Methodology for identifying alternative solutions in a population based data generation approach applied to synthetic biology

A thesis submitted as partial fulfillment of the requirement of Doctor of Philosophy (Ph.D.)

by

Yasoda Jayaweera

Computer Science Department

Brunel University London

—- —-

Abstract

Design is an essential component of sustainable development. Computational modelling has become a useful technique that facilitates the design of complex systems. Variables that characterises a complex system are encoded into a computational model using mathematical concepts and through simulation each of these variables alone or in combination are modified to observe the changes in the outcome. This allows the researchers to make predictions on the behaviour of the real system that is being studied in response to the changes. The ultimate goal of any design process is to come up with the best design; as resources are limited, to minimize the cost and resource consumption, and to maximize the performance, profits and efficiency. To optimize means to find the best solution, the best compromise among several conflicting demands subject to predefined requirements. Therefore, computational optimization, modelling and simulation forms an integrated part of the modern design practice.

This thesis defines a data analytics driven methodology which enables the identification of alternative solutions of computational design by analysing the generational history of the population based heuristic search used to generate the templates. While optimisation is focused on obtaining the optimal solution this methodology focuses on alternative solutions which are sub optimal by fitness or solutions with similar fitness but different structures. When the optimal design solution is less robust, alternative solutions can offer a sufficiently good accuracy and an achievable resource requirement. The main advantage of the methodology is that it exploits the exploration process of the solution space during a single run, by focusing also on suboptimal solutions, which usually get neglected in the search for an optimal one. The history of the heuristic search is analysed for the emergence of alternative solutions and evolving of a solution. By examining how an initial solution converts to an optimal solution core design patterns are identified, and these were used to improve the design process. Further, this method limits the number of runs of the heuristic search as more solution space is covered. The methodology is generic because it can be used to any instance where a population based heuristic search is applied to generate optimal designs. The applicability of the methodology is demonstrated using three case studies from mathematics (building of a mathematical function for a set target) and biology (obtaining alternative designs for genomic metabolic models [GEM] and DNA walker circuits). In each case a different heuristic search method was used: Gene expression programming (mathematical expressions), genetic algorithms (GEM models) and simulated annealing (DNA walker circuits). Descriptive analytics, visual analytics and clustering was mainly used to build the data analytics driven approach in identifying alternative solutions. This data analytics driven methodology is useful in optimising the computational design of complex systems.

Acknowledgements

First and foremost, I would like to offer my sincere gratitude to my supervisor, Professor David Gilbert, for his continuous guidance and support in my studies, for his immense patience, motivation, enthusiasm, and vast knowledge. The completion of my thesis would not have been possible without the support and nurturing of my supervisor. Thank you very much Professor David!

I would also like to thank Dr Alessandro Pandini, Professor Monika Heiner, Dr Steven Swift, Dr Crina Grosan and Professor Juris Viksna for their advice, encouragement, insightful comments, and difficult questions. A special thank you to my colleague, Bello Suleiman for providing me with data for part of my research.

Also, I would like to thank my family for supporting me throughout the course of my PhD studies. Last but not least, I would like to remember Thuru, without whom I wouldn't have started a PhD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Modelling of Complex Systems

Design is an essential component of sustainable development. A good design ensures the object (or the system) being built can perform its intended tasks, has a proper mechanism of handling errors and exceptions and can accommodate requirements which could arise in the future. Thus, a designer is required to have thorough domain knowledge and the ability to foresee all the potential opportunities and threats to ensure a robust design. However, as the complexity of the system increases it becomes impossible for a human designer to gain sufficient insight into the system so as to provide better design and accurate predictions. Therefore, computational modelling has become a useful technique that facilitates the design of complex systems.

Computational modelling is the use of computers to simulate the behaviour of complex systems. Numerous variables that characterises a complex system are encoded into a computational model using mathematical concepts. During simulation each of these variables alone or in combination are modified to observe the changes in the outcome. This allows the researchers to make predictions on the behaviour of the real system that is being studied in response to the changes. One of the main advantages of this method is that it allows the scientists to experiment with endless permutations and combinations of input parameters that have been otherwise impossible to make out by a human experimenter. Thus, computational modelling can expedite research by facilitating scientists to conduct as many simulated experiments as required by computer saving time, money and material. Synthetic biology is one of the novel areas that is benefited by computational modelling (Andrianantoandro et al. 2006).

The ultimate goal of any design process is to come up with the best design; as resources are limited, to minimize the cost and resource consumption, and to maximize the performance, profits and efficiency. To optimize means to find the best solution, the best compromise among several conflicting demands subject to predefined requirements. Mathematical optimization has been extremely successful as an aid to better decision making. Therefore, computational optimization, modelling and simulation forms an integrated part of the modern design practice.

1.2 Computational Design in Synthetic Biology

A human is a multicellular organism. A human body is made up of over 30 trillion cells. A human body has levels of organisations built-in on top of these cells. For example a group of specialised cells make up tissues, specialised tissues make up organs and organs make up organ systems. Respiratory system, digestive system, nervous system, skeletal system and reproductive systems are some of the organ systems that have dedicated functionalities. Such complex network of biologically relevant entities (i.e. integrated activity of organs) is referred to as a biological system. For example the respiratory system consists of five organs namely pharynx, larynx, bronchi, lungs and diaphragm, and the integrated activity of these organs help to achieve the main functionality of the respiratory system; to breathe in oxygen and breathe out carbon dioxide. Similarly single cell organisms have biological systems. For example there are protein-bound organelles in the cytoplasm which in-coordination carries out bacterial metabolism. Bacterial metabolism is another example of a biological system.

System biology is the computational and mathematical modelling of these complex biological systems. As Hiroaki Kitano says it is important to study the structure of a system structure and how it is related to its dynamics (Kitano 2002). Because a system is not just the assembly of its components (i.e. organs or genes and proteins) but how the components are assembled and how they interact with each other. According to Kitano, four main properties can be derived from a system-level understanding of a biological system.

1. System structure – gene interactions and metabolic pathways in intercellular and multicellular structures
2. System dynamics – how the system behaves over time under various conditions
3. The control method – mechanisms that systematically control the state of the cell
4. The design method - Strategies to modify and construct biological systems having desired properties can be devised based on definite design principles and simulations, instead of blind trial-and-error

The Human Genome Project (Collins et al. 1998) is one such example of an effort taken to obtain a system level understanding of the human body. Study of single cell organisms such as bacteria are more popular in systems biology because of the smaller genome. Another example is sequencing of the complete genome of Escherichia coli MG1655 bacteria strain (Edwards and Palsson 2000) and the development of the computational model of E. coli bacterium's metabolic activity by the Passon's group (Feist et al. 2007).

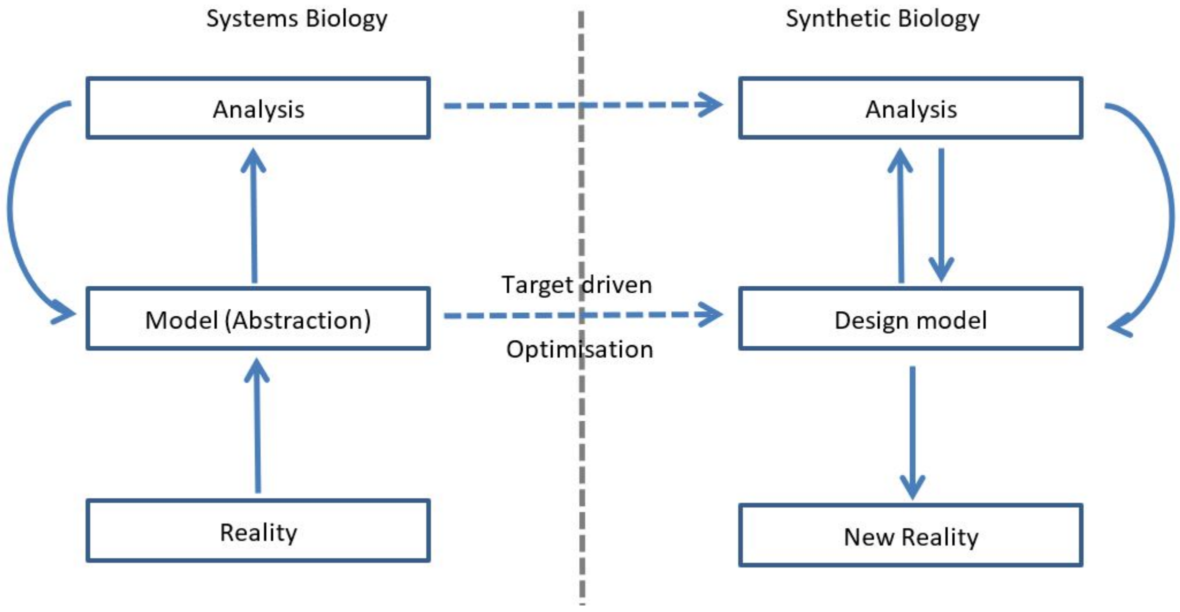


Figure 1.1: Data analytics driven design - This diagram is an illustration of design completed in various stages of synthetic and systems biology

In the process diagram presented in 1.1 the left hand side represents the systems biology aspect of my thesis. Through my work I have looked at two types of realities; E. coli K 12 MG1655 bacteria and a DNA Walker circuit. As mentioned above E. coli K 12 MG1655 bacteria is the strain sequenced by the Passon's group. A computational model was developed by the same group to depict the metabolic activity of the bacteria. This stochastic computational model was simulated in the computer using a special program called Snoopy. The simulation resulted in behaviour (reaction rates and concentration of metabolites). In my Master's dissertation I have analysed the structure (genes, metabolites and reactions) and behaviour of these models to identify similarities between the models and group them using cluster analysis.

The second reality is DNA walker circuits. DNA walker circuit discussed in this thesis was obtained from the work published by the Gilber's group (Gilbert, Heiner, and Rohr 2018). The DNA walker circuit presented the paper is designed to evaluate a binary expression. Walker circuit behaviour was simulated using simulation software called Marcie. Further information on DNA walker circuits will be discussed in Chapter 7.

The most important aspect of systems biology is that the system-level understanding of a biological system paves the road for the ambitious to alter an existing reality and construct new realities. This is where synthetic biology comes into play. Synthetic biology can be simply defined as designing and fabrication of biological components and systems for useful purposes. Today synthetic biology (including genetic engineering) underlies a multi-billion dollar industry offering solutions to some of the most intractable problems. The ability to insert new combinations of genetic material into (or remove unfavourable genetic information from) micro-organisms, animals (Robl 2002) and plants (Yau and Stewart 2013), (Jiang et al. 2013) offers novel ways to produce valuable small molecules into proteins; provides the means to produce plants (Li et al. 2012) and animals (Wall et al. 2005) that are disease resistant; tolerant of harsh environments, and have higher yields of useful products (Nielsen 2001); and provides new methods to treat and prevent human diseases.

