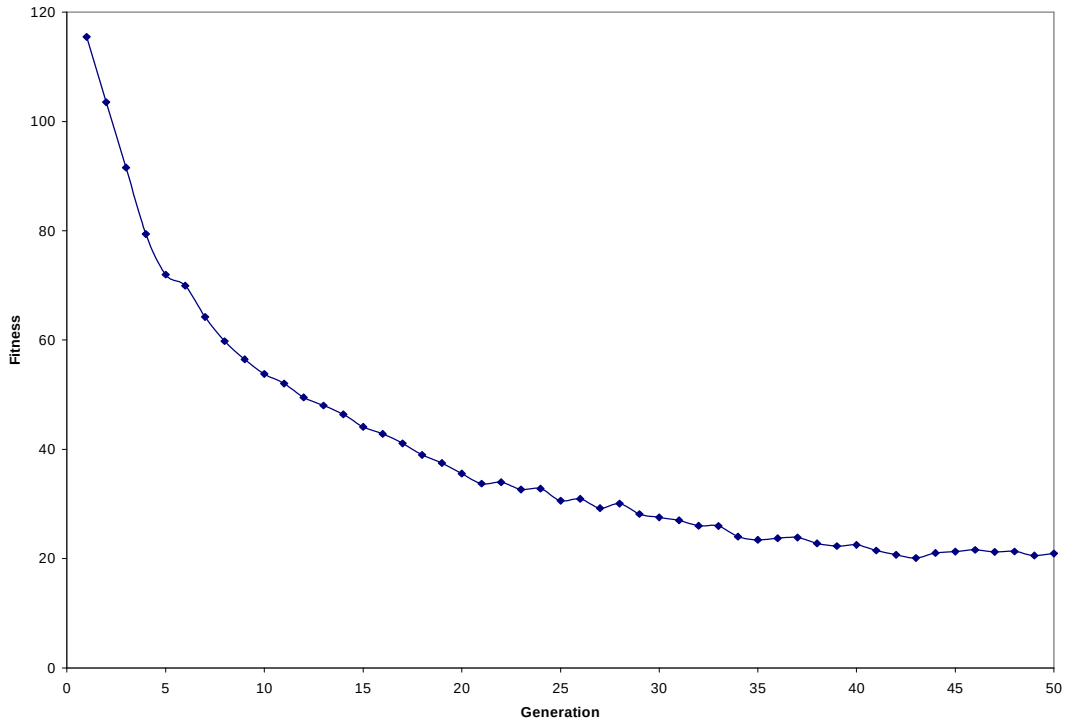


Figure 4-25 50 Generations

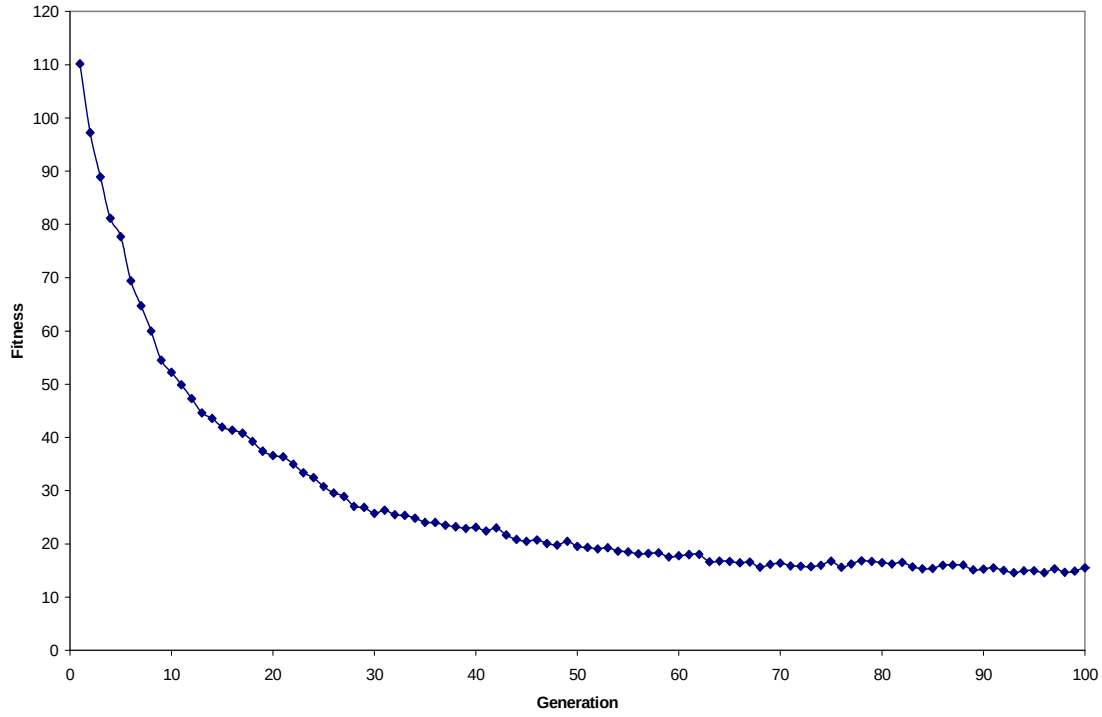


Figure 4-26 100 Generations

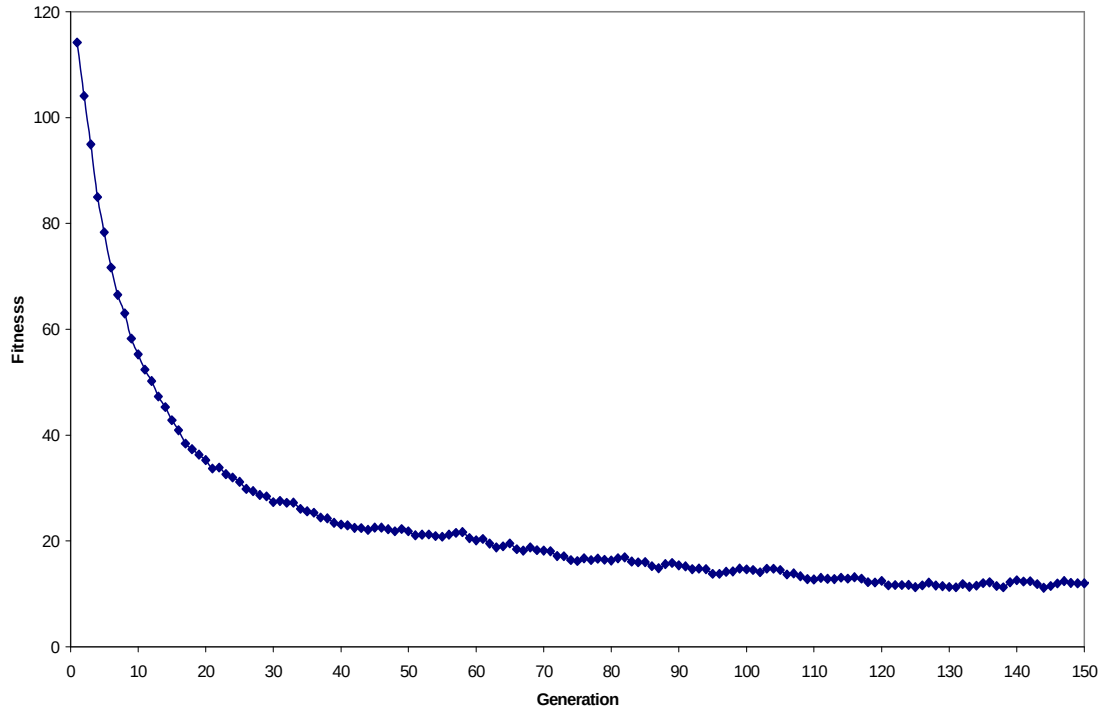


Figure 4-27 150 Generations

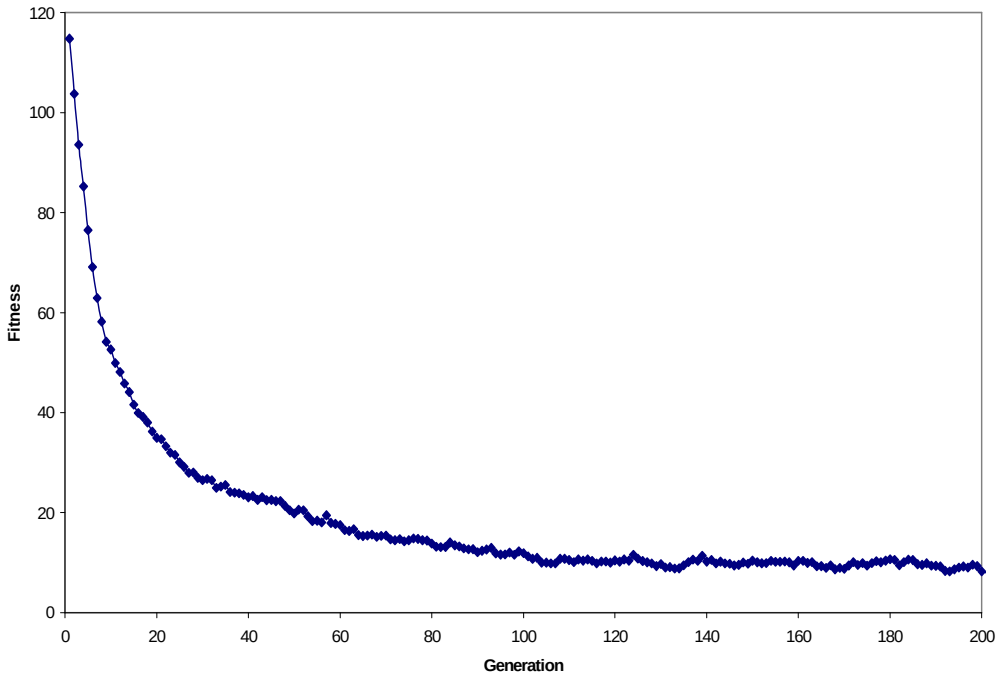


Figure 4-28 200 Generations

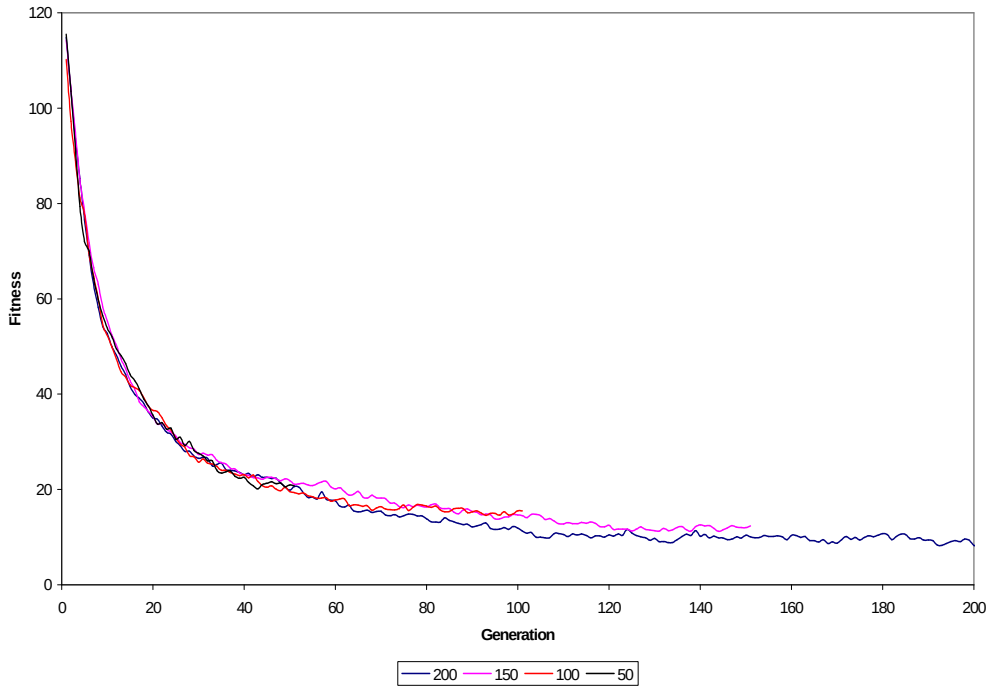


Figure 4-29 Performance graph for orthogonal building test

Figure 4-29 in particular indicates how robust this algorithm is, as all trend lines lie within a narrow range of each other. It also suggests that the most efficient number of generations to use is 100 (although it could be argued that 110 – 120 would be better). This is

because the algorithm has not converged, at a solution, by 50 but soon after 100 it has. Therefore to continue the search beyond this point, for example to 150, is computationally wasteful. If you were determined to expend more CPU time on this problem, restarting the algorithm to repeat the earlier generations rather than continuing with a stable solution would yield a greater return.

Finally Figure 4-30 depicts the returned solutions for each structural system after 100 generations.

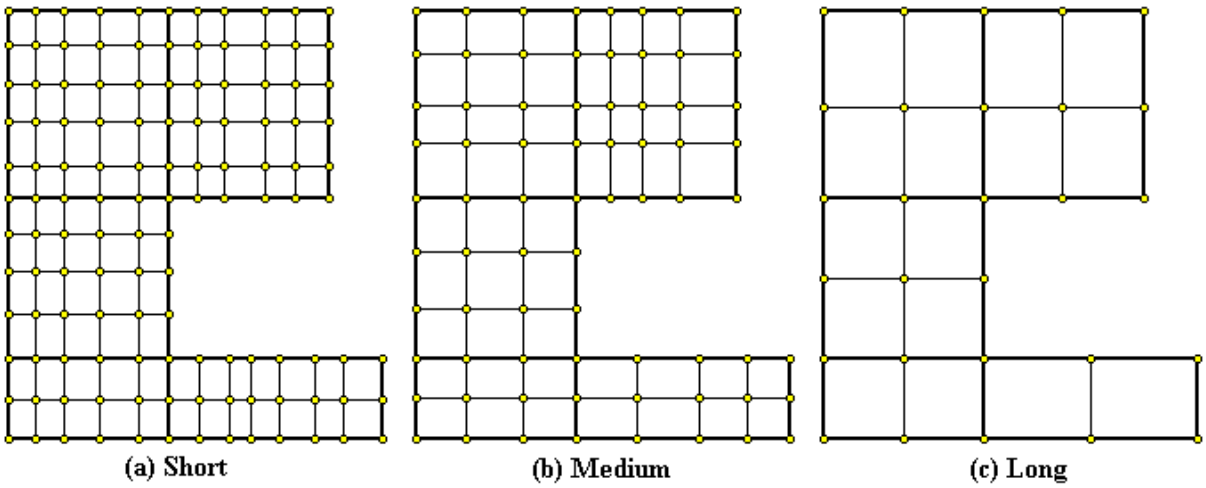


Figure 4-30 Returned solutions for orthogonal building layout

4.8.3 Conclusion

This example demonstrates how OBGRID solves orthogonal building layouts. To accomplish this, OBGRID partitions an orthogonal floor plan into rectangles and then uses the previously described rectangular methodology to design a layout. However an additional complication is the need to ensure column line continuation throughout the building. This constraint is achieved by using an 'adjacency graph', which updates adjacent partitions during initialisation and after evolution.

4.9 OBGRID an Orthogonal Buildings with Atria

This section contains a detailed description of how OBGRID handles orthogonal buildings with atria. It is acknowledged that this process is very similar to that for orthogonal buildings without atria, however this section has been included for completeness.

4.9.1 Partitioning

The floor plan is partitioned in two stages using a sweep line algorithm Figure 4-31. However, event points are any reflex vertex on the boundary or any vertex on an atrium. It should be noted that partitions do not 'cross atria' for example line 'x' in Figure 4-31.

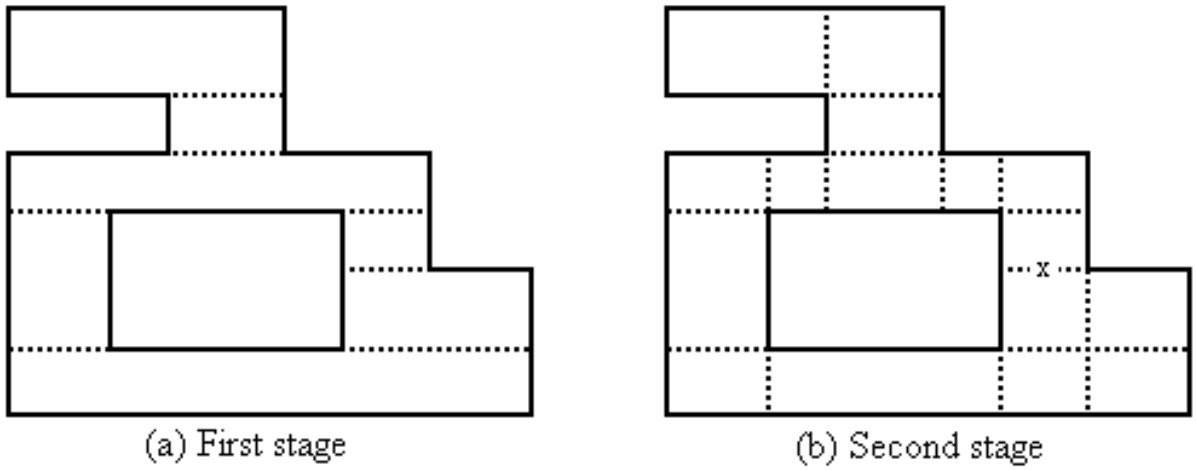


Figure 4-31 Polygon partitioning for orthogonal layout with atria

It is apparent that once atria are included, the number of partitions is dramatically increased. This is because atria add additional event points during partitioning. However the additional partitions are required to retain column alignment via the adjacency graph.

4.9.2 Adjacency Graph

An adjacency graph is associated with a floor plan using the methodology previously described. For example see Figure 4-32. However it should be noted that internal atria are not associated with an adjacency node. Thus column spacings on one side of an atria may not be found on the opposite side.

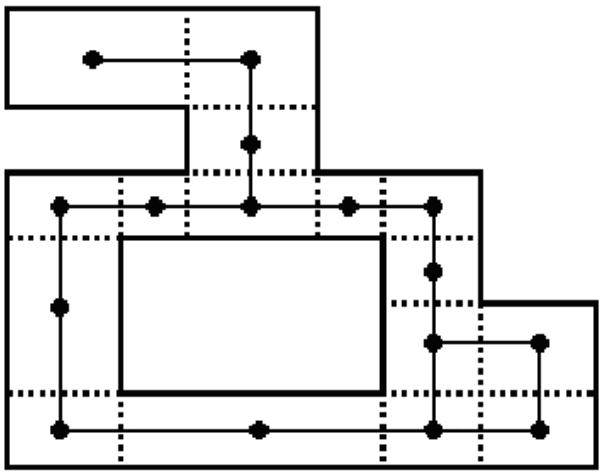


Figure 4-32 Adjacency graph for orthogonal layout with atria

4.10 Illustrative Example: Orthogonal Building with Atria

This section provides an illustrative example of OBGRID designing an orthogonal floor plan with atria. The parameters in the EA tableau Table 4-5 should be considered indicative because the aim of this work is to develop a representation rather than a complete building design system.

4.10.1 Introduction

The following test case was designed to assess OBGRID's performance. Unfortunately, unlike structural optimisation, there are not standard test cases. Therefore the layout shown in Figure 4-33a was developed as was partitioned using the sweep line algorithm described above to give the adjacency graph shown in Figure 4-33b.

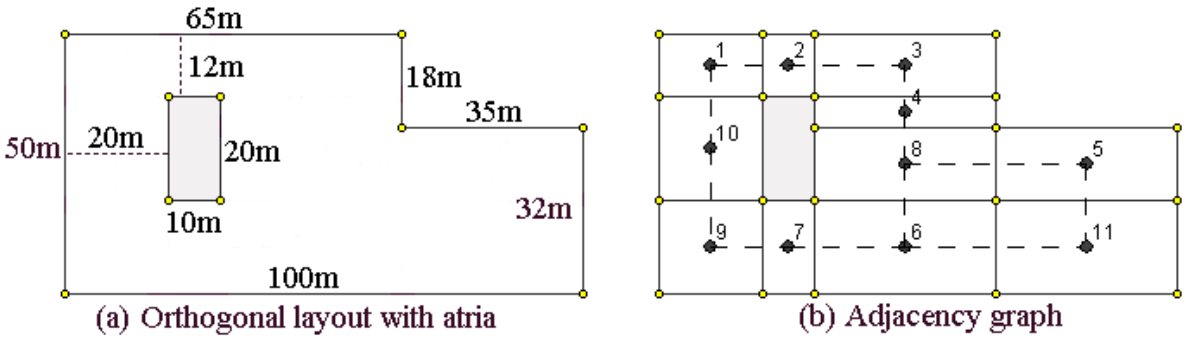


Figure 4-33 Orthogonal layout with atria example

Table 4-5 EA Tableau for orthogonal building with atria example

Objective	Evolve example building designs for the layout shown in Figure 4-29a
Representation	3-part string
Initialisation	Random initialisation (no seeding)
Raw Fitness	Based on: column spacing compatibility and column spacing uniformity
Selection	Tournament (size = 2)
Major Parameters*	P = 1, M = 100, G = 150
Evolutionary Operators:	
$Reproduction_{prob}$	0.1
Mutation operator	Point
$Mutation_{prob}$	0.3
Recombination operator	One point crossover
$Recombination_{prob}$	0.6

*P = Number of populations M = Population size G = Max number of generations

4.10.2 Results

The following 2 performance graphs showing the best, average and worst fitness during an indicative run for a short spanning system and a discussion of the results. Please note that because OBGRID is a minimisation algorithm, a lower fitness is considered beneficial.

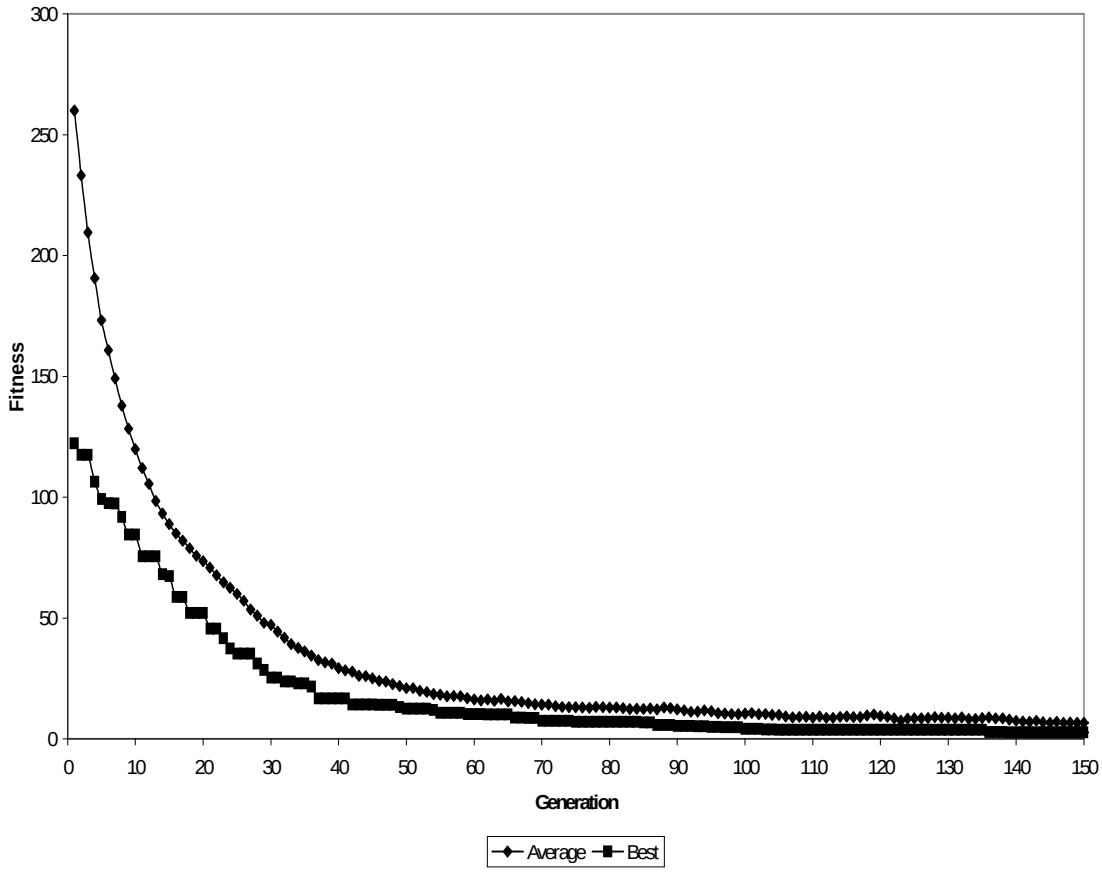


Figure 4-34 Best and average fitness

Figure 4-34 shows a much smaller spread between the average and best fitness when compared to the previous example without atria and for a rectangular outline. This could be because this problem is more challenging as it contains 11 partitions, compared to 5 for the example without atria and 1 for the rectangular outline. The partitions are also much more varied. For example, contrast the long, thin partition 4, with the almost square partition 10. Therefore after initialisation, the 'best' solution is only twice as good as the average (in the rectangular layout problem the best solution had a fitness of just under 8 while the average was approx. 160!). So on reflection a closer spread is expected. In spite of this, the best and average fitness have the usual characteristics: the best improves in steps, while the average gradually increases. This graph also demonstrates that the increased number of generations 150 is not excessive, as better solutions are frequently evolved until generation 135 (compared with generation 37 out of 50 in the rectangular example) reflecting this problems difficulty again.

It is also important to note that elitism was used with layout i.e. the best of generation was always copied over to the next without modification. Although this approach can hinder the search by potentially focusing on local optima, because this is a significantly harder problem it was used after some experimentation indicated its value (see Figure 4-35). Figure 4-35 highlights some of the characteristics found with elitism (if used in an appropriate setting): although both fitness curves have the same overall trend, without elitism it is more ragged and returns inferior results.