There are two main disciplines in synthetic biology; the design and fabrication of biological components and systems that do not already exist in the natural world (creating artificial life) and the re-design (Fuentes et al. 2016) and fabrication of existing biological systems (Benner and Sismour 2005). For example in order to synthetically produce Artemisinin the scientists built a new metabolic pathway in yeast by assembling 10 genes from 3 organisms. This attempt is an example of re-designing of existing biological systems (Yeast) [see (Paddon and Keasling 2014) for further details on synthetic biology].

With the recent advancement in genome editing techniques such as CRIPR modification of biological systems have become much easier. However the main problem is to identify the correct and novel sequences of gene that should be removed, replaced or added. For example a single cell E. coli bacterium consists of 4000 genes. If we were to filter the most suitable combination of genes responsible for the optimal production of bio products, we would have to test all possible combinations of 4000 genes. This large solution space (4000 factorial of possible combinations) makes blind trial and error impossible with regards to time and resources, and it would be a waste to explore each and every possible combination. This problem can be solved by incorporating computational methods such as simulations and data analytics.

In systems biology we create an abstraction of a model that could be simulated in a computer. This model could be used to run experiments. Model simulation has several advantages. Modifications could be done easily and behaviour could be monitored instantly. This method is inexpensive in terms of lab resources required to modify and grow the bacteria in the lab. Further data analytics and machine learning techniques such as optimisation gives novel avenues to explore a large solution space.

1.3 Motivation

However, the extensive level of complexity of real-work systems often makes it difficult to accurately characterise a system into a perfect computational model and the non-linearity in correlations makes it difficult to adequately capture them by optimization tools. Consequently, there is a gap between approximations delivered by optimization and their practical possibilities. In certain instances, the optimal design proposed by the optimisation tool typically involves implementation impracticalities in predicted parameters making the optimal design less robust (i.e. too many modifications, impossible alterations, highly time and resource consuming alterations). In such cases it is considerate to settle for robust suboptimal design options with sufficiently good accuracy and reasonable resource expenditure. Hence alternative solutions with optimal (similar behaviour and different structures) or suboptimal behaviour play a vital role in the design process.

Computational optimisation is a widely used technique to search a large solution space in order to find the optimal solution that delivers the required behaviour. The process begins with the random solution or a generation of solutions. Then the solution(s) are expressed and the fitness of each individual is evaluated. The individuals are modified to improve their fitness. This process is repeated for a certain number of generations or until a solution has been found.

An optimisation approach is focused on finding the best solution. However, an optimisation approach comes across several intermediary solutions which are sub optimal in fitness during the continuous improvement cycle that runs from the initial solution until the program reaches the optimal solution. But these solutions get discarded at the end. Hence, the history of an optimisation program holds valuable information in identifying potential alternative solutions. I am proposing an approach where the history of an optimisation approach is saved and analyzed in order to find alternative solutions and improve computational design.

Therefore, analysing the history of an optimisation approach is essential in identifying these alternative solutions. For this purpose, data analytics techniques are applied. Generation history from optimisation and characteristics of individual solutions could be used to gain insights on the solution space explored by the optimisation tools. This amounts to a large volume of data generated over a short period of time with different data types (i.e. real data, nominal, ordinal, time series, graphs and etc). identifying alternative solutions may involve performing descriptive statistics, structural comparisons, groupings, pattern recognition, solution profiling and etc depending on the solution type. In order to analyse these large amounts of data efficiently and accurately, it is essential to incorporate data analytics techniques.

Therefore the research question I intend to answer through my research is "When using computation optimisation to improve system design, how can alternative solutions be efficiently found by analysing the history of the optimisation approach?"

I intend to look at three different example scenarios where different optimisation approaches are applied for improving system design. The history of the optimisation program will be stored and analysed in identifying alternative solutions.

1.4 Contributions

Contributions of this thesis are the following:

- **A general methodology to identify alternative designs by analysing the optimisation history of population based heuristic search for computational models**

  In order to address the challenges mentioned above a novel data analytics driven methodology is defined in this thesis. The methodology enables the identification of alternative solutions of computational designs by analysing the generational history of the population based heuristic search used to generate the designs.

  The methodology is composed of three main phases. In the first phase a mathematical model which depicts the behaviour of the whole or part of a complex system is defined. In the second phase, computational optimisation methods are used to generate designs which optimises a pre-defined target behaviour. The choice of optimisation method depends on the model characteristics. Optimisation archive is generated in this phase. The archive stores all the solutions which were generated throughout the optimisation process. These populations of solutions are ordered by the order of appearance and generation. In the third phase the optimisation archive will be analysed using data analytics techniques to obtain three main insights, namely, extracting alternative solutions (behaviourally similar but structurally different) by solution profiling, core model enhancement by examining the evolution of a particular solution over time in a generation, and formalizing model constraints.

The proposed methodology is generic because it can be used to any instance where a population based heuristic search is applied to generate optimal designs for complex computational models. The methodology was applied to three problem areas, which were represented using a three different data structures; i) binary trees (producing a mathematical function for a set target), ii) sequences (optimising designs for genomic metabolic models of bacteria) and iii) graphs (producing optimal DNA walker circuits). In each case a different heuristic search method was used: Gene expression programming (arithmetic expressions), genetic algorithms (GEM models) and simulated annealing (DNA walker circuits). Descriptive analytics, visual analytics and clustering was mainly used to build the data analytics driven approach in identifying alternative solutions. The methodology is explained in detail in Chapter 3.

The application areas covered in this thesis are mainly from synthetic biology. According to the synthetic biology design approach described by Henier & Gilbert (Heiner, Gilbert, and Donaldson 2008), computational models that depict the desired behaviour need to be physically constructed in order to verify its behaviour and the computational approach which was used to generate the design (Figure 1.2 depicts how computational design is applied in synthetic biology). Information obtained from physical implementation is then used to improve the computational design process. Therefore alternative computational solutions (design) can be used to drive the engineering of alternative physical designs. In this case the [alternative] solution acts as an [alternative] design template (or blueprint).

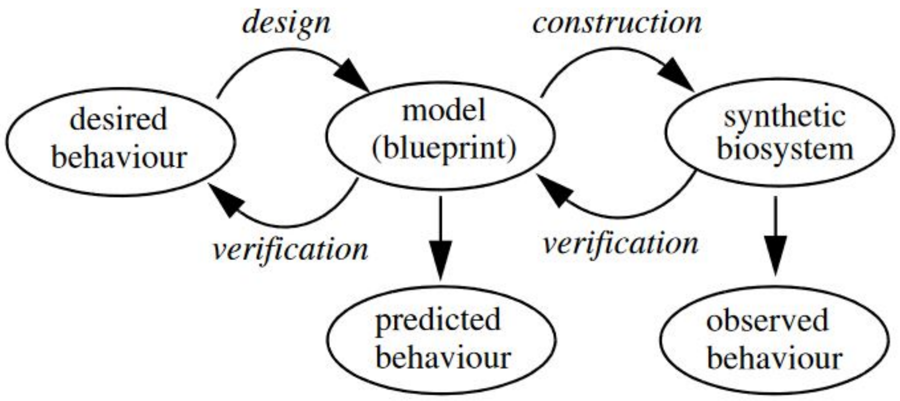


Figure 1.2: The role of modeling in synthetic biology - The diagram represents the connection between design and construction of bio-systems. Firstly, blueprints of systems with a predicted behaviour are created and tested. Next bio-systems are constructed using the blueprints. Finally, the observed behavior of these physical systems are used to verify and improve designs.

- **Application of the methodology to identify alternative tree structures in a population of arithmetic expressions generated using Gene Expression Programming**

  The proposed methodology was applied to extract alternative solutions from a population of arithmetic expressions. An optimisation program called gene expression programming (GEP) was used to generate arithmetic expressions for a predefined target with a predefined set of operands and operators (four operators were used: addition, division, multiplication and division). Arithmetic expressions were represented using a binary tree data structure. Solutions generated during different runs of the optimisation process were compiled into a population of solutions. A pair-wise structural similarity distance inspired by tree edit distance was used to compare the tree structures and cluster to identify alternative solutions. Application of the methodology to arithmetic expression and analysis is described in detail in Chapter 4.

- **Application of the methodology to identify alternative GEM models in a population of metabolic network models generated using Genetic algorithms**

  In this scenario the methodology is applied to genome metabolic models (GEM) of *E.coli* bacteria which simulates the metabolic activities of the *E.coli* bacteria. The model is originally a biochemical network model however, it can be depicted as a graph. A genetic algorithm optimisation technique was used to generate designs which optimised the predefined target behaviour. During the second phase of the methodology, solutions for each generation were stored in the optimisation archive. A GEM model composes of three main data categories, namely, genes, reactions and metabolites which is correlated in multiple levels. However, for the analysis carried out in this thesis the data was considered in a sequential level. GEM models were compared pair-wise. A pair-wise similarity measure was defined and computed between pair of models based on the gene and reaction composition of the two models. For example, the number of common genes between two models or number of common reactions between two models. For models A and B the similarity could be defined as $(A \cap B)/MAX(|A|, |B|)$. Application of the methodology and analysis of the solution populations is described in detail in Chapter 5.

- **Application of the methodology to identify alternative DNA circuit layouts in a population of circuit graphs generated using simulated annealing**

  In this example the methodology is used to analyse a population of DNA circuits. A DNA circuit is represented as a graph data structure which is consisted of nodes and arcs laid out on a Cartesian plane. The DNA circuit discussed in this thesis evaluates a simple logic expression. Simulated annealing optimisation technique is used generate circuit layouts which optimises (reduces) the area and (information) leaks in the circuit. Simulated annealing program was implemented in order to generate the solutions. Optimisation archive stores solutions generated in each run along with the iteration number. In the analysis stage the structures of circuits are compared using a novel comparison method inspired by the RMSD method to identify structurally similarities. Next the solutions will be clustered based on the similarity score to identify alternative solutions. Four different versions of DNA circuit named Toy, Toy0, Toy1 and Toy2 are examined in this section. Application of the methodology and analysis of the solution populations is described in detail in Chapter 6.