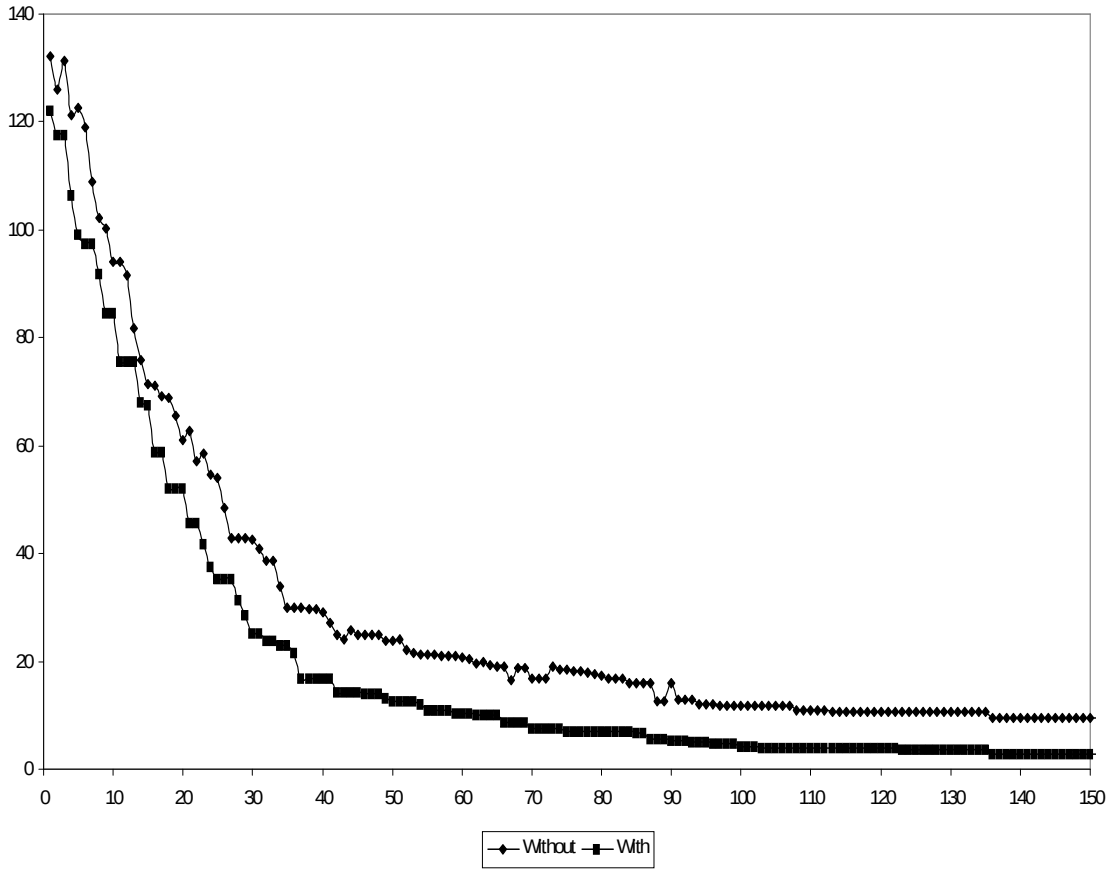


Figure 4-35 Comparison with and without elitism

Tbe worst fitness graph (see Figure 4-36) shows a greater trend of improvement when compared to the rectangular layout problem. This is probably because given that the problem is more complex they have less chance of destroying a good layout as these are harder to find (where as for the rectangular problem, the 'best' solution was actually quiet easy to locate). Also the evolutionary operators are only applied to one partition per generation. Therefore their effect is diminished because fitness is cumulative therefore they have less effect.

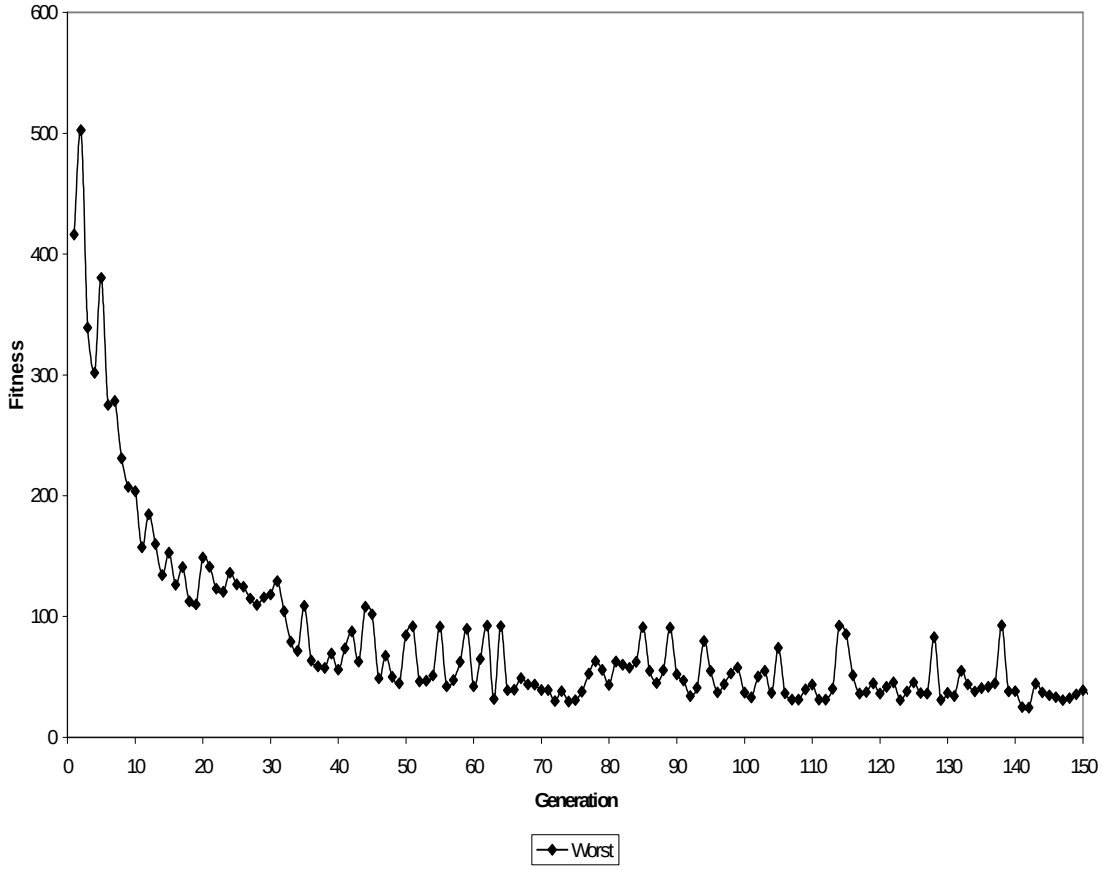


Figure 4-36 Worst fitness

Figure 4-37 indicates the best layouts returned for each structural spanning system, all spans are within their economic range. However as previously noted, OBGRID does tend to struggle evolving regular column spacings as the number of columns increases.

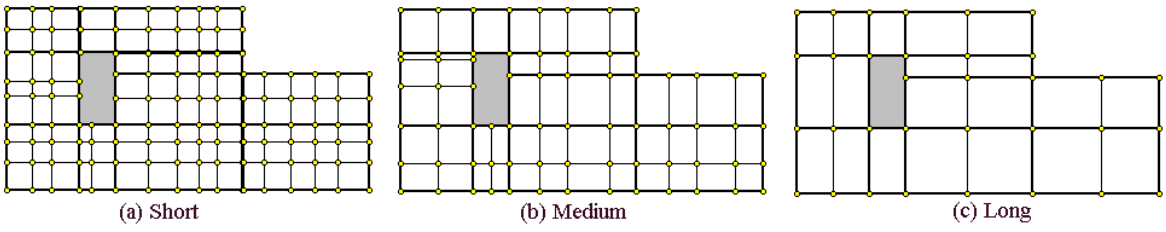


Figure 4-37 Returned solutions for orthogonal building with atria

4.10.3 Conclusion

This example demonstrates that OBGRID is capable of solving orthogonal layouts including atria and indicates one deficiency of this methodology: the inclusion of atria tends to bias the search towards shorter spanning systems because the number of partition increases and thus

each partition becomes smaller (when compared to the equivalent layout without atria). And as each partition is solved independently, the floor plan's average span length is reduced. This could limit the performance of this approach with very complex layouts.

4.11 Conclusions

The EA based methodology described in this chapter is able to solve conceptual layout design problems for orthogonal, commercial buildings which is an improvement over all existing systems that are limited to rectangular floor plans. This work achieves this, by partitioning orthogonal floor plans using a sweep line algorithm to create rectangular sections that can be solved individually. Also to ensure column line continuity, an adjacency graph that associates adjacent partitions, is used especially during initialisation and evolution. However the inclusion of atria, to a floor plan, tends to increase the number of partitions biasing the search towards shorter spanning systems. This is because once atria are included, the partitions become smaller and as each partition is solved independently the spans are reduced. This could limit the performance of this approach with very complex layouts however OBGRID seems to handle the examples effectively, although the only true test would be to trial OBGRID over a period of months in a design office.

5 Conceptual Geometric Design of 'Geodesic-like' Domes

5.1 Abstract

Dome layout design is a non-trivial task because every joint and member must be located on the dome's external surface and not impinge on the internal void. The only previous stochastic methodology (Shea and Cagan, 1997) tackles this by creating a 2D truss that is subsequently projected onto a predefined curved surface. Therefore the solution is a 3D object, but the search is conducted in 2D. While this 'projection' or 2.5D technique reduces the number of problem variables, by constraining the third dimension to be dependent on the planar layout, it also excludes a dome's two most important variables from the search: surface area and enclosed volume. Thus the results, while spatially innovative, are typically sub-optimal.

This chapter describes a new approach using an evolutionary algorithm with string representation that designs directly in 3D, with surface area and enclosed volume as the major search parameters. The string representation encodes support and joint positions, which are converted into a dome by constructing its corresponding convex hull. Once constructed, the hull's edges become the structural members and its vertices the joints. Finally, structural analysis is used to determine performance within the context of user-defined constraints. This technique avoids many of the problems experienced by the previous approach that suffers when restrictive constraints such as the requirement to maintain $1/8^{th}$ symmetry are removed.

The aim of this chapter is to investigate existing and develop new knowledge for dome design. It should be noted that there is no obvious connection between the structure investigated in this chapter and the last. This is because this thesis is focused on investigating how civil engineering structures can be represented using evolutionary algorithms. Therefore domes were deliberately chosen because they are very different to buildings and thus the research had to start at the beginning.

Keywords: geodesic domes, evolutionary algorithm, convex hull, incremental algorithm.

5.2 Introduction

Domes are a common architectural structure, synonymous with many landmark buildings including St Peter's Basilica (Rome) and St Paul's Cathedral (London). Traditionally domes

are created by rotating an arch about its' vertical axis. However, in the 1950's a new approach was proposed: Geodesic domes.

5.2.1 Geodesic Domes

Invented by Buckminster Fuller in 1954 (The Buckminster Fuller Institute, 2005), geodesic domes have homogeneity in both member length and nodal angular incidence and are considered by some to be the strongest, lightest and most efficient building system (Motro, 1994). Geodesic dome geometry is usually based upon the sub division of a spherical surface into triangles (because triangles are the simplest non-deformable rigid shape). However, geodesic dome geometry may also be based upon the sub division of any Platonic[13] or Archimedean[14] solid. Perhaps one of the most famous geodesic domes is the Epcot Center in Florida (Figure 5-1).

Figure 5-1 Epcot Center (Florida)

There are four types of geodesic dome (Motro, 1994): frame (or skeleton) single layer domes; truss or double layer domes; stressed skin domes; formed surface domes. However, this chapter will only consider the first type.

[13] Convex polyhedra with identical faces constructed of congruent, regular polygons. There are exactly five Platonic solids the cube, dodecahedron, isosahedron, octahedron and tetrahedron.

[14] Convex polyhedra that have a similar arrangement of nonintersecting regular convex polygons of two or more different types arranged in the same way about each vertex with all sides the same length. There are exactly thirteen Archimedean solids (Weisstein, 2005).

5.2.2 Geodesic Patterns

Geodesic domes based on spheres, start by inscribing 'great circles' onto the sphere (a process that can create no more than 120 similar but irregular triangles on the surface or a maximum of 20 equilateral triangles). Alternate and triacon breakdowns (Motro, 1994) are then applied to this network of triangles (Figure 5-2). In Figure 5-2 'frequency' refers to the number of subdivisions per side of the original triangle. Thus a frequency 2 breakdown subdivides each side of the original triangle into two. Once a breakdown has been applied, the geodesic layout is complete.

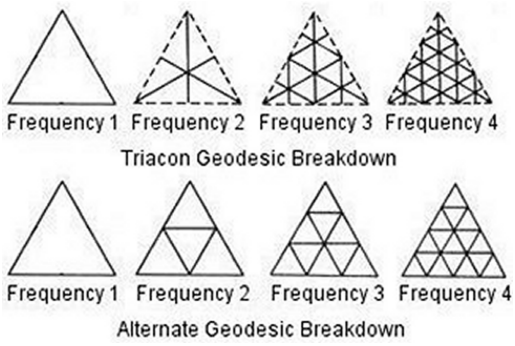


Figure 5-2 Triacon and alternate geodesic breakdowns

It should also be noted that this work only creates domes with geodesic characteristics not strict geodesic domes. This is because geodesic breakdowns are not explicitly enforced therefore there the evolved structures may not adhere to geodesic patterns (as defined by the triacon and alternate breakdowns). Thus the solutions will be described as 'geodesic-like'. Geodesic breakdowns cannot be enforced in this work, because the representation does not consider shapes, only points. However the representation is capable of evolving spatially innovative and structurally efficient designs.

5.3 Related Work

Within the field of structural design using stochastic search algorithms, very little research has been published on dome design. Therefore this section will discuss papers by Porter et al (1995) and Shea and Cagan (1997) in detail.

Porter et al (1995) use a genetic algorithm to compute the length and location of geodesics[15] (not geodesic domes) on complicated curved surfaces. They demonstrate a technique capable of producing results comparable to the theoretical optima for spherical surfaces. However, they only calculate a linear set of geodesics (between two points), so each geodesic links to at most two others (one at each end). In dome design however, an arbitrary number of members are connected at each structural joint. Therefore their technique is not appropriate for dome design.

Shea and Cagan (1997) apply simulated annealing (Kirkpatrick, 1973) combined with a shape grammar representation to dome design, a process they call 'shape annealing'. Their technique, constructs a 2D truss that is projected onto a predefined 3D curved surface constraining the third coordinate (z) to be dependent on the other two (x,y). Therefore, search is conducted within a 2D design domain. However, while they demonstrate that shape annealing is capable of generating novel solutions that are comparable to those produced by other shape optimization techniques (Pedersen, 1973), projection hampers the search by removing two of the most important variables: enclosed volume and surface area. Therefore, once some of the constraints are removed e.g. design is required to maintain $1/8^{th}$ symmetry; most of the evolved solutions bear little resemblance to geodesic domes. For example, a few extremely large members may dominate the dome so that the evolved structure is actually more like a pyramid or simply not resemble a dome (Figure 5-3).

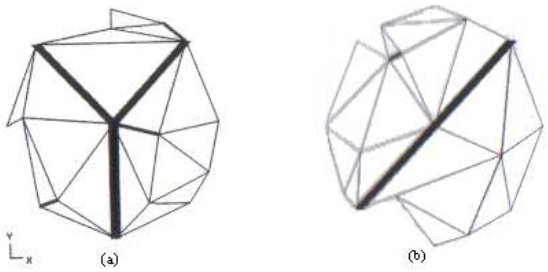


Figure 5-3 Example results from Shea and Cagan (1997)

5.4 Convex Hulls

Computational Geometry is the design and analysis of efficient algorithms (usually computer based) for solving geometric problems (Shamos, 1978) and convex hulls are one of its

[15] A locally length-minimising curve.

fundamental structures. The following section will provide an overview of convex hulls including what they are, their applications and some issues related to their construction while the subsequent section will describe a hull construction algorithm in detail.

5.4.1 What are convex hulls?

The convex hull of a finite set of points is considered to be the convex polyhedra with the smallest volume that encloses that set (Figure 5-4). This work makes extensive use of convex hulls to create dome from a set of vertices by using the incremental algorithm, which are described in the following sections.

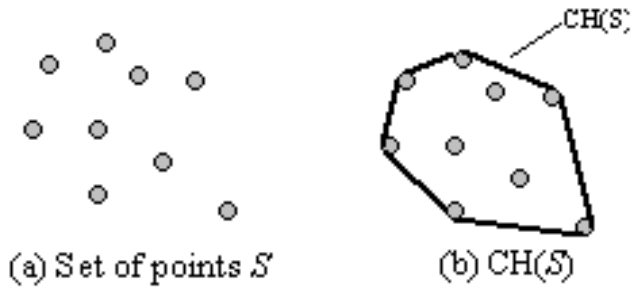


Figure 5-4 Convex hull CH(S) of S

5.4.2 Applications of convex hulls

Convex hulls produce convex approximations of non-convex point sets. Therefore they are commonly used in the following applications (this list is by no means exhaustive, merely indicative):

- ***Pattern recognition:*** A complex shape may be approximated via its convex hull and compared to a database of known shapes (Laszlo, 1996).
- ***Motion planning:*** A robot may approximate its footprint via a convex hull to simplify terrain negotiation (Laszlo, 1996).
- ***Computer animation:*** In computer games etc. collision detection may be improved by approximating shapes to their convex hulls and only comparing the actual shapes if the hulls indicate a collision (de Berg et al. 1997).

5.4.3 Polyhedra

This sub section contains a general discussion about polyhedra: the shape formed by convex hulls. Polyhedra are considered to be three-dimensional objects composed of a finite number of flat faces, edges and vertices (Figure 5-5a). They can also be described as the 3D generalisation of a 2D polygon[16]. Within this work, every dome will be convex and have triangular faces: technically a simplicial complex[17]. However, domes will be referred to as convex polyhedra.

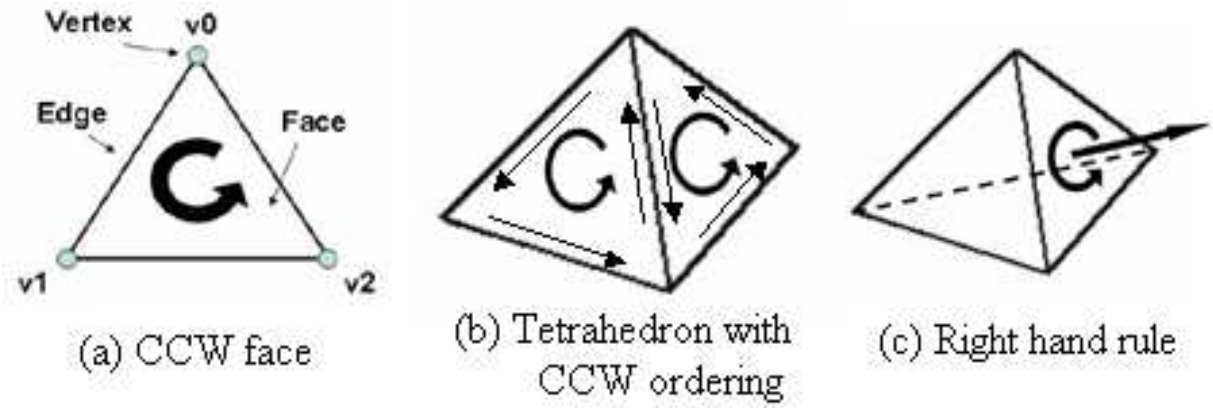


Figure 5-5 Polyhedral properties

Polyhedral faces (Figure 5-5a), in this work, have an important feature: they maintain their vertices so that when 'viewed' from the exterior, vertices have a counter clockwise (CCW) ordering ensuring the right hand rule always yields a vector normal to the face, pointing away from the polyhedron (O'Rourke, 1998). This is not simply for aesthetic reasons, as the right hand rule is used judiciously during convex hull construction.

5.4.4 Signed volumes

The volume V of a pyramid with a base area B and height h can be calculated by:

$$V = \frac{B.h}{3} \qquad (1)$$

However (Eq 1) does not allow for the direct computation of tetrahedral volume from vertices (as required during this work). Therefore, volumes will be calculated via the

[16] The region of the plane bounded by a finite collection of line segments, forming a simple closed curve.

[17] Space with a triangulation.

determinant form of the cross product. For example, a tetrahedron defined by four vertices (x_i, y_i, z_i) has the volume:

$$V = \frac{1}{3!}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \quad (2)$$

The volume calculated by (Eq 2) is described as 'signed' because it can be positive or negative. Signed volumes form an integral part of many algorithms in computational geometry because they remove the need to perform the complex calculations to determine angular relationships between points (especially when considering spatial relationships). For example, whether a point is to the left or right of another. During this work, a negative volume is generated when a face *f* forms a tetrahedron with a point *p* that can 'see' its vertices in a CCW manner (Figure 5-6).

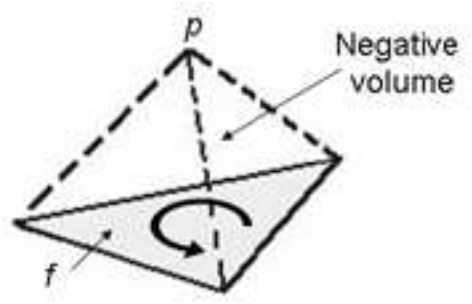


Figure 5-6 Negative volume generated by CCW face f and point p

5.4.5 Visibility

The incremental algorithm is based upon determining the visibility of a face from a point. Therefore, a simple yet robust routine is required. A face *f* is considered to be visible from point *p*, iff[18] a line drawn from *p* to some point *x* interior to *f* does not intersect with the polyhedra at any point other than *x*. For example in (Figure 5-7), *f* is visible from *p'* but invisible from *p"*.

[18] "if and only if".

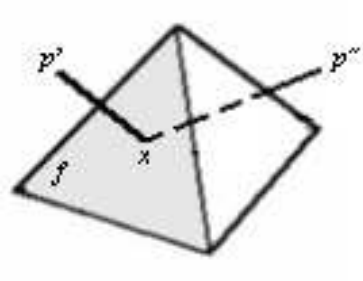


Figure 5-7 Example visibility of face f from points p' and p"

Visibility can also be formally defined using sets (Eq 3). It should be noted that (Eq 3) defines a face that is 'edge on' to *p* to be invisible. A face is considered to be 'edge on' when only its edge is visible from *p* i.e. the face's vertices and point *p* are coplanar.

$$iff \quad px \cap CH = \{x\} \tag{3}$$

The visibility of a face *f* from a point *p* is determined by calculating the signed volume of the tetrahedron defined by *f* and *p*. *f* is considered to be visible from *p*, iff the signed volume is negative.

5.5 Incremental Algorithm in 2D

Several algorithms have been developed to construct a convex hull (O'Rourke, 1998). However this chapter only considers one: the incremental algorithm. The following section discusses the incremental algorithm in detail, starting with an overview and an illustrative example in 2D. The following section describes the implementation for this work.

5.5.1 Overview

The incremental algorithm constructs the convex hull *CH* of a finite set of points *S* by taking a subset S_{sub} of *S* and constructing its convex hull $CH(S_{sub})$. Having constructed $CH(S_{sub})$ the algorithm adds an additional point to S_{sub} and updates the hull (if required). This process continues until all points from the original set *S* are included in the convex hull. Figure 5-8 illustrates the incremental algorithm in 2D.

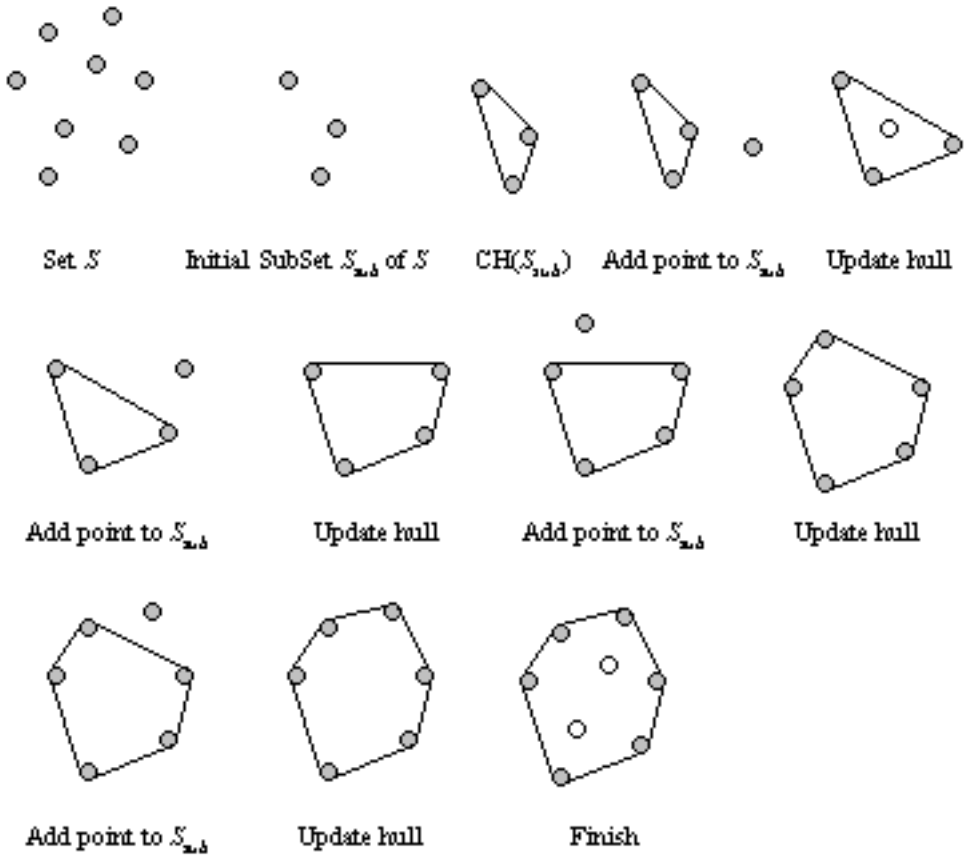


Figure 5-8 Illustrative example of the incremental algorithm in 2D

5.5.2 Illustrative example

This section provides an illustrative example of an evolutionary algorithm combined with a 2D convex hull algorithm. The aim is to evolve the largest possible circle within a square of side length 200m. A string representation was used containing points randomly located in the problem domain. The EA tableau (Table 5-1) details the values applied to the key evolutionary parameters however it should be noted that no attempt has been made to optimise any values. For more information on the evolutionary operators and fitness function, please review the subsequent sections.