- **Software suite used for the data generation and analysis purposes**

  Following is the list of software used and implemented for the data generation and analysis of each application which is described in above points 2 through 4. The software suite can be found in the Github repository at `https://github.com/yasodaj/AlternativeStructures`.

  - **Processing of arithmetic expressions**
    - Gene Expression Programming optimisation program developed using Java to automatically generate arithmetic expressions
    - Python code for comparing the similarity between two tree structures for mathematical expressions
    - R code for clustering of arithmetic expressions and generating and visualising summary statistics

    A detailed description is available in Chapter 4.
  - **Processing of GEM models**
    - R code to analyse and visualise the commonality in genes and reactions in GEM models

    A detailed description is available in Chapter 5.
  - **Processing of DNA walker circuits**
    - Simulated Annealing program to automatically generate DNA circuit layouts using Java

* Java program to validate a given DNA circuit layout
* Java program to compute the leaks and area of a given DNA circuit layout
* R program to plot DNA circuits using a node-arc connection table
* Java program to structural compare two DNA circuit layouts using the RMSD minimisation algorithm
* R code for clustering of DNA circuits and generating and visualising summary statistics

A detailed description is available in Chapter 6.

1.5 Publications

- Towards dynamic genome-scale models

Gilbert, D., Heiner, M., Jayaweera, Y. and Rohr, C., 2019. Towards dynamic genome-scale models. Briefings in bioinformatics, 20(4), pp.1167-1180.

This analysis was carried out for a paper titled "Towards dynamic genome-scale models " and the research was led by Prof. David Gilbert, Prof.Monika Heiner and Christian Rohr. Paper was published in the Briefings in Bioinformatics Oxford Journal. The paper proposes a methodology and related workflow based on publicly available tools to profile and analyze whole genome-scale biochemical models. The article provides guidance to modellers in computational methods to identify and apply suitable combination of tools for analysing dynamic behaviour of large scale metabolic models. In the workflow presented in the paper, the last step, trace analysis, introduces several data analytics techniques to analyze the behaviour of dynamic simulation traces. My contribution for the paper was identifying and applying suitable data analytics techniques and visualization methods to perform whole system data analytics and sub-system data analytics. This included writing R scripts for visualizing system behaviour, applying hierarchical clustering over reaction rate traces, investigating the evolution of properties over time, clustering the subsystems by their average behaviour, clustering the subsystems according to the degree of their structural inter-connectivity, pairwise comparing the clusterings by behaviour and structural inter-connectivity, clustering by subsystem average behaviour, clustering according to subsystem structural inter-connectivity, pairwise comparison of the clusterings by behaviour (both min-growth and enhanced-growth model) and structural inter-connectivity. The thesis doesn't contain any excerpts from the paper. However, clustering and custom distance measurement techniques used in the paper was useful when analysing the system behaviour of genome metabolic models of bacteria and DNA walker circuits (example application area discussed in chapter 6 and 7).

1.6 Thesis Outline

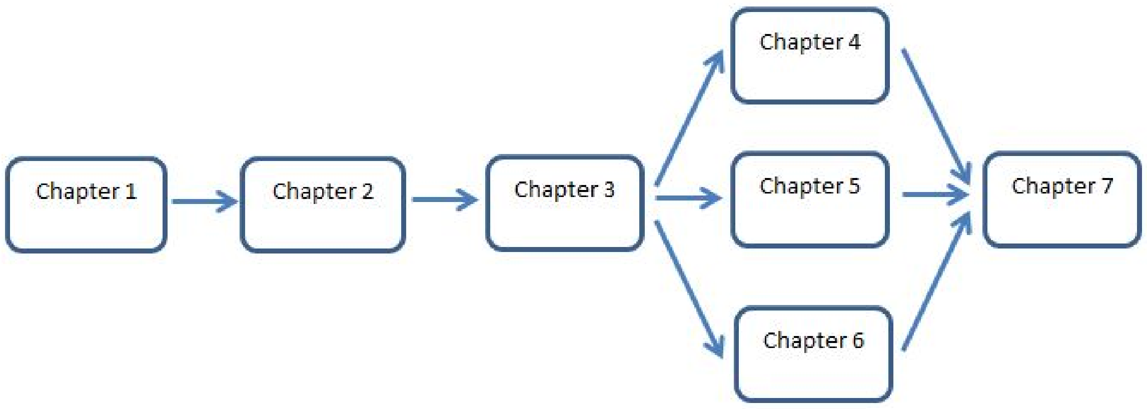


Figure 1.3: Thesis outline - The thesis can be read in the above order.

Chapter 2 provides an introduction to the main concepts discussed in the thesis. Existing data analytics driven methodologies are discussed too. Data driven methodology to identify alternative solutions is introduced in Chapter 3. Chapter 4, 5 and 6 discusses about the application of the methodology to identify alternative solutions. In Chapter 4 the methodology is applied to a mathematical problem where an optimisation approach is used to obtain alternative mathematical expression for a given target. Application of the methodology to GEM models is discussed in Chapter 5. Chapter 6 describes the application of the methodology to identify alternative solutions to DNA walker circuits. This chapter also includes the implementation of the Simulated annealing optimisation program to heuristically search the optimal design, method to compare two layouts by structure and application of optimisation to improve computation design of walker circuits. Finally conclusions, open problems and potential directions for future work are presented in Chapter 7.

Chapter 2

Applications and current trends in design, synthetic biology and computational optimisation

Introduction

This chapter provides a brief introduction to key concepts focused in my research. First an overview of the design process is provided. The research is focused in the area of biology. Hence next sections focus on providing an introduction to systems and synthetic biology and how computational design and modelling is used to improve design in biological systems. Next the use of data analytics and computational optimisations in modelling biological systems to improve the design of systems will be presented. Finally the effective use of alternative solutions in the design process will be discussed.

2.1 Design

2.1.1 Design: A definition

In English, the word design is both a noun and a verb. The word 'design' was introduced to the English language in 1540s. The word design is derived from the Latin word *designare* meaning 'to designate' and the French word *desseign* which gives a similar meaning (Harper 2015).

As a noun it means *a plan or drawing produced to show the look and function or workings of a building, garment, or other object before it is made.* As a verb it means *decide upon the* *look and functioning of (a building, garment, or other object), by making a detailed drawing of it*(Oxford Dictionary 2004).

Concept of design is used in many fields such as art, engineering, software development, fashion, architecture, processes design, product design and etc in different ways. Design is planning to manufacture an object, system, component or structure. Hence design can be formally defined as:

"a specification of an object, manifested by an agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints" (Ralph and Wand 2009).

2.1.2 Design as a process

Design is an essential component of sustainable development. Safety of a building depends on the strength of the structure it is built on. Similarly a good design ensures the object you are building can perform it's intended tasks, has a proper mechanism of handling errors and exceptions and can accommodate functionalities which could come up in the future. Apart from deciding the object's functionality a design could used to determine how existing resources can be optimally utilised to create the object. Hence design is a vital part of any development.

Design is being used in all most all disciplines and the process of design significantly varies depending on the context it is being used. Hence it is difficult to define a standard process for design. However Dorst and Dijkhuis discusses about two basic and fundamentally different ways of designing (Dorst and Dijkhuis 1995).

1. Design as a rational problem solving process

   Design is done by staying within the logical-positivism framework of science. Each aspect of the design process is either analytic or conclusively verifiable or at least confirm-able by observation and experiment. There is much stress on the rigour of the analysis of design processes, objective observation and direct generalizability of the findings. Logical analysis and contemplation of design are the main ways of producing knowledge about the design process. The Rational Model is based on a rationalist philosophy (design is informed by research and knowledge in a predictable and controlled manner) and underlies the waterfall model systems development life cycle. (Brooks Jr 2010)

   The Rational Model states:

- Designers attempt to optimize a design candidate for known constraints and objectives
- The design process is plan-driven
- The design process is understood in terms of a discrete sequence of stages

Typical states of The Rational Model includes

- Pre-production design: Main task of this phase is to conceptualize and document the design solution. This includes stating the design goals, researching existing similar design solutions and analysing the design goals in order to finalise the design solution and compile a defining a design requirement specification
- Design during production: During this phase a prototype of the design defined in the previous stage will be implemented and the tested. This helps continuous improvement of the designed solution
- Post production design feedback for future designs: The tested prototype solution will be implemented in the actual production environment. An evaluation of the solution will be done in order to identify the areas of improvement and future growth
- Redesign: any or all stages in the design process repeated with corrections made at any time before, during, or after production

This form of design is mainly followed in engineering (mechanical, electrical), software development, physics and architecture.

2. Design as a process of reflection-in-action

   Designers which follow this design paradigm believe that any design problem is unique and a core skill of designers' lies in determining how every single problem should be tackled. Such fundamentally unique problems are tackled with the aid of constructionist view of human perception- and thought processes. A designer sees design as a 'reflective conversation with the situation'. Problems are actively set or 'framed' by designers, who take action (make 'moves') improving the (perceived) current situation. This has always been left to the professional knowledge of experienced designers, and has not been considered describable or generalizable in any meaningful way. Thus people who follow this design paradigm find it unacceptable that these problems cannot be described in the prevalent analytical framework, and that their solving therefore cannot really be taught in the professional schools.

The Action-Centric model is an example for design as a process of reflection-in-action. The model states:

- Designers use creativity and emotion to generate design candidates
- The design process is improvised (without preparation)
- No universal sequence of stages is followed – analysis, design and implementation are contemporary and inextricably linked

This form of design is mainly followed in arts and fashion.

2.1.3 Design disciplines

How design is used in engineering, computer science and biology will be discussed below.

- Design in engineering

  Design in engineering covers a range of disciplines such as process design, vehicle design, military design, building and structural construction design, space-craft design, and appliance design. Design is a component of the engineering process. Design is carried out as a rational problem solving process. By applying scientific and mathematical principles to practical ends such as the design of efficient and economical structures, machines, processes, and systems. Hence the design process is more similar to The Rational Model. For an in-depth review refer (Lewis, Chen, and Schmidt 2006; Paynter 1961).

- Design in software engineering

  Design is used in several aspects in software engineering namely software process design, software design patterns, design paradigm in programming and user interface design.

  When it comes to designing software processes (building software programs) there is a separate category of design paradigms used namely waterfall model, spiral model, evolutionary model and Agile model. There is another variation of design called software design patterns. These patterns define how to develop reusable solution to a commonly occurring problem within a given context. Singleton, Abstract factor, Factory and Bridge are some examples for design patterns. Another variation of design used in software engineering is the logical design in programming. Object-oriented programming, logic programming and structured programming are few examples of different programming paradigms. These design models explains how a program is constructed and executed. Designing user interfaces is a separate paradigm which involves programming, arts and user psychology (Pressman 2005).

- Design in biology Adrian Mackenzie in his paper "Design in synthetic biology" characterises the design process by analysing the application of design techniques, such as bioinformatics software, presentation of information (i.e. diagrams), in modelling biological structures such as metabolic pathways, minimal genomes and biological standard parts (Mackenzie 2010). According to Mackenzie design processes in synthetic biology is based on two main notions; meta-technique and meta-material.

  It is observed that, in biology, design principles are used as a means of eliminating uncertainties and complexities. Meta-technique refers to the process of encapsulating these design techniques and formalizing them through practices of collaboration and standardisation. Meta-material refers to the methods used to represent biological behaviour in a designable format such as models, layered-processes, pathways and networks. Meta-materials and meta-techniques in combination represents dynamism of living things in the design processes. Even though, in real world, biological systems are a collection of complex and interlinked subsystems that work in synchronization, the computational modelling process is de-coupled. Computational modelling (designing) of biological systems are done in mainly three ways; development of complex individual and interlinked subsystems, changing the interactions between the interlinked subsystems, and implementation of the whole biological system. Hence Mackenzie identifies three main construction approaches in synthetic biology which are related to design principles such as abstraction, decoupling and modularity used in engineering. They are;

  - Device-based standardised construction
  - Mid-range problem-focused re-engineering of microbes as biotechnologies
  - Whole genome engineering

  Design in biology plays a significant role in how the public view novel technology such as biotechnology. A detailed explanation on design in synthetic biology can be found in (Mackenzie 2010).

2.2 Data Analytics in Synthetic Biology Design

2.2.1 Introduction to Data Analytics

Data analytics refers to qualitative and quantitative techniques and processes used to enhance productivity and business gain. Data is extracted and categorized to identify and analyse behavioural data and patterns, and techniques vary according to organizational requirements. Bioinformatics is the field in which data analytics techniques are incorporated in the field of biology. Bioinformatics is conceptualising biology in terms of molecules (in the sense of physical chemistry) and applying "informatics techniques" (derived from disciplines such as applied maths, computer science and statistics) to understand and organise the information associated with these molecules, on a large scale (Hucka et al. 2003).

In my thesis I have proposed a methodological approach to identify alternative solutions in a population based optimisation approach. The methodology is presented in Chapter 3. In instances where computational optimisation is used to improve design of complex system I have proposed a method to store the history of the optimisation process and extract alternative solutions. In order to extract solutions from the history different analytical techniques (mainly clustering and statistical analysis) was applied. These techniques were applied in three different example applications to understand the relationship between structure and behaviour of computational models. Applications include identification of alternative solutions through analysing structure and output of mathematical expressions (see Chapter 4), behaviour of metabolic models of bacteria and their structure (gene and reaction composition of models) (see Chapter 5), and, DNA walker circuit layout and its behaviour (see Chapter 6).

2.2.2 Prior Work: Analytics in Synthetic Biology

The research proposed in this plan builds on mainly two research efforts conducted previously; the master's dissertation and contributions to research paper, Reaction profiling for behavioural analysis of genome-scale biochemical models.

“Data Analysis Techniques for Metabolic Models of Bacterial Strains to Support Computational Design for Synthetic Biology” was my master's dissertation which I experimented on a set of data analysis techniques which can be applied to generate various perspectives of the E. coli genomic data. 55 publically available computational models of E.coli and Shigella strains were used for the analysis. The 55 models were broadly classified into two classes as E.coli and Shigella or into four classes as Shigella, Commensal Strains, Intestinal Pathogens and Extra-Intestinal Pathogens. A combination of supervised and unsupervised learning approaches was used to check whether these computational models could be grouped into their respective classes through the analysis of static properties and behaviours of the models. Three clustering algorithms (Hierarchical (Guess and Wilson 2002), K-medoids (Park, Lee, and Jun 2006) and DBSCAN (Ester et al. 1996)) with Euclidean distance as the dissimilarity measure were applied on static data. Hierarchical and DBSCAN clustering algorithms were applied to behaviour data with distance measures Euclidean, Discrete Wavelet Transform (DWT) and Dynamic Time Warping (DTW) (Liao 2005). The codes used for the analysis in the mastered thesis were generalized and compiled into a set of libraries. Some of these libraries were reused for the analysis in the paper making the analysis fast. Paper "Reaction profiling for behavioural analysis of genome-scale biochemical models" discusses about a set of methods to profile and analyse genome-scale biochemical models, based on the dynamic behaviour over time of reactions as well as of metabolites. We demonstrate our methodology by applying it to a reduced model of the whole genome metabolism of E.coli K-12. I performed clustering, and data analysis, over time series of reaction rates.

2.3 Optimisation

In a broad sense optimisation is the process of selecting the best possible solution for a problem from a set of all possible solutions.

Optimisation has become an important tool used in decision making on system design, analysis and operation. It is widely used in engineering, in electronic design automation, automatic control systems, and optimal design problems arising in civil, medical, chemical, mechanical, and aerospace engineering. Optimisation is also used for problems arising in network design and operation (shortest route, pipeline networks), finance (portfolio management), supply chain management, scheduling, and many other areas. The list of applications is still steadily expanding.

For most of these applications, optimisation is used as an aid to a human decision maker who supervises the process, checks the results, and modifies the solution based on actions suggested by the optimisation problem. For instance buying or selling assets to achieve the optimal portfolio. Additionally, with the improvement of computational power there is a rapid growth in embedded optimisation where optimisation is used to automatically make real-time choices, and carry out the associated actions, with no (or little) human intervention or oversight. For example Search

Engine Optimisation, News feed optimisation in Facebook. (references)

2.3.1 Why Optimisation?

As discussed in Chapter 1, the research question which is intended to answer through this research is *how to generate alternative designs for engineering bacteria, with suitable combinations of genes responsible for the optimal production of a specific bio product?*

Bacteria are cells. Cells have many parts, each with a different function. Some of these parts, called organelles, are specialized structures that perform certain tasks within the cell. Different species of bacteria have different structures and their capabilities varies according to these structures. Building block of a cell is proteins. Genetic information (i.e. DNA) in a bacteria has the instruction manual describing the way these proteins should be created so the bacteria can get the embedded functionalities. In other words genetic information of bacteria decides the structure and capabilities of a bacteria. Hence the design of a bacteria refers to the DNA sequence of the bacteria.

Bacteria has several thousand genes. As mentioned above the total genome of a *E. coli* consists of 4000 genes. The most suitable combination of genes responsible for optimal production of bio products can be found after testing all the possible combinations of 4000 genes. This large solution space (4000 factorial of possible combinations) is impossible to explore with regards to time and computational resources and it would be a waste of resources to explore each and every possible combination.

Therefore optimisation provides a mechanism to automatically traversing through the large solution space in an efficient manner (time and resource). With proper optimisation methods we could also get the alternative design solutions which could be useful apart from the optimal solution and enhance the search by specifying constraints which designs should abide to. Additionally reliable solutions can be found without needing a lot of pre-experience information relating to the problem.

2.3.2 Definition

Optimisation is the process of selecting the best possible solution for a problem utilizing the available resources while ensuring the constraints imposed in achieving the solution are not violated (Qing 2006).

An optimisation problem can be mathematically defines as follows.

Find $\quad \mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_N]$ where

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_i(x) \leq 0, \quad i = 1, ..., m \\ &&& h_i(x) = 0, \quad i = 1, ..., p \end{aligned}$$

x is the optimisation parameter

$f(x)$: is the objective function to be minimized over the variable x

$g_i(x) \leq 0$ are m inequality constraints

$h_i(x) = 0$ are p equality constraints

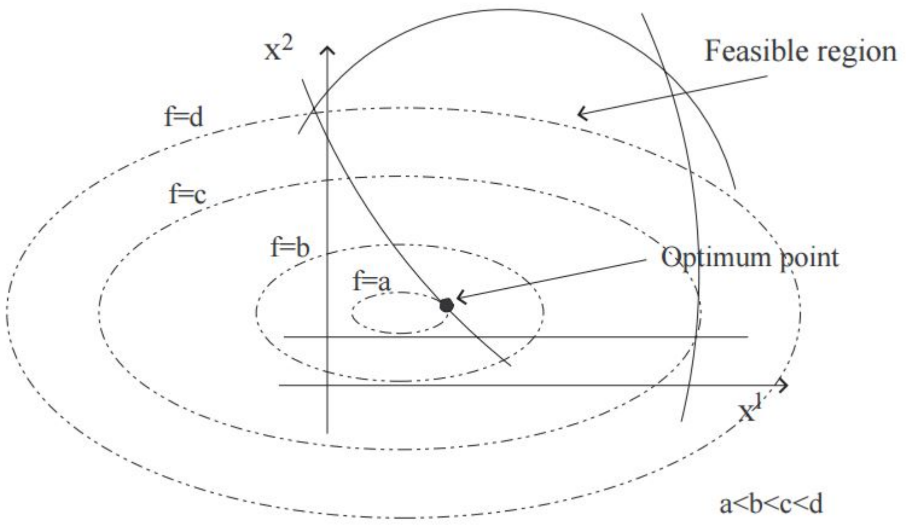


Figure 2.1: Feasible Search Space of Optimisation with multiple objective functions

For instance in portfolio optimisation, an investor seeks the best way to invest some capital in a set of n assets. The variable x represents the investment in an asset. The objective or cost function might be a measure of the overall risk or variance of the portfolio return. In this instance, the optimisation problem corresponds to choosing a portfolio allocation that minimizes risk, among all possible allocations that meet the firm requirements. The constraints might represent a limit on the budget (i.e., a limit on the total amount to be invested), the requirement that investments are non-negative (assuming short positions are not allowed), and a minimum acceptable value of expected return for the whole portfolio.

2.3.3 Properties of an Optimisation Problem

As explained above an optimisation problem is made up of three essential properties: optimisation parameters x; objective functions, $f(x)$ and constraint functions $g_i(x)$ and $h_i(x)$.

- Optimisation Parameters

  Optimisation parameters (x) are the quantities that can be treated as variables in an optimisation problem. For example in the investment fund management problem, the optimisation parameters are the amounts of money invested in each fund. Optimisation parameters are also referred to as decision variables. An optimisation parameter can be continuous, discrete, or even symbolic.