Table 5-1 EA tableau for 2D illustrative example

Objective	Maximise enclosed area, minimise perimeter
Representation	String containing random points
Initialisation	Random initialisation (no seeding)
Raw Fitness	Based on: enclosed volume and surface area
Selection	Tournament (size = 3)
Major Parameters	P = 1, M = 200, G = 60

Evolutionary Operators:	
$Reproduction_{prob}$	0.1
Mutation operator(s)	Mutate existing point, add new points, delete existing points
$Mutation_{prob}$	0.4 (the actual mutation operator is selected at random)
Crossover operator	n point crossover
$Crossover_{prob}$	0.5

5.5.3 Results

Figure 5-9 shows the fitness of the best of generation during the run, while (Figure 5-10) indicates the best layout found in generation 54 (the light grey circle indicates the optimum).

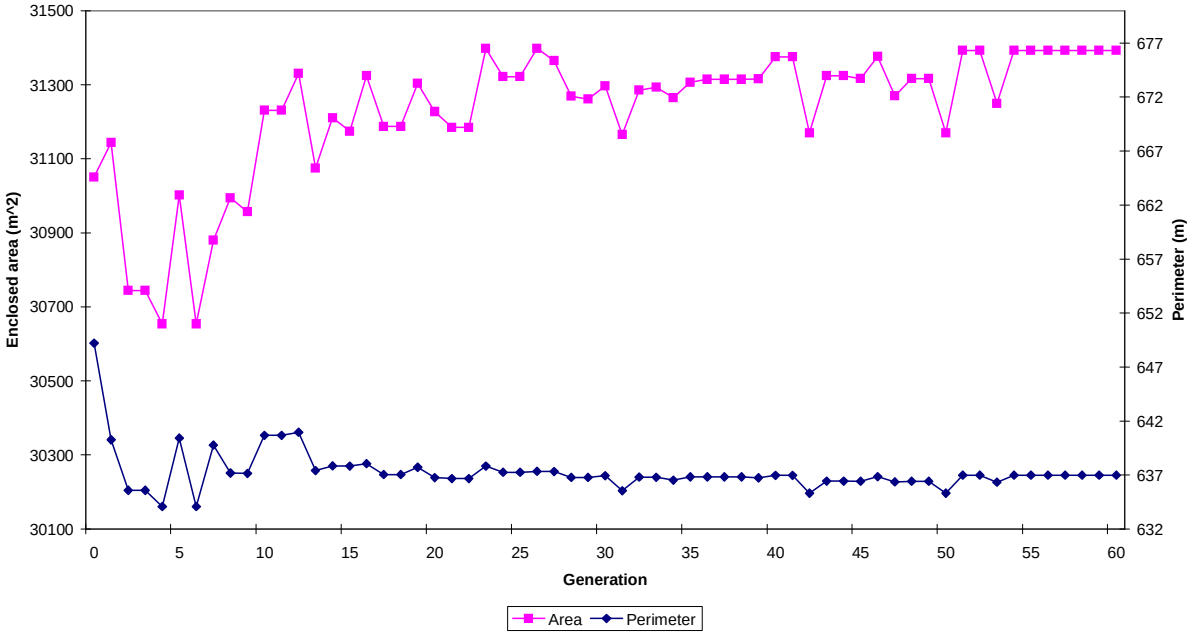


Figure 5-9 Performance graph for 2D example

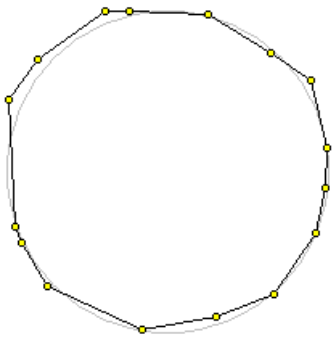


Figure 5-10 Best of generation 54 for 2D example

5.5.4 Conclusion

Although simple, this example demonstrates how effective the combination of an evolutionary algorithm and convex hull algorithm can be, as the best solution has an enclosed area (31392.4 m^2) within 0.1% of the optimum, while the perimeter (639.99 m) is within 1.8%.

5.6 Incremental Algorithm in 3D

The previous section provided an overview of the incremental algorithm and an illustrative example in 2D, however domes are a 3D structure. Therefore the following sections describe how the incremental algorithm can be implemented in 3D. The implementation described is a $O(n^2)$ algorithm. This means that if the number of points n doubles, algorithm execution time will increase four-fold. A possible improvement is discussed in the future work section of this thesis.

In this work, the initial subset S_{sub} always contains just four points: three non-collinear[19] points and a fourth non-coplanar[20] point. This ensures that the initial convex hull is always a tetrahedron: its base formed by the non-collinear points and its apex by the non-coplanar point. If S does not contain these points, it is 2D and invalid for this problem.

When an additional point p_i is added to S_{sub}, the issue of whether to update the existing convex hull $CH(S_{sub})$ involves considering the question: Are there any faces of $CH(S_{sub})$ visible from p_i?

- ***No***. If none of $CH(S_{sub})$'s faces are visible from p_i, then p_i must be internal to $CH(S_{sub})$. Therefore $CH(S_{sub})$ is still valid, as it encloses all points and remains unaltered.
- ***Yes***. If some of $CH(S_{sub})$'s faces are visible from p_i, then p_i must be exterior to $CH(S_{sub})$. Therefore $CH(S_{sub})$ is invalid, because it no longer encloses all points and must be updated to include p_i.

[19] Three or more points are collinear if they lie on the same straight line.

[20] Four or more points are coplanar if they lie on the same geometric plane.

5.6.1 Updating the convex hull CH_{i-1}

This section describes how an existing convex hull is updated to include a new point. The convex hull is updated in two stages: locating the horizon and incorporating the external point.

Conceptually, the external point p_i divides the existing hull into two regions: the visible and the invisible. The horizon (de Berg, 2000) is formed by the series of edges that are adjacent to both a visible and invisible face (Figure 5-11) and can be located once the visibility of every face from p_i has been determined.

To incorporate the external point into the existing convex hull, a new set of new faces must be appended to it. All new faces will be triangular, constructed from a horizon edge and have an apex at p_i (Figure 5-11). After building these new faces, the original faces (that were visible from p_i) are now underneath the new faces and should be deleted (along with any superfluous edges and vertices). At the end of this process convex hull is completely updated (Figure 5-11).

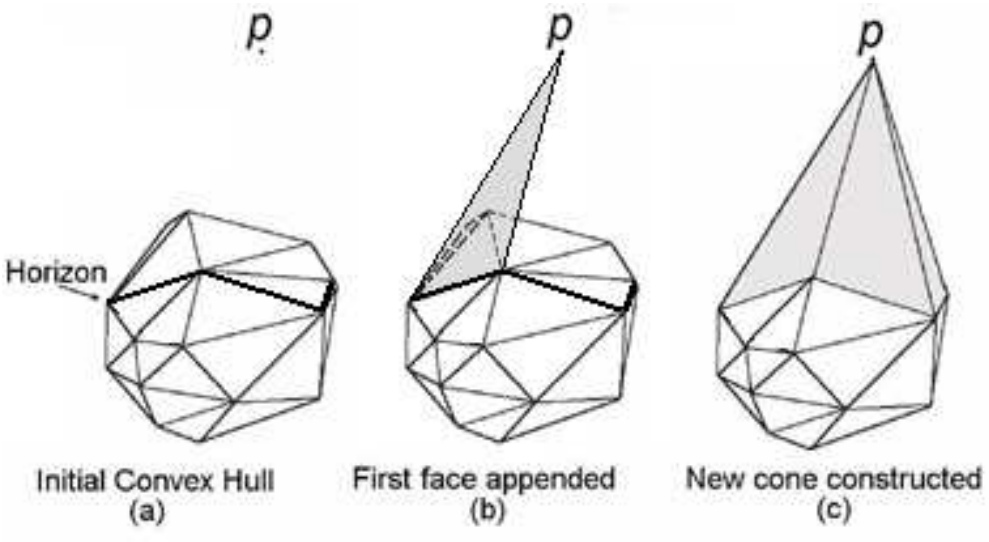


Figure 5-11 Updating an existing hull (adapted from O'Rourke 1998)

At this point, it is worth returning to the definition of visibility that considers 'edge on' faces to be invisible (see 5.4.5 Visibility). If 'edge on' faces are considered to be invisible, then any new faces will be simply appended to existing 'edge on' faces. However, if 'edge on' faces are considered to be visible, then the algorithm will attempt to remove them and replace them with a single new face. Unfortunately, the new face may not be triangular or

result in the existing face fracturing into a series of smaller faces making the algorithm significantly more computationally intensive (de Berg, 2000). This is why 'edge on' faces are treated as invisible (in this work).

5.7 Current Work

This work uses an evolutionary algorithm (EA) with string representation to search for potential solutions and following sections describe its structure and function.

5.7.1 Representation

The representation allows potential solutions to be included in the EA's search and several canonical forms have been published including string and trees. This section discusses how a representation was developed for geometric dome design.

Although domes are skeletal structures containing joints and loads, this work considers the members to be implicitly defined by the joint layout. This is because members must form the external surface and not impinge on the internal void. Therefore a member can only span between 'adjacent' joints. Geodesic domes are also composed of triangles, again limiting the joints a particular member can span to. In light of this, this work considers dome design to be more of a parametric problem. Once the joint layout has been evolved, member configurations can be determined. Parametric problems are generally best represented by string genomes. Therefore this work uses a 3-section string representation (Figure 5-12), with each gene encoding a potential vertex on the convex hull. Genes are composed of software objects as shown in Figure 5-13 (supports are considered to be vertices at Z = 0).

It is acknowledged that section 1 and sometimes section 2 (when user defined support locations are used) could be removed from the genome because they are constant for all individuals. However they have been retained because they add 'transparency'. Transparency is the idea that the user should have a single reference to for an individual (as in nature where all cells contain the complete genome rather than just the sections appropriate to its own function). For example, if the location and magnitude of loads is removed from the genome where should it be placed and why?

- ***Section 1:*** encodes the location of and magnitude of loads that must be supported by the structure (in addition to its self weight).

- ***Section 2:*** encodes the location of the dome supports. Dome supports represent locations at which the dome is attached to the ground or supporting structure. In this work, dome supports are vertices in the plane z = 0. Supports can be user specified or searched for during the evolutionary process. For example if the user has predetermined support locations then it is pointless for the algorithm to search for the optimum because they are fixed. However if the user has no preference support locations are included in the search.
- ***Section 3:*** encodes the location of potential dome vertices (structural joints). For non-trivial structures this is the largest section of the genome. However, each gene is only a potential vertex because they may not lie on the genome's convex hull (as generated by the incremental algorithm) and therefore may not form the dome.

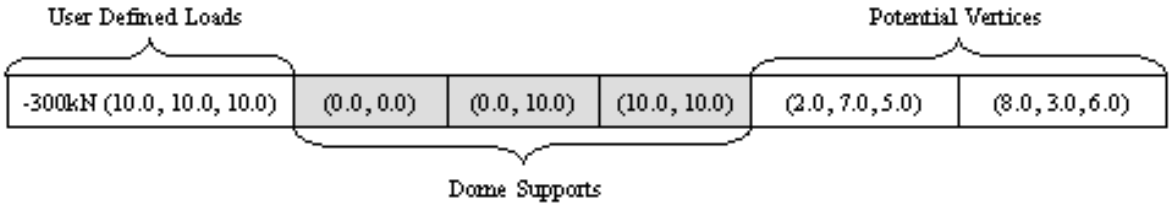


Figure 5-12 Example genome for dome design

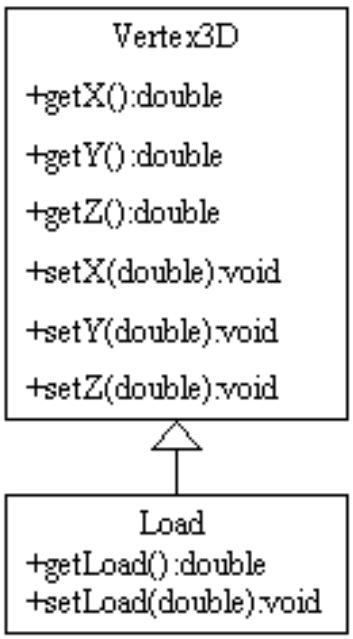


Figure 5-13 Class diagram for dome genes

5.7.2 Genome ordering

The incremental algorithm has an interesting feature: as it gradually constructs the convex hull, the final structure is dependent on the order in which the vertices are added. So two convex hulls constructed from the same set of vertices but with different orderings, could have identical vertices but different arrangements of faces and edges and thus different structural responses. Therefore, the EA must consider genome ordering during its search.