- Objective Function

  In general there are several acceptable solutions for a given problem. However the purpose of optimisation is to select the best possible solution. Therefore, a criterion has to be specified for comparing different solutions. This criterion, when expressed as a function of the optimisation parameters (design variables), is called as the objective function. For instance, in the investment fund management problem, the best plan is the one which gives a maximum return.

  An optimisation problem can have more than one objective function. For example, in the investment fund management problem, the best plan can also be selected based on which has the maximum return and minimum risk. Even though almost all optimisation problems have objective functions, there are instances where an objective function is not required. For instance, when designing integrated circuit layouts the goal is to find optimisation parameters (design variables) that satisfy the constraints of the model. In such a scenario a user does not particularly want to optimise anything, hence there is no necessity to define an objective function.

- Constraint Function

  Constraints specify the restriction on the values an optimisation parameter (design variable) can have. For example, the investment fund management problem, the amount of money invested should not exceed the available money. Hence design constraints represent the limitation on the performance or behaviour of the system.

  Same as objective functions, when defining an optimisation problem constrains are not absolutely necessary. In certain scenarios constrain functions and objective functions are interchangeable.

2.3.4 Types of Minima

Minima is used to describe the solution for an optimisation problem. Figure 2.2 is an illustration of the different types of minima.

- Global Minimum

  A point x^* is a global minimum of the function $f(x)$ if we have $f(x^*) < f(x)$ for any x such that $x^* \neq x$. This point is marked in 2.2 as global minimum.

- Strong Local Minimum

  A point x^* is a strong local minimum of the function $f(x)$ if we have $f(x^*) < f(x)$ for any $x \in V(x^*)$ and $x^* \neq x$, where $V(x^*)$ defines a neighbourhood of x^*. Three points which fall under this definition is marked in 2.2.

- Weak Local Minimum

  A point x^* is a weak local minimum of the function $f(x)$ if we have $f(x^*) \leq f(x)$ for any $x \in V(x^*)$ and $x^* \neq x$, where $V(x^*)$ defines a neighbourhood of x^*.

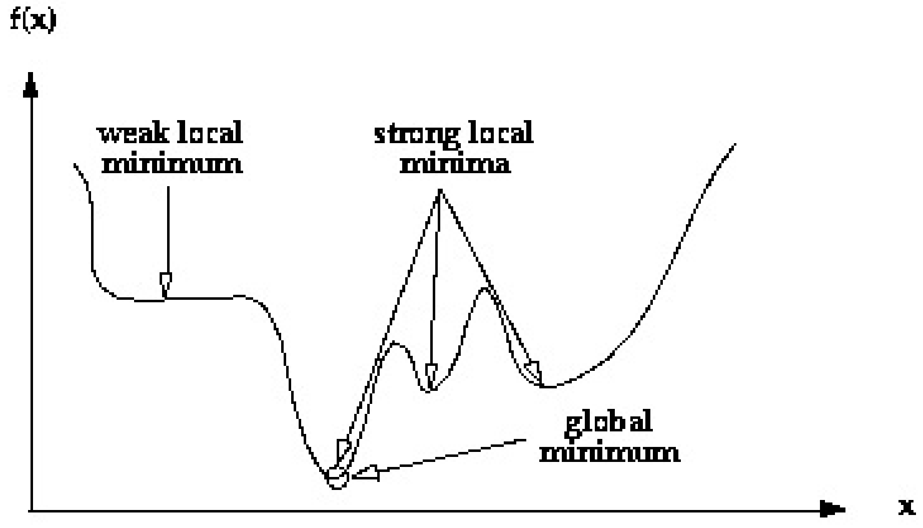


Figure 2.2: Types of minima in optimisation [source: https://www.phy.ornl.gov/csep/mo/node5.html]

Usually in an optimisation problem, we seek for the global minimum (or maximum) of $F(x)$. An optimisation problem may have two or more local minima. Since optimisation algorithms in general are iterative procedures which start with an initial estimate of the solution and converge to a single solution, one or more local minima ma be missed. If the global minimum is missed, a suboptimal solution will be achieved. Therefore as a best practice optimisation is performed several times with different initial estimates. This is a summarised explanation on minima; see (Antoniou and Lu 2007) for further details.

2.4 Classification of optimisation problems applied in optimal design

In reality the problems we encounter are different in nature. The physical properties involved and how they behave are different from one instance to another. For example managing an investment fund is different to deciding the optimal trajectory for space mission. Hence it is difficult to have one optimisation algorithm to model all the problems. There are numerous optimisation algorithms proposed to cater different types of problems.

This introduces the need for a suitable categorisation of the algorithms used in the context of optimal design in order to provide a guideline for selecting the appropriate algorithm for your need. Literature on optimisation gives no definitive answer as to what is the categorisation of optimisation algorithms. Like most of the concepts in science it is subjected to individual perspective. Several structures have been imposed based on the properties of an optimisation problem. Below are some of the major categorisations of the algorithms.

Optimisation algorithms are design and inspired by real world problem. Hence the word algorithm and problem will be used interchangeably during explanations.

- **Unconstrained Optimisation and Constrained Optimisation**

  Important distinction between optimisation problems are based on the types of values allowed for decision variables. Unconstrained optimisation problems have No constraints on the values which variables can accept. Derivative free optimisation, non-linear equations, non-linear least-squares problem are few examples for unconstrained optimisation.

  Where as in contrast in constrained optimisation problems there are constraints on the values variables can accept. The constraints on the variables can vary widely from simple bounds to systems of equalities and inequalities that model complex relationships among the variables. Bound constrained optimisation, linear programming, quadratic programming and non-linear programming are few examples for constrained optimisation.

  All three example applications discussed in this thesis uses constrained on the decision variables. In the first example, using gene expression programming optimisation to generate the optimal mathematical expression, there are constraints on the number and type of operators that are allowed, the range of operands allowed and the length of the expression (see Chapter 4). In the second example, a genetic algorithm is used to optimize the biomass production of a GEM model of bacteria, the number of genes and reactions associated with genes are constrained and only allowed to pick form a pre-defined list (see chapter 5). In the last example, where random restart hill climbing and simulated annealing algorithms are used to optimize the layout of DNA walker circuits, the number of nodes and where they can be placed on the Cartesian plane have constrains (see Chapter 6).

- **Black-box optimization**

  Most of the optimization methods require us to have an understanding about the underlying fitness function, and especially its rate of change. When an optimization problem can be formulated numerically (i.e., without the use of simulations), such optimization methods often present the most efficient choice. However, in certain scenarios (e.g. architectural design practice), an explicit formulation of the objective function and of its gradient often is unavailable. Architectural designers generate and evaluate design candidates employing simulations and other quantitative measures derived from a parametric model (Oxman 2006) without specifying a mathematical expression that relates the model parameters to the fitness criterion. Therefore, in black-box optimization, it is unknown how the fitness function is calculated. A set of known decision variables with known lower and upper bounds will be fed to the black-box. The black-box will then output the fitness value which will be used to evaluate the fitness of the candidate. Optimization problems involving numerical simulations are one of the primary applications of black-box methods (Wortmann and Nannicini 2016).

  From the work presented in this thesis, there is only one example that falls under bank-box optimization. The second example application area discussed in this thesis (see Chapter 5) uses a model-based optimisation search. In this example, a genetic algorithm is employed to optimise the structure of a genomic metabolic model of bacteria. The optimisation program generates a solutions and then the solution is passed on to a simulator in order to compute the fitness. The simulator uses ordinary differential equations to predict metabolic behaviour of bacteria. The fitness value is then used in the evaluation process in the genetic algorithm (see Chapter 5).

2.4.1 Optimisation problems considered in this thesis

My research was focused on defining a methodological approach to identifying alternative solutions by analysing the optimisation history of a optimisation program. The methodology was applied to three example application areas. When considering the characteristics of these example applications, they can be broadly categorized into two main classes of search problems; metaheuristics and model-based optimisation.

1. Metaheuristics

   Metaheuristic is a technique designed to find a sufficiently good solution (not the best/ globally optimal solution) to an optimization problem within a limited computational capacity (i.e. time) (Parejo et al. 2012). Metaheuristic search can be considered as a minimization or maximization problem. When searching over a large set of feasible solutions, metaheuristic search often finds good solutions more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution. Metaheuristics search sample a subset of solutions which is otherwise too large to be completely enumerated or otherwise explored. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. Two major components of a metaheuristic algorithm are: intensification (exploitation) and diversification (exploration). Diversification refers to generate diverse solutions in order to explore the search space on a global scale, while intensification refers to focus the search in a local region knowing that a current good solution is found in this region. A good balance between intensification and diversification will help in reaching global optimal (Yang 2011).

   Two out of the three example applications fall under this category. First application is use of gene expression optimisation algorithm to produce a mathematical expression that would evaluate to a specified target value while using the user specified operators and operand range (see Chapter 3). In this example an infinite set of feasible solutions are available and the metaheuristic search will find a good solutions more quickly. The second example is where a random restart hill climber and a simulated annealing algorithm were used to identify the optimal layout of the nodes of a DNA walker circuit that would minimize the information leakages and circuit area (see Chapter 6).

2. Model-based optimisation

In model-based optimization, a mathematical function is interpolated through known function values, and this interpolated function is used as a model of the unknown fitness landscape (Wortmann and Nannicini 2016). A proxy model thus provides instant estimates of the performance of design candidates. Model-based optimization strategies alternate between improving the surrogate model (exploration), and using the surrogate as a guide to find good design candidates (exploitation). Model-based methods typically require only a small number of function evaluations, and thus are especially appropriate for cases where such evaluations take a long time. The combination of finding good design candidates and estimating the performance of the whole design space is unique to model-based optimization methods and makes them seem especially appropriate for the architectural design process. The surrogate model allows playful interactions, while the discovered good design candidates provide starting points for further exploration (Bradner, Iorio, Davis, et al. 2014).

The second example application area discussed in this thesis (see Chapter 5) uses a model-based optimisation search. In this example, a genetic algorithm is employed to optimise the structure of a genomic metabolic model of bacteria. The optimisation program generates a solutions and then the solution is passed on to a simulator in order to compute the fitness. The simulator uses ordinary differential equations to predict metabolic behaviour of bacteria. The fitness value is then used in the evaluation process in the genetic algorithm.

2.5 Existing algorithms and approaches

In this section I intend to discuss about three main existing approaches related to the methodological approach I have proposed in identifying alternative solutions from the history of a computational optimisation program. Also I would compare the existing approaches with my approach highlighting the similarities and differences in application of the concepts.