5.7.3 Initialisation

As this work is aimed at the conceptual design stage, the initial number of input parameters has been kept to a minimum: the user is only required to input the location of any loads and define the size of the circular base. If required the user can stipulate the number and location of the dome supports and ensure that they are constant for all individuals but if not, the algorithm will search for appropriate support positions during the run.

5.7.4 Initialisation of dome supports

Dome supports represent the locations at which the dome is attached to the ground or supporting structure. Some structural optimization techniques specify support positions using a ground structure (Dorn et al, 1964), but this can bias or inhibit the search (especially when an asymmetric or lateral loading is applied to the dome). Therefore this work, removes the need for a ground structure including number and location of supports in the search.

Support locations are a series of randomly generated points on the circumference of the circular base (the base circumference is the same for all individuals) generated by selecting two numbers x_1 and x_2 from a uniform distribution between -1 and 1 (ensuring that the sum of the square of both numbers is not greater than or equal to 1). The corresponding Cartesian coordinates related to x_1 and x_2 are given by (Eq 4) (Weisstein, 2005).

$$x = \frac{x_1^2 - x_2^2}{x_1^2 + x_2^2} \qquad y = \frac{2.x_1^2.x_2^2}{x_1^2 + x_2^2} \tag{4}$$

NB z coordinates are not generated as the base is assumed to lie on the plane z = 0.

5.7.5 Initialisation of dome vertices

Vertices are generated from random points within a cube that is centered on the dome's base and with a side length of equivalent to the diameter of the base. This procedure is used to prevent the EA searching in completely unproductive regions (a large number of useless points will still be generated, however these must be included to allow the EA to explore the search space). To prevent additional supports being generated, vertices may not lie on the domain boundaries. While this does improve the search, it does prevent the algorithm from evolving domes, which has sections wider than the base.

At the outset each individual has a random number of vertices in its genome (an upper limit of 100 vertices and lower limit of 1 was generally used in this work, however this was purely arbitrary and no attempt was made to optimize it). However because the dome is only constructed from vertices that lie on the convex hull, it does not necessarily follow that all of these will be used to construct the dome. This can cause bloat.

5.7.6 Evolutionary operators

Within the EA's search, the loads section of the genome is unaffected (as these loads must be carried by every solution) while the crossover and mutation operations are individually applied to the two remaining sections.

- ***Recombination****:* An 'n-point' crossover operator, which is a generalised version of one-point crossover with several cut points, is employed in this working creating variable length genomes. An example n-point crossover operator is shown in Figure 5-14 although integer genomes are used for clarity.

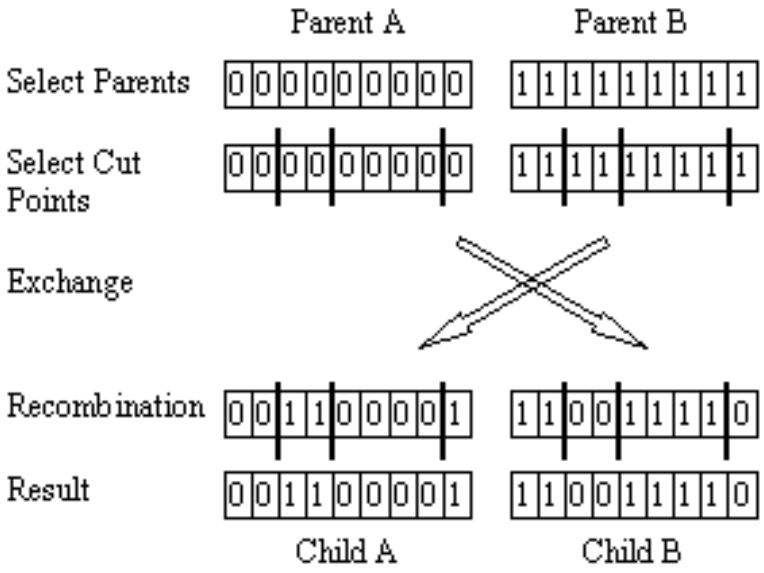


Figure 5-14 Example n-point crossover

- ***Mutation****:* Several mutation operators are used in this system: point, shuffle, addition and deletion. Point mutation (Figure 5-15) randomly selects a gene to alter and then uses the same procedures as described during initialisation to generate a new point depending on whether a support or vertex is selected. Shuffle mutation reorders a length of the genome (Figure 5-15). This operator is included because genome ordering is important thus a solution maybe improved by shuffling the genes. Addition mutation adds a random number of new points while deletion removes a random number (although there must always be at least 4 vertex in the genome).

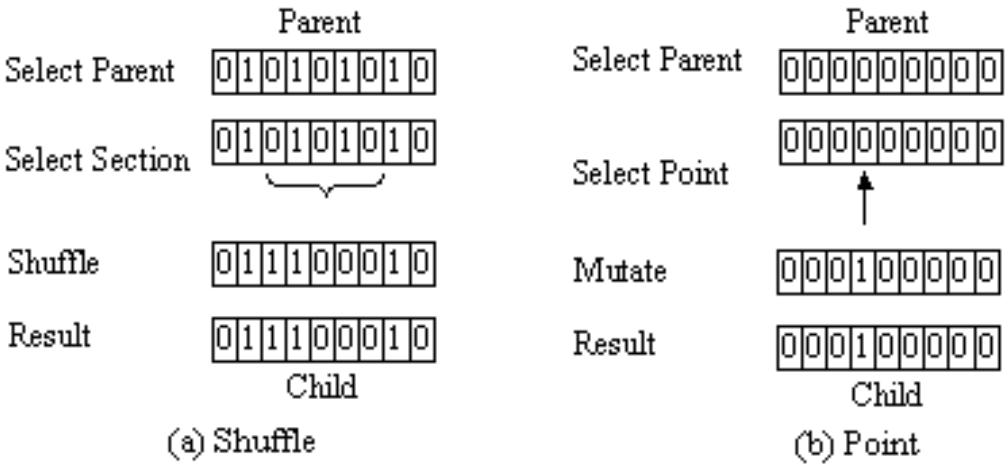


Figure 5-15 Mutation operators for dome design

5.7.7 Selection

This work uses a conventional tournament selection technique (Goldberg, 1989). In tournament selection a predetermined number of individuals are randomly selected from the population and ranked according to fitness, with the fittest individual being chosen. As the tournament size is increased, the selection pressure is increased as it favours the chance of a fit individual being selected.

5.7.8 Fitness function

A fitness function is used by an EA to evaluate how 'good' a particular solution is. This work uses enclosed volume and surface area as its major objectives, which are combined with a structural parameter that seeks to ensure constraints such as allowable buckling, tensile and compressive stresses are not violated (it also includes a weight component).

To search for the optimum number and location of supports the EA initially generates a random number of supports and uses structural weight and stress constraints to guide it. This is because for every additional support there must be at least two additional structural members which increases the overall weight: while the removal of a support increases the loads carried by each remaining structural member which may violate a structural constraint. Both of these scenarios reduce the individual's fitness and hence the algorithm is guided towards an optimum.

Before an individual's fitness can be calculated, the vertices contained in the genome must be converted into a domical structure. This process is accomplished by constructing the genome's convex hull, via the incremental algorithm. Once a convex hull is constructed, its edges become the structural members of the dome. Having built the dome, structural analysis is used to determine whether it performs within the constraints specified above, if not the individual is penalized using a quadratic penalty function (Richardson et al, 1989).

Finally the dome's surface area and volume ratio is determined along with its overall weight. At the end of this process an all individuals are ranked according to the three main criteria (with position 0 being considered the best). An individual's fitness is based upon the cumulative positions by ranking. Therefore this is a minimization problem.

Shea and Cagan (1997) introduced several additional objectives into their fitness function such as an aesthetic value and group penalties that encouraged the evolution of member clusters with the same length or cross sectional area. These objectives have not been included in this work, as the requirement to minimize the surface area to volume ratio encourages the evolution of structures with similar member lengths.

However, there is one important omission from this work that was present in Shea and Cagan's technique: assigning different cross sectional areas to individual members. This work applies one cross sectional area to the whole structure (although it can be modified during the evolutionary process). The genome applied during this work does not consider individual members, as an explicit parameter therefore there is no way of storing individual cross-sectional areas for exchange during the evolutionary process. Geodesic domes aim to have homogeneity with regard to member sizes, so this is not such a major issue.

5.7.9 'Junk' genes

The fitness function does not stipulate that all of the genes contained in an individual's genotype are expressed in the phenotype i.e. not all potential vertices in section 3 of the representation are expressed in the final dome. This is because some potential vertices will be internal to the convex hull and hence not present in the dome. These genes are called 'junk' genes and it is possible for the genome to contain numerous junk genes. To illustrate this concept, consider the convex hull created from a 2D set containing 4 points (Figure 5-16). In Figure 5-16 the convex hull is formed by three vertices, therefore the fourth point is superfluous i.e. a 'junk' gene.

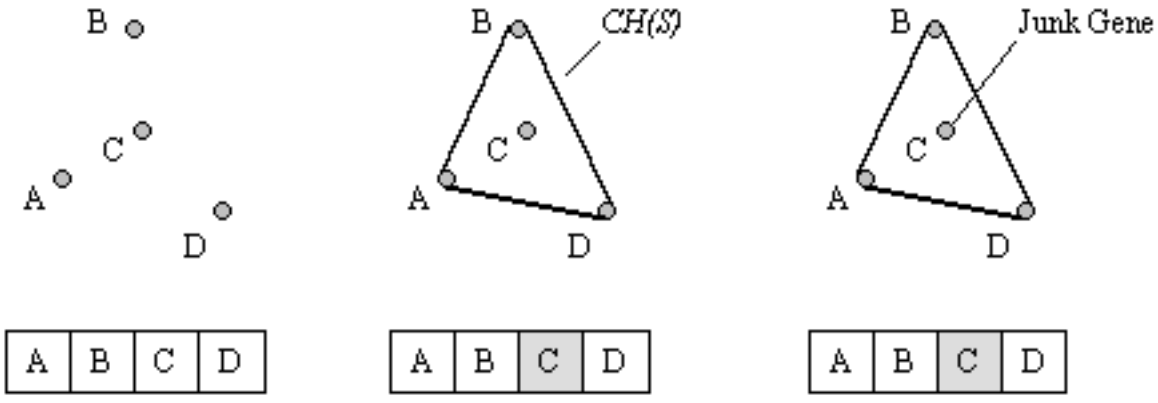


Figure 5-16 Example genome containing a junk gene in dome design

The fitness function does not penalize a solution for having junk genes because they are irrelevant to the phenotype however they do add a significant computational overhead. Unfortunately junk genes cannot simply be removed, because this could potentially cause vital information to be lost. As a compromise, before a solution is evolved all junk genes are identified and a deletion operator applied (each junk gene has a 50% chance of deletion). If this stage is not included, the genome tends to bloat as per genetic programming.

5.8 Illustrative Example

This section provides an illustrative example of the search technique described in this chapter. The aim of the experiment is to evolve a solution that maximizes the enclosed volume while minimizing the surface area at the same time.

5.8.1 Introduction

The following test case was designed to assess search performance, as there are no standard test cases. The EA tableau (Table 5-2) details the values applied to the key evolutionary parameters however it should be noted that no attempt has been made to optimise any parameters related to evolutionary operators.

Table 5-2 EA Tableau for dome design

Objective	Maximise enclosed volume, minimise surface area
Representation	String containing points
Initialisation	Random initialisation (no seeding)
Raw Fitness	Based on: enclosed volume and surface area
Selection	Tournament (size = 3) with Elitism
Major Parameters	P = 1, M = 400, G = 25
Evolutionary Operators:	
$Reproduction_{prob}$	0.1
Mutation operator	Point, shuffle, addition and deletion
$Mutation_{prob}$	0.4
Crossover operator	N point crossover
$Crossover_{prob}$	0.5

5.8.2 Results

The performance graph (Figure 5-17) shows the fitness of the best of generation during the run, while (Figure 5-18) indicates the best layout evolved.