2.5.1 Automatically Defined Function (ADF) in genetic algorithms

Genetic algorithm (GA) is a metaheuristic. A GA is always applied to a population of individual objects. Each individual object in the population is associated with a fitness value. The GA transforms the population into a new generation of the population using genetic operations such as crossover (sexual recombination) and mutation. These genetic operation are analogous with the Darwinian principle of reproduction and survival of the fittest. Each individual in the population represents a possible solution to a given problem. The genetic algorithm attempts to find a very good (or the best) solution to the problem by genetically breeding the population of individuals over a series of generations (Koza 1995).

Genetic programming (GP) is a technique of evolving programs. GP is an extension of the conventional genetic algorithm in which each individual in the population is a computer program. The search space in genetic programming consists of all possible computer programs. Each computer program is composed of a set of functions and terminals (the inputs to the undiscovered computer program) appropriate to the problem domain. The functions may be standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions, or domain-specific functions. A mathematical expression or parse tree can be considered as a computational program (Koza 1995). An automatically defined function (ADF) is a function that gets dynamically evolved during a run of generic programming. This function can be either subroutine, subprogram, DEFUN, procedure, or module. According to (Koza et al. 1996) the functionality of the ADFs are as follows: "When automatically defined functions are being used, a program in the population consists of a hierarchy of one (or more) reusable function-defining branches (i.e., automatically defined functions) along with a main result-producing branch. Typically, the automatically defined functions possess one or more dummy arguments (formal parameters) and are reused with different instantiations of these dummy arguments. During a run, genetic programming creates different subprograms in the function-defining branches of the overall program, different main programs in the result producing branch, different instantiations of the dummy arguments of the automatically defined functions (function defining branches), and different hierarchical references between the branches."

The first example application in my thesis is about using optimisation to generate mathematical expressions. I used an existing optimisation approach named gene expression programming which has adopted ADFs in creating the mathematical expressions (Ferreira 2006). Further, The paper "Use of Automatically Defined Functions and Architecture-Altering Operations in Automated Circuit Synthesis with Genetic Programming" (Koza et al. 1996) discusses how ADFs are used in determining the layout of and IC. The third example application where I used an simulated annealing algorithm to generate the circuit layout for a DNA walker circuit, can be improved by adopting ADFs.

2.5.2 Innovization in multi-objective optimization

Multi-objective optimization (MO) is mathematical optimization problem that involves optimizing more than one objective function at the same time (Huang, Zhang, and Li 2019). Therefore, optimality of solutions will be decided based on trade-offs between two or more conflicting objectives. Multi-objective optimization theoretically does not have one solution but produces a set of Pareto-optimal solutions.

A solution is called Pareto optimal if no change could lead to improved fitness for some solutions without fitness some other solutions being reduced, or if there is no scope for further Pareto improvement (a change that result in a new fitness where some solutions will have an improved fitness, and the fitness of other solutions will remain same).

Each of these pareto-optimal solutions which are produced by MO, is optimal subjective to a trade-off among the objectives. Since the outcome of the optimisation are multiple solutions, multi-objective optimization is ideal for finding a set of alternate solutions. This gives the opportunity to either for finally choosing a single preferred solution or to launch a future analysis. Evolutionary algorithms (EAs) are ideal for solving multi-objective optimization problems because it is population based optimisation approach (Deb, Bandaru, and Celal Tutum 2012).

Since Pareto-optimal solutions are all optimal, they are likely to possess some common properties (design principles) related to design variables, objectives and constraints. These common properties could be identified as 'signatures' to Pareto-optimal solutions (Deb, Bandaru, and Celal Tutum 2012). The process of extracting these common properties from a set of Pareto-optimal solutions in the form of mathematical relationships between the variables and objective functions is known as Innovization. Temporal innovization refers to the study of evolution of design principles over generations of an multi-objective evolutionary approach (Bandaru and Deb 2015).

Innovization is not a straight forward simple procedure. As proposed by (Deb and Srinivasan 2006) innovization procedure includes a) obtaining a set of Pareto-optimal solutions from a multi-objective optimization tool; b) clustering to identify well-distributed solutions; c) modify the solutions using a local search procedure. Next two independent procedures are used to verify these filtered solutions (extreme and intermediate Pareto-optimal solutions). First, a single-objective optimization procedure (e.g genetic algorithm) is applied on the extreme Pareto-optimal solutions to verify each objective function subject to satisfying given constraints. Second, intermediate Pareto-optimal solutions are verified by using the normal constraint method

(Messac and Mattson 2004). It is similar to obtaining a best representative of each grouping of Pareto-optimal solutions. Finally, a data-mining strategy must be used to automatically evolve design principles from the combined data of optimized design variables and corresponding objective values (Deb and Srinivasan 2006). An approach has been proposed by (Bandaru and Deb 2011) to automatically discover common design principles in Pareto-optimal solutions using machine learning procedures such as clustering.

I have proposed a methodological approach to identify alternative solutions from the history of a computational optimisation program in chapter 3 of the thesis. The methodology includes a "Temporal analysis" component which focuses on analysing the optimisation archive (a data storage composed of all visited solutions by an optimisation tool within a single or multiple run). Temporal analysis will look at how a solution has evolved from the initial solution to optimal solution in the current run and identify sub design patterns that remains constant overtime. In comparison with innovization (or temporal innovation) mentioned above, temporal analysis naively looks at the evolution of a solution within a single run. However, there are few differences between the two approaches. Firstly, while Innovization focuses on defining design principles (identifying mathematical relationships between decision variables and objective functions), temporal analysis component in my methodology aims at identifying constant sub patterns in the overall model with the aim of improving the optimisation process (reducing the time to reach an optimal solution by reducing unnecessary variability). Secondly, innovization is applied to a set of Pareto-optimal solutions generated by a multi-objective optimization approach, while temporal analysis focuses on the evolution of a single solution generated by any optimisation approach (multi-objective or single objective optimisation). Thirdly, innovization depends on a particular type of optimisation tool, whereas temporal analysis can be applied to the output of any optimisation tool. Innovization is a well-defined complex procedure involving the use of several computational and data mining techniques. At present, temporal analysis uses only visual analytics to identify the constant sub-components within a solution. However, temporal analysis could certainly be improved by adopting certain concepts used in innovization especially by adopting data mining techniques to automatically identify the constant sub components.

2.5.3 Multiple Distinct Solutions in evolutionary algorithms

Traditionally an optimisation algorithm produces only one solution at the end of a single run. This is ideally the best solution found during the search process. However, we can modify optimisation algorithms to output more than one solution that satisfy the fitness requirements within a single run. These solutions are referred to as multiple distinct solutions.

Multiple distinct solutions are important for couple of reasons. First, the availability of many solutions provide the user with a choice. For example, in a timetabling problem, having an issue with one timetable will make the timetable unusable. In this case having a second solution is helpful as we do not have to run the algorithm again to generate a new timetable. Another example is, in a Travelling salesman problem, there could be a scenario where one road is blocked and you need to find an alternative route to get to the destination. Therefore, having more than one path is beneficial. Secondly, when we are unable to obtain the optimal solution or when the implementation of the optimal solution is less cost effective in real life, it would be ideal to have multiple distinct solutions (Turner 1994).

An advantage of multiple distinct solution is that it improves solution quality, with in some cases, little or no extra computational cost. A drawback of multiple distinct solutions is that if the multiple solutions are similar, and if one is rendered unsuitable for the task, the rest will be unsuitable and redundant as well. Therefore, having diverse multiple distinct solutions is useful (Turner 1994).

Multiple distinct solutions are mainly coupled with genetic algorithms (GA). Multiple distinct solutions work well with GA rather than simulated annealing (SA) or stochastic hill climbing (SH) algorithms because each generation in a GA has a population of solutions. Where as in SA or HC it looks at only one solution within an iteration. In order to incorporate multiple distinct solutions into a GA specific algorithm should be added to a normal GA. Spatial selection, islands, crowding, sharing and tribes are some of the methods adopted by GAs to produce multiple distinct solutions (Turner et al. 1996).

Multiple distinct solutions with GA have been widely applied to solve the timetabling problem. “Comparing Genetic Algorithms, Simulated Annealing, and Stochastic Hill climbing on Timetabling Problems” (Ross and Corne 1995), “Obtaining Multiple Distinct Solutions with Genetic Algorithm Niching Methods” (Turner et al. 1996). and “Genetic algorithms and multiple distinct solutions” (Turner 1994) are three studies which compares the effectiveness of multiple distinct solutions with GA against SA and HC optimisation algorithms. According to the finding of the paper, when comparing solution quality SA is better than the GA. However, the GA performed better compared to the multiple distinct approach. As presented by (Ross and Corne 1995) “number-of-distinct-solutions advantage offered by the GA is evidently most useful in those cases where its solution quality performance is acceptable in comparison to SA and SH”. Also, multiple distinct solution approach is not ideal when there are many solutions and it's easy to obtain them.

The concept of alternative solutions I have proposed in my work is similar to multiple distinct solutions. The purpose of alternative solutions is to provide biologists (since the example applications I have picked are from synthetic biology) with a choice when the optimal solution is not feasible to implement. However, currently I do not propose a mechanism to measure the similarity between solutions in terms of viability. Since these are computational models, they need to be tested in in-vitro to understand the behaviour and significance of similarity values and boundaries. However, there are several advantages in my approach as well. Unlike GAs which need to have additional algorithms embedded to facilitate multiple distinct solutions, my methodological approach can work with the basic optimisation algorithm. The history of the optimisation program will be stored and alternative solutions will be extracted from the history. Further, the proposed methodology can be applied to any form of optimisation approach GA, SA and SH as long as the history is stored. Also, multiple distinct solutions enabled GA needs to finish in order to identify the distinct solutions, however, in my approach it is possible to access the optimisation archive and look for alternative solutions while the optimisation process is still running.

2.6 Current Approaches to Obtain Alternative Solutions

2.6.1 DNA Circuits and DA

There is a study done by a group of researchers at University of Tokyo on automating the design of DNA circuits using simulated annealing (Kawamata, Tanaka, and Hagiya 2009). Even though the approach used for automation of design of circuits (using simulated annealing) is similar there are several differences in the published work and the work presented in this thesis. The focus of the published work is on automating the design of DNA logic gates using simulated annealing. The heuristic search is designed to optimise the design by optimising its chemical kinetic parameters (i.e. reactions of DNA molecules and the time change of the concentration of structures). Hence the implementation of the simulated annealing algorithm is different with respect to the fitness measure, termination condition and acceptance probability. In contrast the work presented in this thesis focuses on automating the design of DNA walker circuits. DNA walker circuits and DNA logic gates are different in nature. In a nut shell a DNA walker circuit can be considered as a collection of several DNA logic gates. Further, optimisation process is used as a mechanism not only to automate the design of circuits but also to identify alternative designs with similar behaviour yet having structural differences. Hence the simulated annealing process is slightly different than the one in the published work. Fitness measure is defined based on structure of the circuit which uses physical parameters such as area and leaks. The published method is a maximization optimisation while the proposed optimisation algorithm is a minimization problem.

2.6.2 Memory-based schemes for evolutionary algorithms

A memory in an optimisation context can be defined as a storage of previously evaluated solutions. In an optimisation environment memory schemes are used to store useful information from the current environment and reuse it later in new environments. Several research approaches have shown that memory schemes are useful in real-world optimization problems. Real-world optimisation problems usually involve several conflicting objectives. Conflicting objectives occur when an improvement in one objective function mirrors the worsening of another one. The main aim in multi-objective optimisation is to select the best trade-offs among these conflicting objectives. Real-world optimisation problems are usually categorized under dynamic optimisation problems (DOPs). Speciality about DOPs are that the fitness function, design variables and/or environment conditions change over time. Hence the goal of a dynamic optimisation problem is not to locate the optimal solution but to tract the moving optima with time. For any optimization algorithm, proper balance between exploration and exploitation of the search space is necessary to achieve a global optimal solution. Exploration (i.e. diversification) involves global search in the search space and exploitation (i.e. intensification) involves search in a local region depending upon the current best solution. Too much of exploration and exploitation harmfully affects the performance of the algorithm by increasing the convergence time and increasing the chances to fall into local optima. Memory-based schemes are employed in DOPs to address two main problems: premature convergence and limitation on number of fitness evaluations permitted. Premature convergence occurs when a population for an optimization problem converged too early, resulting the end solution being suboptimal. This method is employed in several evolutionary algorithms such as genetic algorithms (GA) [2, 5, 8, 4] and particle swarm optimisation (PSO) [1, 3, 6, 7].

1. Estimate the fitness value with reference to the search history when the number of fitness evaluations are limited

2. Maintain solution diversity in generations by using the best solutions in the previous

generation to construct the initial population in the next generation This is achieved using two types of memory schemes: implicit and explicit memory schemes

2.7 Importance of computational modelling in design

Design is an essential component of sustainable development. A good design ensures the object (or the system) being built can perform it's intended tasks, has a proper mechanism of handling errors and exceptions and can accommodate requirements which could arise in the future. Thus, a designer is required to have thorough domain knowledge and the ability to foresee all the potential opportunities and threats to ensure a robust design. However, as the complexity of the system increases it becomes impossible for a human designer to gain sufficient insight into the system so as to provide better design and accurate predictions. Therefore, computational modelling has become a useful technique that facilitates the design of complex systems.

Computational modelling is the use of computers to simulate the behaviour of complex systems. Numerous variables that characterises a complex system are encoded into a computational model using mathematical concepts. During simulation each of these variables alone or in combination are modified to observe the changes in the outcome. This allows the researchers to make predictions on the behaviour of the real system that is being studied in response to the changes. One of the main advantages of this method is that it allows the scientists to experiment with endless permutations and combinations of input parameters that have been otherwise impossible to make out by a human experimenter. Thus, computational modelling can expedite research by facilitating scientists to conduct as many simulated experiments as required by computer saving time, money and material. Synthetic biology is one of the novel areas that is benefited by computational modelling.

The ultimate goal of any design process is to come up with the best design; as resources are limited, to minimize the cost and resource consumption, and to maximize the performance, profits and efficiency. To optimize means to find the best solution, the best compromise among several conflicting demands subject to predefined requirements. Mathematical optimization has been extremely successful as an aid to better decision making. Therefore, computational optimization, modelling and simulation forms an integrated part of the modern design practice.

However, the extensive level of complexity of real-world systems often makes it difficult to accurately characterise a system into a perfect computational model and the non-linearity in correlations makes it difficult to adequately capture them by optimization tools. Consequently, there is a gap between approximations delivered by optimization and their practical possibilities. In certain instances, the optimal design proposed by the optimisation tool typically involves implementation impracticalities in predicted parameters making the optimal design less robust (i.e. too many modifications, impossible alterations, highly time and resource consuming alterations). In such cases it is considerate to settle for robust suboptimal design options with sufficiently good accuracy and reasonable resource expenditure. Hence alternative solutions with optimal (similar behaviour and different structures) or suboptimal behaviour play a vital role in the design process.

Analysing the output from optimisation tools are essential in identifying these alternative solutions. For this purpose, data analytics techniques are applied. Generation history from optimisation and characteristics of individual solutions could be used to gain insights on the solution space explored by the optimisation tools. This amounts to a large volume of data generated over a short period of time with different data types (i.e. real data, nominal, ordinal, time series, graphs and etc). identifying alternative solutions may involve performing descriptive statistics, structural comparisons, groupings, pattern recognition, solution profiling and etc depending on the solution type. In order to analyse these large amounts of data efficiently and accurately, it is essential to incorporate data analytics techniques.

Summary

The chapter has provided a brief introduction to design process in general and presented an overview of design paradigms applied in engineering, software engineering and synthetic biology area. The two main forms of biological systems that were discussed are metabolic models of bacteria which simulate the metabolic activity of an E.coli bacteria, and DNA circuits which is biochemical circuit built using DNA strands. Computational optimisation is a widely used technique to optimise computational design. Optimisation plays a vital role in the design process by enabling the global solution space for a problem which otherwise impossible to do by hand. However, a common limitation with computationally designed solutions is that the feasibility in implementing them in silico could be difficult. These difficulties are often attributed to modifications which consume a lot of resources. Having alternative solutions (behaviourally similar but structurally different) solutions will aid biologists to make effective design decisions. Therefore a methodology will be proposed in the next chapter which aims at utilising the computational optimisation history which was used to generate the optimal solution, in order to extract alternative solutions.

Chapter 3

Methodological approach to identify alternative solutions from the history of a computational optimisation program

Introduction

Traditionally design was done based on experience and intuition of designers. However, with the advancement of technology, incorporating computational strategies in the conventional design process provided with the ability to explore complex systems. Computational design, which takes advantage of mass computing power, machine learning, and large amounts of data, is changing the fundamental role of humans in the design process. The methodology described in this chapter aims at improving the computational design process. The methodology is mainly applied in identifying alternative solutions in a population based data generation approach.

3.1 Elements of the approach

As described in Chapter 2 it is a widely established and effective practise to incorporate optimisation into the computational design process. Given a computational model optimisation is used as a technique to select the best (i.e. optimal) value combinations required for input parameters in order to generate the optimal outcome. However, in reality the best computational design obtained through optimisation approach might not be feasible to implement.

Usually in an optimisation approach, the optimal solution is the output. The intermediary solutions which are generated from the initial solution until the search reaches the optimal solution is discarded at the end. Each optimisation run contains a rich source of information that could be used to derive alternative conclusions. The speciality of the methodology proposed in this thesis is that it analyses these intermediary solutions in order to improve the overall design process. The methodology defined in this chapter aims at exploiting the optimisation history in order to enhance the computational design process. A data analytics driven search approach is utilized to analyse the optimisation history. The analysis focuses mainly on three aspects with the aim of improving computational design; extract alternative solutions, observe the evolving nature of solutions to predict recurring patterns and reinforce design constraints.

The methodology is applied to three examples. In all three examples, an optimisation algorithm is used to generate computational designs to satisfy a predefined target behaviour. The examples are: use of gene expression programming optimisation to generate mathematical expressions for a predefined target value, generation of computational designs for models of genome metabolic bacteria using genetic algorithm optimisation and finally, creation of computational designs for DNA walker circuits using a simulated annealing optimisation algorithm. These examples will be discussed in detail in Chapters 4,5, and 6 respectively.

3.1.1 Workflow of the methodology

In this section the work flow of the methodology is described. The methodology consists of three main phases:

1. Mathematical model definition:

   As the methodology focuses on improving computational design it is necessary to have a mathematical model that mimics the behaviour of the whole or part of the complex system should be created.

2. Target driven optimisation:

   Once the mathematical model is formed, computational optimisation approaches can be applied to automatically generate solutions which satisfy predefine target behaviour. An optimisation archive is created in this step. Output at each iteration of the optimisation process is stored in the optimisation archive. Data in the archive is used for further analysis in phase three.

3. Approach to analyse the optimisation archive:

Historical data from optimisation is stored in the optimisation archive. This data will be analysed in three different ways to gain three different categories of insights.

(a) Extract alternative solutions

(b) Temporal analysis

(c) Standardize design constraints

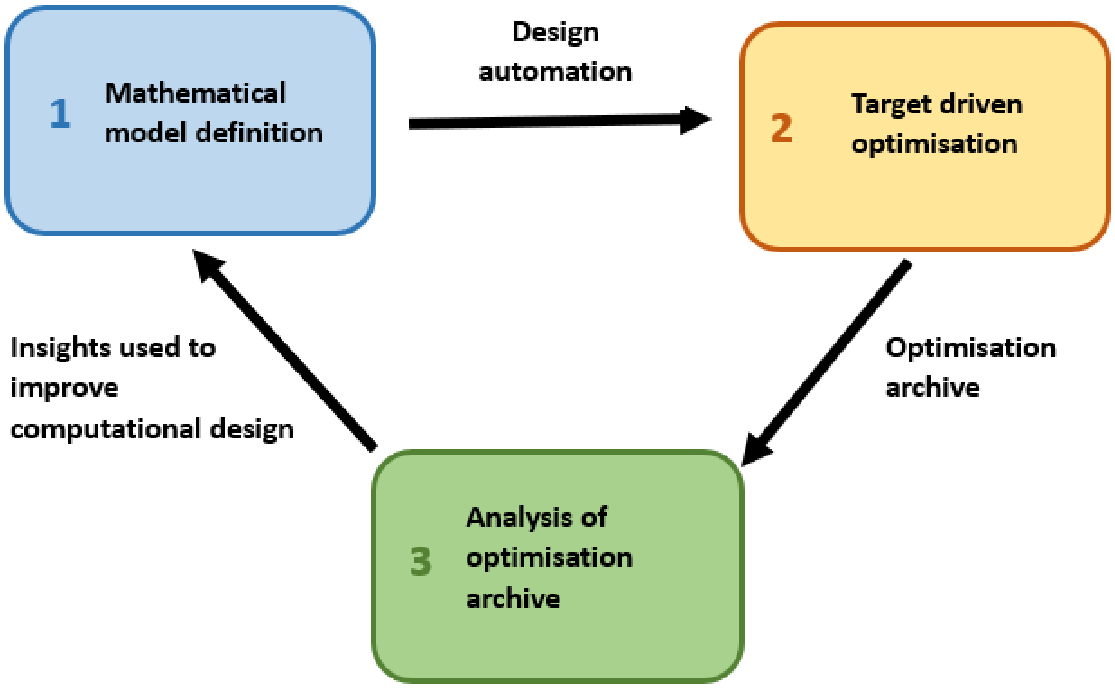

1 Mathematical model definition
Design automation
2 Target driven optimisation
Optimisation archive
3 Analysis of optimisation archive
Insights used to improve computational design

Figure 3.1: Methodology work flow overview

A representation of the overall work flow is shown in figure 3.1. Each phase of the work flow will be explained in detail in the following sections.