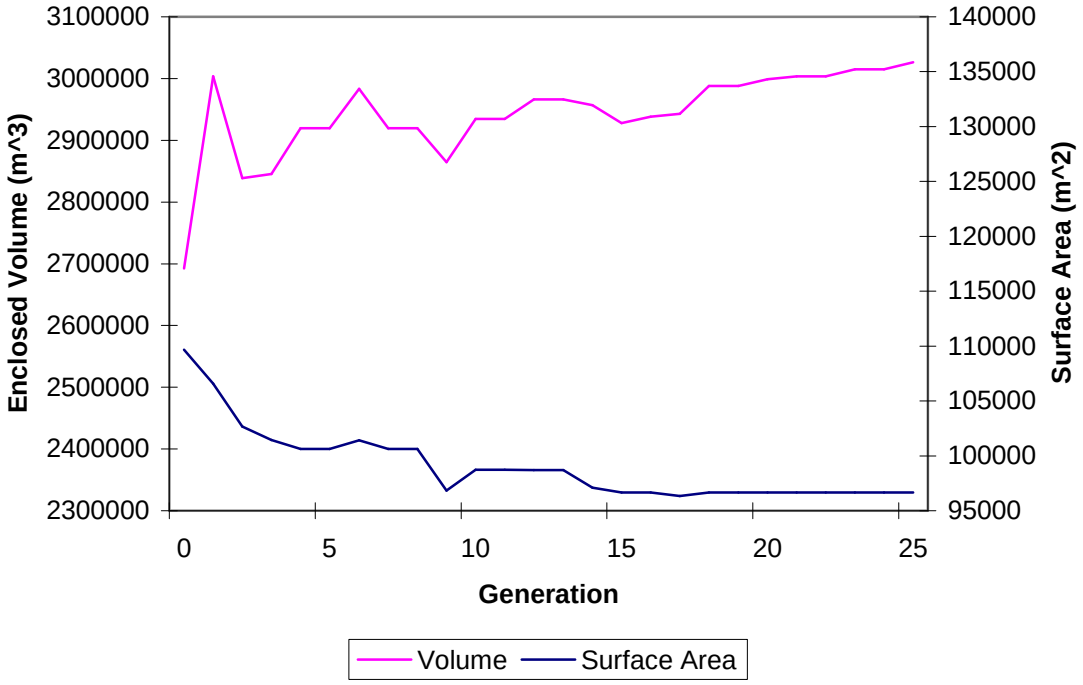


Figure 5-17 Performance graph for dome example

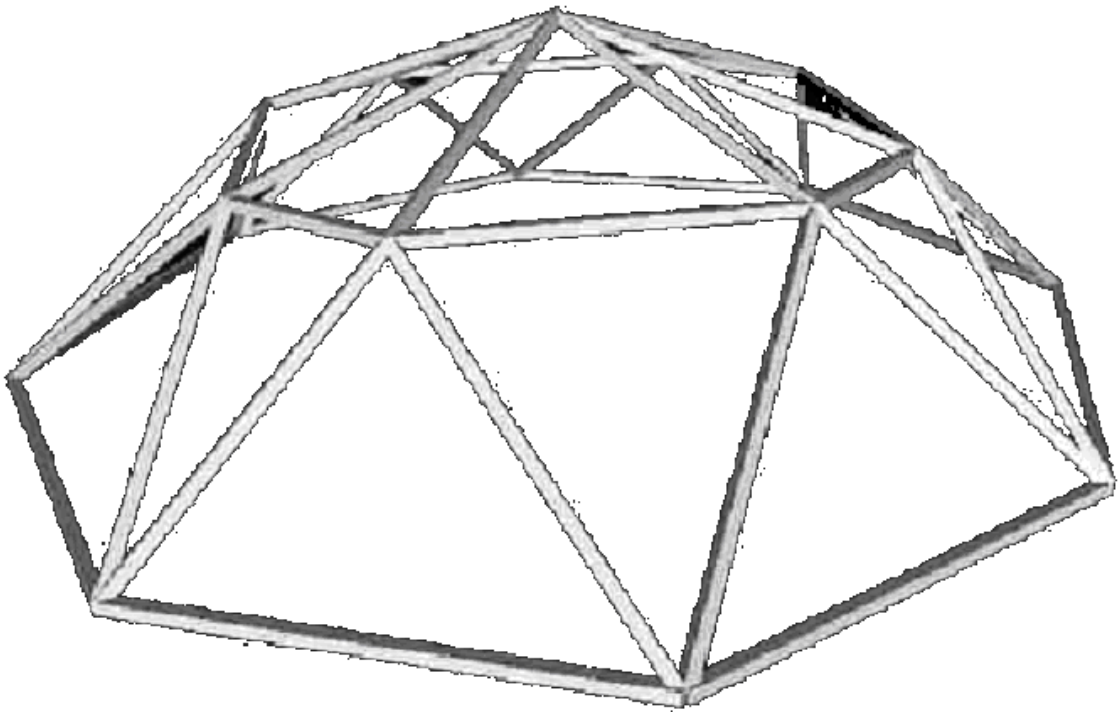


Figure 5-18 Example dome design for illustrative example

5.8.3 Conclusion

This example demonstrates how the proposed representation maybe used to evolve ‘geodesic-like’ domes.

5.9 Conclusions

This chapter demonstrates an EA combined with a convex hull algorithm (incremental algorithm) to create a system capable of designing ‘geodesic-like’ domes directly in 3D. It is shown that this produces viable and efficient structural designs whilst avoiding many of the problem experienced by the previous approach that projected a 2D truss on to a predefined curved surface. However because the vertices section of the genome only contains potential genes, the genome has a tendency to bloat (contain large numbers of superfluous genes).

6 Summary and Future Work

6.1 Introduction

This chapter will consider the key findings, of this thesis, in relation to its original objectives and discuss possible directions for future work. The aim of this work is to investigate how some civil engineering design problems, in particular structures, can be represented in evolutionary algorithms. Many representations have been used in design each with its own strengths and weaknesses: strings are generally used for parameters based problems, voxels for shape discovery, while trees and graphs are used for skeletal structures. Within civil engineering design, the most commonly studied structure is the truss and three main representations have been used, each with their own pros and cons. However in general trees and graphs are the most suited to trusses because they permit the adaptability required for topological design: as strings are linear structures with each element having at most two connections: left and right. Unfortunately, most physical structures contain elements that connect to an arbitrary number of elements. Therefore higher dimensional representations such as trees or graphs have a more appropriate form.

6.2 Summary of Investigative Work Versus Original Objectives

This thesis had two main objectives, each will now be considered.

6.2.1 Investigate existing and develop new representation for orthogonal building design

Chapter 4 considers the conceptual layout design of commercial office buildings. It starts with a review of the existing work in this field, all of which are limited to rectangular floor plans. A 3-section string representation with real encoding is proposed as this ensures column alignment is retained during evolution, while polygon partitioning is used to decompose floor plans. This technique can evolve suitable solutions for orthogonal buildings with atria. This is an improvement over all previous research.

6.2.2 Investigate existing and develop new representation for dome design

Chapter 5 demonstrates an evolutionary algorithm combined with a convex hull algorithm creating a system capable of designing ‘geodesic-like’ domes. However this work will only create domes with geodesic characteristics not true geodesic domes because geodesic

breakdowns are not explicitly enforced. The previous approach projected a 2D truss on to a predefined curved surface losing the key variables of surface area and enclosed volume. This work searches using these variables and produces more 'dome-like' results. However the representation has a tendency to bloat because the vertices in the genome are not guaranteed to be included in the final structure.

6.3 Future Work

This section discusses the possible directions for future work.

6.3.1 Orthogonal building design

While this work proposes a representation capable of solving an orthogonal layout it will not handle an irregular one, therefore this is most obvious area for future development (however this work could form the basis of such a system). One possible approach to consider would be to divide an irregular layout into rectangles and right-angled triangles (rather than simply partitioning a layout in rectangles). Triangular partitions could be represented by a similar genome arrangement to that already described, however the x and y column spacing would only apply to the opposite and adjacent sides. The other major area for improvement is the fitness function. At the present time this assigns a single numerical value to each solution. However if this representation to be used on real world problems, a multi-objective fitness function might be more appropriate.

6.3.2 Dome design

At present this work only considers enclosed volume, surface area and a structural component including structural response of the dome from its weight and applied loads and weight. A more realistic fitness function could include wind loading etc and perhaps incorporate the material used to cover the dome. On a more practical note, the convex hull algorithm could be improved to give O(n log n) performance. To achieve this, the algorithm must maintain a 'conflict graph' indicating which faces are visible (de Berg et al, 1997).

Also the proposed system has only been applied to the design of domes but theoretically it could be used to design any object that is required to have a continuous, convex surface for example aircraft nosecones.

7 References

Atmar W. On the Rules and Nature of Simulated Evolutionary Programming. In: Fogel D.B. and Atmar W. (eds). *Proceedings of the First Annual Conference on Evolutionary Programming*. La Jolla: CA. pp 17-26. 1992.

Ashour A.F, Alvarez L.F. and Toropov V.V. Empirical modelling of shear strength of RC deep beams by genetic programming. *Computers and Structures*. Pergamon. 81. 2003. pp 331-338.

Azid I.A. and Kwan A.S.K. A layout optimisation technique with displacement constraint. In: Topping, B.H.V. and Kumar, B (eds).*Optimization and Control in Civil and Structural Engineering*. Civil-Comp Press: Edinburgh. 1999. pp 71-77.

Babovic V, Drecourt J-P, Keijzer M. and Hansen P.F. A data mining approach to modelling of water supply assets. *Urban Water*. 4. 2002. pp 401-414.

Bäck T, Hammel U. and Schewefel H.-P. Evolutionary Computation: Comments on the History and Current State. *IEEE Transactions on Evolutionary Computation*. 1(1). pp 15-28. 1997.

Baker J.E. Adaptive Selection Methods for Genetic Algorithms. In: Grenfenstett, J.J. (ed), *Proceedings for the First Annual Conference on Genetic Algorithms and their Applications*. 1985.

Baron P, Fisher R, Tuson A, Mill F. and Sherlock A. A voxel-based representation for evolutionary shape optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. 13. 1999. pp 145-156.

Bendsoe M.P. and Kikuchi N. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*. 71, 1998. pp 197-224.

Beyer H.-G. and Schwefel H.-P. Evolutionary Strategies- A comprehensive introduction. *Natural Computing*. 1. pp 3-52. 2002.

Bhattacharya M and Nath B. Genetic Programming: A review of some concerns. In: Alexandrov V.N. (ed) ICCS2001. *Lecture Notes in Computer Science 2074*. Springer-Verlag. 2001. pp 1031-1040.

Borkowski A. and Grabska E. *Representing Designs by Composition Graphs*. In: International Association for Bridge and Structural Engineering (IABSE) Colloquium. Bergamo. IABSE Reports (72). 1995.

Borkowski A, Grabska E., Nikodem P. and Strug B. On genetic search of optimal layout of skeletal structures. In: Schnellenbach-Held M. and Denk H. (eds). *Advances in intelligent computing in engineering*. Proceedings of the 9th International EG-ICE Workshop. Darmstadt. Germany. 2002.

Bradshaw J. and Miles J.C. Using standard fitnesses with genetic algorithms. *Advances in Engineering Software*. 28. 1997. pp 425-435

Camp, C. Pezeshk, S. and Cao, G. Optimized design of two-dimensional structures using a genetic algorithm. *Journal of Structural Engineering*. 1998. pp 551-559.

Coello C.C, Hernandez F.S. and Farrera F.A. Optimal Design of Reinforced Concrete Beams Using Genetic Algorithms. *Expert Systems with Applications*. Elsevier Science. 12(1). 1997. pp 101-108.

Coley D.A. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Publishing. 2003.

Darwin C. *The Origin of the Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*. Murray: London. 1859.

De Berg M. *Computational geometry: algorithms and applications*. New York: Springer. 2000.

De Jong K.A. Genetic algorithms are NOT function optimisers. In: Whitley L.D. (ed). *Foundations of Genetic Algorithms –2*. Morgan Kaufmann. 1993.

Deb, K. *Multi-objective optimization using evolutionary algorithms*. Wiley. 2002.

Deb, K. and Gulati, S. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elements in Analysis and Design*. 37. 2001. pp 447-465.

Dorado J, Rabunal J.R, Puertas J, Santos A and Rivero D. Prediction and modelling of the flow of a typical urban basin through genetic programming. In: Cagnoni S (ed). *EvoWorkshops 2002*. LNCS 2279. Springer-Verlag. 2002. pp190-201.

Dorn, W.C, Gomory, R.E. and Greenberg, H.J. Automatic design of optimal structures. *Journal de M'ecanique*.1964. pp 25-52.

Dumitrescu D, Lazzerini B, Jain L.C and Dumitrescu A. *Evolutionary Computation*. CRC Press: London. 2000.

Dym C.L. *Engineering design: A synthesis of views*. Cambridge University Press. 1994.

Eiben A.E. and Schoenauer M. Evolutionary Computing. *Information Processing Letters*. 82. 2002. pp 1-6.

Eisfeld M. and Scherer R. Assisting conceptual design of buildings by and interactive description logic based planner. *Advanced Engineering Informatics*. 17. 2003. pp 41-57

Fenves S.J, Rivard H. and Gomez N. Conceptual structural design in SEED. *Journal of Architectural Engineering*. 1(4). 1995. pp 179-186

Finger S. and Dixon J.R. *A review of research in mechanical engineering design*. Part 1: Descriptive, prescriptive and computer-based models of design process, research in engineering design. 1 (1). 1989. pp 51-67.

Fogel D.B. Phenotypes, genotypes and operators in evolutionary computation. In: *IEEE International Conference on Evolutionary Computation*. 1995.

Fogel D.B. *Evolutionary Computation: The Fossil Record*. IEEE Press: Piscataway. 1998.

Fogel D.B. *Evolutionary Computation (Towards a New Philosophy of Machine Intelligence*. IEEE Press: NY. 2000.

Fogel L.J. Autonomous automata. *Industrial Research*. 4. 1962. pp 14-19.

Fuyama H, Law K.H. and Krawinkler H. An interactive computer assisted system for conceptual structural design of steel buildings. *Computers and Structures*. 63(4). 1997. pp 647-662

Goldberg D.E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing. 1989.

Grew R.J. The use of a knowledge based system as an aid in the preliminary design of building structures. In Topping B.H.V. (ed) *Developments in Artificial Intelligence for Civil and Structural Engineering*. Civil-Comp Press. 1995. pp 177-182

Grierson D.E. and Khajehpour S. Method for conceptual design applied to office buildings. *Journal of Computing in Civil Engineering*. 16(2). 2002. pp 83-103

Grierson D.E. and Pak W.H. Optimal sizing, geometrical and topological design using genetic algorithms. *Journal of Structural Optimisation*. 6. 1993. pp 151-159.

Griffiths D.R. and Miles J.C. Determining the Optimal Cross Section of Beams. In: Topping B.H.V (ed). *Proceedings of the Seventh International Conference on The Application of Artificial Intelligence to Civil and Structural Engineering*. Paper 36. Civil-Comp Press. 2003.

Hajela, P. and Lee, E. Genetic algorithms in truss topological optimization. International Journal. 32(22). 1995. pp 3341-3357.

Harty N. and Danaher M. A knowledge-based approach to preliminary design of buildings. Structural Board Paper 10312. *Proceedings of the Institution of Civil Engineers, Structures and Buildings*. 104. 99. pp 135-144

Holland J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press: Ann Arbor. 1975.

Hong Y.S. and Bhamidimarri R. Evolutionary self-organising modelling of a municipal wastewater treatment plant. *Water Research.* 37. 2003. pp 1199-1212.

Howard D. and Roberts S.C. The prediction of journey times on motorways using genetic programming. In: Cagnoni S et al (ed). *EvoWorkshops 2002*. LNCS 2279. Springer-Verlag. 2002. pp 210-221.

Hudson M.G. and Parmee I.C. The application of genetic algorithms to conceptual design. In: Sharpe J (ed). *AI system support for conceptual design*. Proceedings of the 1995 Lancaster University Workshop on Engineering Design. Springer-Verlag. 1995. pp 17-36.

Ishino Y. and Jin Y. Estimate design intent: a multiple genetic programming and multivariate analysis based approach. *Advanced Engineering Informatics*. 16. 2002. pp 107-125.

Jefferson D, Collins R, Cooper C, Dyer M. Flowers M, Kort R, Taylor C. and Wong A. Evolution as a theme in artificial life: The genesys/ tracker system. In: Langton C.G. and Farmer D. (eds). *Artificial life II.* Addison-Wesley. 1990.

Kane C. and Schoenauer M. Topological optimum design using genetic algorithms. *Control Cybernet*. 25(5). 1996. pp 1059-1088.

Keijzer M. and Babovic V. Dimensionally aware genetic programming. In: Banzhaf W. *Proceedings of the Genetic and Evolutionary Computation Conference*. July 13-17. Orlando (USA). 1999.

Khajehpour S. and Grierson D.E. Profitability versus safety of high-rise office buildings. *Journal of Structural and Multidisciplinary Optimisation*. 25. 2003. pp 1-15

Khajehpour S. and Grierson D.E. Filtering of Pareto-Optimal trade-off surfaces for building conceptual design. In Topping B.H.V. and Kumar B. (eds). *Optimzation and control in Civil & Structural Engineering*. Civil-Comp Press. Edinburgh UK. 1999. pp 63-70

Kirkpatrick S, Gerlatt C. D. Jr, and Vecchi M.P. Optimization by Simulated Annealing. *Science*. 220. 1983. pp 671-680.

Kirsch, U. On singular topologies in optimum structural design. *Structural Optimization*. 72. pp 133-142.

Kojima F, Kubota N and Hashimoto S. Identification of crack profiles using genetic programming and fuzzy inference. *Journal of Materials Processing Technology*. Elsevier. 108. 2001. pp 263-267.

Köppen M. and Nickolay B. Design of image exploring agent using genetic programming. *Proceedings of IIZUKA'96*. Japan. 1996. pp 549-552.

Koza J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press:Cambridge. 1992.

Lawson B. *How designers think: The design process demystified*. Architectural Press. 1997.

Laszlo M.J. *Computational Geometry and Computer Graphics in C++*. Prentice Hall. 1996.

Lee D.G, Lee B.W. and Chang S.H. Genetic programming model for long-term forecasting of electric power demand. *Electric power systems research*. Elsevier. 40. 1997. pp17-22.

Lewontin R.C. *The Genetic Basis of Evolutionary Change*. Columbia University Press: NY. 1974.

Matous K, Leps M, Zeman. J. and Sejnoha M. Applying genetic algorithms to selected topics commonly encountered in engineering practice. *Computer Methods in Applied Mechanics and Engineering*. 190. 2000. pp 1629-1650.

Michaelewicz Z. *Genetic algorithms + Data Structures = Evolutionary Programming*. Springer-Verlag:Berlin. 1999.

Miles J.C. and Moore C.J. *Practical Knowledge Based Systems in Conceptual Design*. Springer-Verlag. 1994.

Miles J.C, Sisk G.M. and Moore C.J. The conceptual design of commercial buildings using a genetic algorithm. *Computers and Structures*. 79. 2001. pp 1583-1592

Mitchell, A.G.M. The limits of economy of material in frame structures. *Philosophical Magazine*. 8(47). 1904. pp 589-597.

Moran T.P. and Carroll J.M. Overview of Design Rationale. In: Moran T.P. and Carrol J.M. (eds). *Design Rationale Concepts, Techniques and Use*. A Computers, Cognition and Work Publication. 1996. pp 1-19.

Montana D.J. and Czerwinski S. Evolving control laws for a network of traffic signals. *Proceedings of the Firs Annual Conference: Genetic Programming*. July 28-3. Stanford University. 1996. pp 333-338.

Motro R. Review of the development of geodesic domes. In: Makowski Z.S. (ed). *Analysis, design and construction of braced domes*. Cambridge University Press. 1994. pp 387-412.

O'Rourke J. *Computational Geometry in C*. Second Edition. Cambridge University Press. 1998.

Pahl G. and Beitz W. *Engineering design: A systematic approach*. Springer-Verlag. 1996.

Pareto V. *Cours D'Economie Politique*. Switzerland: Lausanne. 1896.

Parmee I.C. Evolutionary and adaptive strategies for efficient search across whole system engineering design hierarchies *AIEDAM*. 12. 1998. pp 431-445

Park K-W, and Grierson D.E. Pareto-Optimal Conceptual Design of the Structural Layout of Buildings Using a Multicriteria Genetic Algorithm. *Computer-Aided Civil and Infrastructure Engineering*. 14. 1999. pp 163-170.

Pedersen P. Optimal joint positions for space trusses. *Proceedings of the American Society of Civil Engineers.* 99(ST12). 1973. pp 2459-2475.

Porter B, Mohamed S.S. and Crossley T.R. Genetic computation of geodesics on three-dimensional curved surfaces. *Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95)*. IEEE Conference Publication No.414. University of Sheffield (UK), 12-14th September. 1995. pp 448-453.

Rafiq M.Y. and Southcombe C. Genetic algorithms in optimal design and detailing of reinforced concrete biaxial columns supported by a declarative approach for capacity checking. *Computers and Structures*. 69. 1998. pp 443-457.

Rafiq M.Y, Bugmann Y. and Easterbrook D.J. *Building concept generation using genetic algorithms integrated with neural networks*. *Proceedings of AI in Structural Engineering, IT for Design, Manufacturing, Maintenance and Monitoring*. EG-SEA-AI. Wierzba, Poland. 1999. pp 165-173

Rajan, S.D. Sizing, shape and topology design optimization of trusses using genetic algorithm. *Journal of Structural Engineering*. 1995. pp 1480-1487.

Rajeev S. and Krishnamoorthy C.S. Genetic algorithm-based methodology for design optimisation of reinforced concrete frames. *Computer-Aided Civil and Infrastructure Engineering*. 13. 1998. pp 63-74

Rajeev, S. and Krishnamoorthy, C.S. Genetic algorithms-based methodologies for design optimization of trusses. *Journal of Structural Engineering*. 1997. pp 350-358.

Rechenberg I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog: Stuttgart. 1973.

Richardson J.T, Palmer M.R, Liepins G. and Hilliard M. Some guidelines for genetic algorithms with penalty functions. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann. 1989. pp 191-197.

Robert M, Reiss M. and Monger G. *Biology (Principles and Processes)*. Thomas Nelson and Sons. 1993.

Roberts S.C. and Howard D. Detection of incidents on motorways in low flow high speed conditions by genetic programming. In: Cagnoni S et al (ed). *EvoWorkshops 2002. LNCS 2279*. Springer-Verlag. 2002. pp245-254.

Rosenmann M. The generation of form using evolutionary approach. In: Dasgupta D and Michalewicz Z (eds). *Evolutionary algorithms in engineering applications*. Springer. 1997. pp 69-86.

Rudolph G. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*. 5 (4). 2001. pp 410-414.

Ruy, W.S. and Yang, Y.S. Topology design of truss structures in a multicriteria environment. *Computer-Aided Civil and Infrastructure Engineering*. 16. 2001, pp 246-258.

Sahab M.G, Ashour A.F. and Toropov V.V. A hybrid genetic algorithm for reinforced concrete flat slab buildings. *Computers and Structures*. 83. 2005. pp 551-559

Schwefel H-P. *Evolutionsstrategie und numerische Optimierung*. PhD Thesis. TU Berlin. Germany. 1975.

Shamos M.I. *Computational Geometry*. PhD Thesis. Yale University, New Haven, UMI #7819047. 1978.

Shea K. and Cagan J. Innovative dome design: Applying geodesic patterns with shape annealing. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. 11. 1997. pp 379-394.

Shrestha S.M. and Ghaboussi J. "Evolution of optimum structural shapes using genetic algorithm". *Journal of Structural Engineering.* ASCE. 124(11). 1995. pp1331-1338.

Sisk G.A. The use of a GA-Based DSS for Realistically Constrained Conceptual Building Design. Cardiff University. PhD Thesis. 1999.

Sisk G.M, Miles J.C. and Moore C.J. Designer Centered Development of GA-Based DSS for Conceptual Design of Buildings. *Journal of Computing in Civil Engineering*. ASCE. 17(3). 2003. pp 159-166.

Soibelman L. and Pena-Mora F. Distributed multi-reasoning mechanism to support conceptual structural design. *Journal of Structural Engineering*. 126(6). 2000. pp 733-742

Stiny G, Gips J. "Shape Grammars and the Generative Specification of Painting and Sculpture". In C V Freiman (ed). Proceedings of IFIP Congress71, Amsterdam: North-Holland 1460-1465. Republished in O R Petrocelli (ed), The Best Computer Papers of 1971: Auerbach, Philadelphia. 1972. pp 125-135.

The Buckminster Fuller Institute. [www] <URL:http://www.bfi.org/domes/>. [Accessed 20 May 2005].

Turban E. *Decision Support and Expert Systems: Managerial Perspectives*. Macmillan. 1998.

Ullman D.G, Stauffer L.A. and Diettrich T.G. Preliminary Results of an Experimental Study of the Mechanical Design Process. In: Waldron M.B. (ed). *NSF Workshop of the Design Process*. Ohio State University. pp 143-188.

Wang S.Y. and Tai K. Graph representation for structural topology optimization using genetic algorithms. *Computers and Structures*. 82. 2004. pp 1609-1622.

Yang Y. and Soh C.K. Automated optimum design of structures using genetic programming. *Computers and Structures*. 80. 2002. pp 1537-1546.

Watson A.H. and Parmee I.C. Systems identification using genetic programming. *Proceedings of ACEDC'96*. 1996.

Watson A.H. and Parmee I.C. Improving engineering design models using an alternative genetic programming approach. *Proceedings of International Conference on Adaptive computing in design and manufacture*. 1998. pp 193-206.

Weisstein E.W. "*Archimedean Solid*", From *MathWorld*--A Wolfram Web Resource, [www] <URL: http://mathworld.wolfram.com/ArchimedeanSolid.html>, [Accessed 20 May 2005].

Weisstein E.W. Circle point picking. In: *MathWorld*--A Wolfram Web Resource, [www] <URL: http://mathworld.wolfram.com/CirlePointPicking.html>, [Accessed 20 May 2005].

Weisstein E.W. Convex Hull. In: *MathWorld*--A Wolfram Web Resource, [www] <URL: http://mathworld.wolfram.com/ConvexHull.html>, [Accessed 20 May 2005].

Whigham P.A. and Crapper P.F. Modelling rainfall-runoff using genetic programming. *Mathematical and Computer Modellin*. 33. 2001. pp 707-721.

Wolpert D.H. and Macready W.G. No free lunch theorems for optimisation, *IEEE Transactions on Evolutionary Computation*. 1(1). 1997. pp 67-82.

Zhang Y. and Miles J.C. Representing the problem domain in stochastic search. In: Schnellenbach-Held M. and Hartmann M. (eds). *Next generation- intelligent systems in Engineering*. 2004. pp 156-168.