3.1.2 Mathematical model definition

A mathematical model is a description of a system with the aid of mathematical concepts. In order to create a mathematical model, firstly, we need to select the characteristics of the system we would be observing. Next, physical observations of the complex system should be recorded by measuring the change of these characteristics. Therefore, it is important that these characteristics should be quantifiable. Eventually, these characteristics will be translated to parameters of the mathematical model and the values for the parameters will be estimated using physical observations. Observations could be made on part of the system or on the whole system depending on its complexity. Also the characteristics and the number of observations made depend solely on the complexity of the physical system as well.

Observing a system can be defined as measuring the behaviour of the system in response to different stimuli. Therefore it is essential to identify quantifiable characteristics in the complex system. For example in a GEM model, which depicts the metabolic activities of E coli bacteria, concentration of chemical compounds is a quantifiable characteristic. E coli feeds on sugar and produce ethanol. In order to understand the metabolism with respect to sugar concentration, the change in sugar and ethanol concentration should be measured over time. Here the sugar and ethanol concentrations are quantifiable parameters.

Once the physical observations are made, mathematical concepts are applied to identify the relationships between the input and output parameters (input parameter is the property which decreases in quantity over time and the quantity of the output property increases over time). Once this is completed a formal mathematical description of the system is made and it is refereed to as the mathematical model. A mathematical model includes the input and output characteristics parameters, assumptions which it is based on (aspects which are kept constant) and how input and output properties are related in the complex system.

In this thesis three example applications of the methodology is discussed in the next three chapters. Mathematical expressions described in Chapter 4 is a whole system but not necessarily a complex system. GEM models which are discussed in Chapter 5 depicts only the metabolic activities of E coli bacteria which can be considered as part of the complex system. Finally DNA circuits represents a whole complex system. Figure 3.2 is an abstraction of the mathematical definition phase.

3.1.3 Target driven optimisation

In this phase computational optimisation is applied to the model defined in the first phase. Computational optimisation was used as a means of obtaining the best model that results in the target behaviour which is predefined before running optimisation. Optimisation was set up to select the best model by testing models with different combinations of values for parameters of interest.

1. Defining design requirements:

   Once the mathematical model is completed, define the design requirements expected from

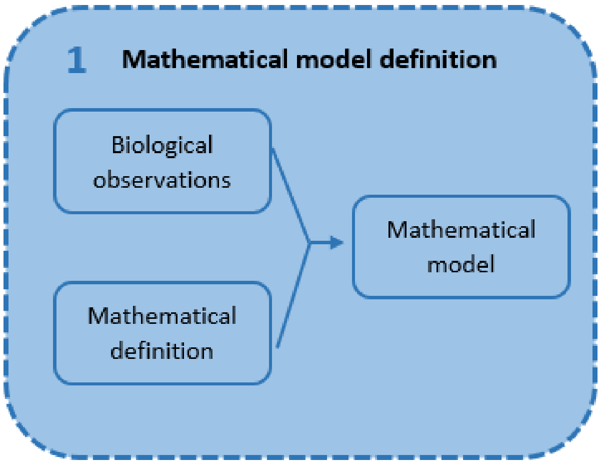


Figure 3.2: Mathematical model definition

the new design. When the number of conflicting objectives increases it is difficult to obtain a solution that would provide a significantly higher output. Hence it is ideal to change a small set of parameters that are linked while keeping the rest of the parameters constant.

2. Selecting parameter to be optimised:

   The next step is to identify the parameters that is correlated with the defined design requirements. The parameters include decision variables (input parameters for the optimisation model and output of the optimisation model), and design constraints (design requirements that must be met for an output to be valid). Design constraints are usually expressed as equalities and inequalities.

3. Defining the objective function:

   An objective function is a performance index which quantifies the quality of a solution by the selected decision variables. Depending on the design requirements objective function could be maximized or minimized. This could be a mathematical function computed using the values of the decision variables or an output from a simulation program.

4. Selecting an appropriate optimisation algorithm: Once the objective function is defined, next step would be to chose an appropriate optimisation algorithm for the search process. Selection of the optimisation algorithm depends on several factors such as the nature of design requirements, design variables (single or multiple input variables) and objective function.

5. Implementing and executing the optimisation algorithm:

The next step is to implement the optimisation program by encoding the decision variables, design constraints and objective function. This can be done in any programming language. The number of times of execution and the number of iterations depends on how efficiently the small change function can search the total solution space. (A small change function is used to introduce a small change to the current solution in order to change the direction of search in order to find a solution with better fitness. More details on the small change function is discussed in section 6.4.2). Usually these parameters should be selected using experimental methods.

As the methodology is focused at identifying alternative designs two main conditions should be satisfied in the design process for any optimisation approach: the best solution acceptance criteria should be designed to accept solutions that are equal in finesses and the history of the heuristic search should be stored.

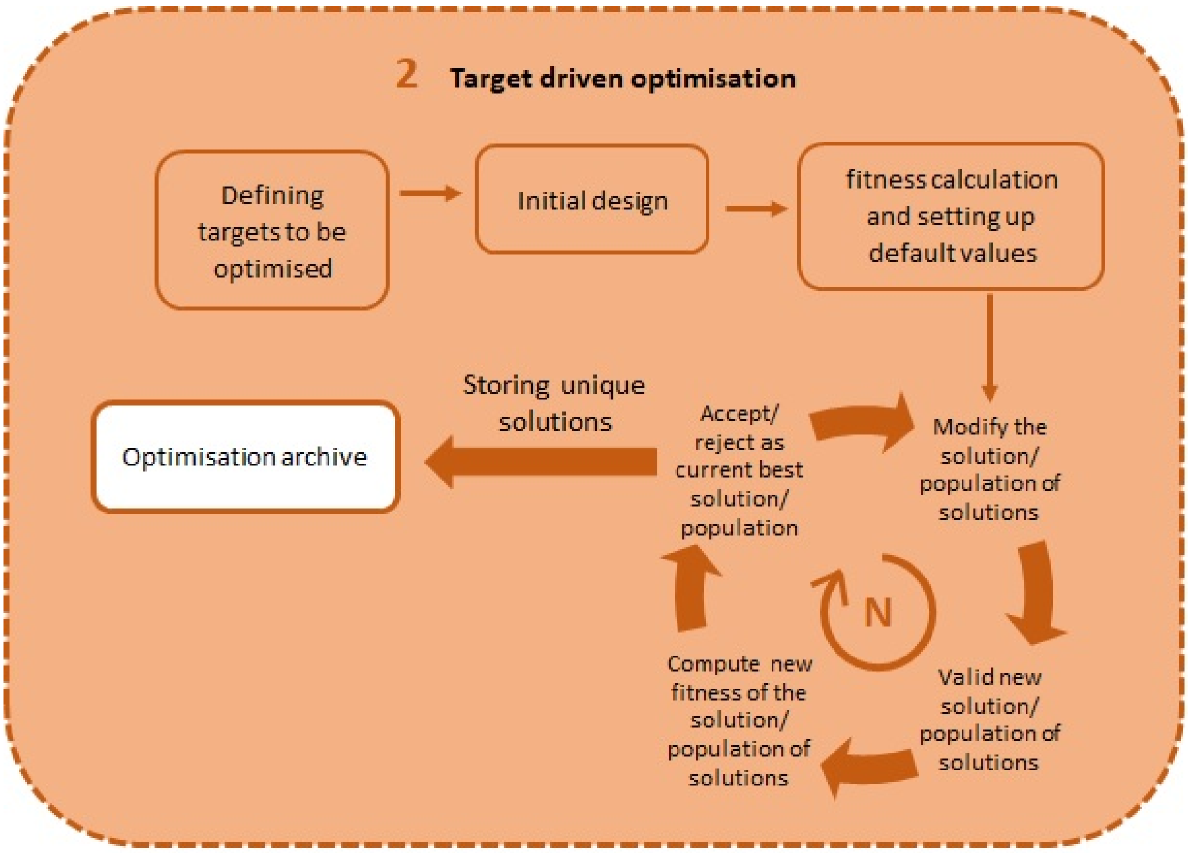

2 Target driven optimisation
Defining targets to be optimised
Initial design
fitness calculation and setting up default values
Storing unique solutions
Optimisation archive
Accept/ reject as current best solution/ population
Modify the solution/ population of solutions
N
Valid new solution/ population of solutions
Compute new fitness of the solution/ population of solutions

Figure 3.3: Target driven optimisation

3.1.4 Analysis of optimisation archives

The history of a heuristic search includes a population of solutions. A population in this context refers to all the solutions explored during the heuristic search. That is all the solutions generated by the optimisation program This population of solutions is a subset of the total solution space for a given problem. Data collected from a single run should ideally contain details of the solution, its fitness, current best fitness and the order in which they appeared in the history (respective iteration the solution appeared). If sufficient number of optimisation runs were completed optimisation history data is going to amount to a large volume. Therefore, in order to analyse this large volume of data effectively data analytics techniques must be applied.

The generational data (optimisation history) collected from the heuristic search will be analysed in mainly two different ways; ranking and profiling of solutions. Ranking of solutions involves ordering the solutions based on decision variables agreed upon previously. The decision variables refer to the variables used in the optimisation process in order to evaluate the fitness of a solution. Therefore, these include input variables, constraints and fitness value of the optimization algorithm. The ordering can be based on single or multiple parameters. This method is useful when there is only one solution for each fitness category as it will aid in selecting sub optimal solutions. Descriptive data analytics techniques can be applied for ranking.

Profiling involves grouping of solutions based on selected decision variables agreed upon previously. For grouping solutions mainly clustering techniques are used.

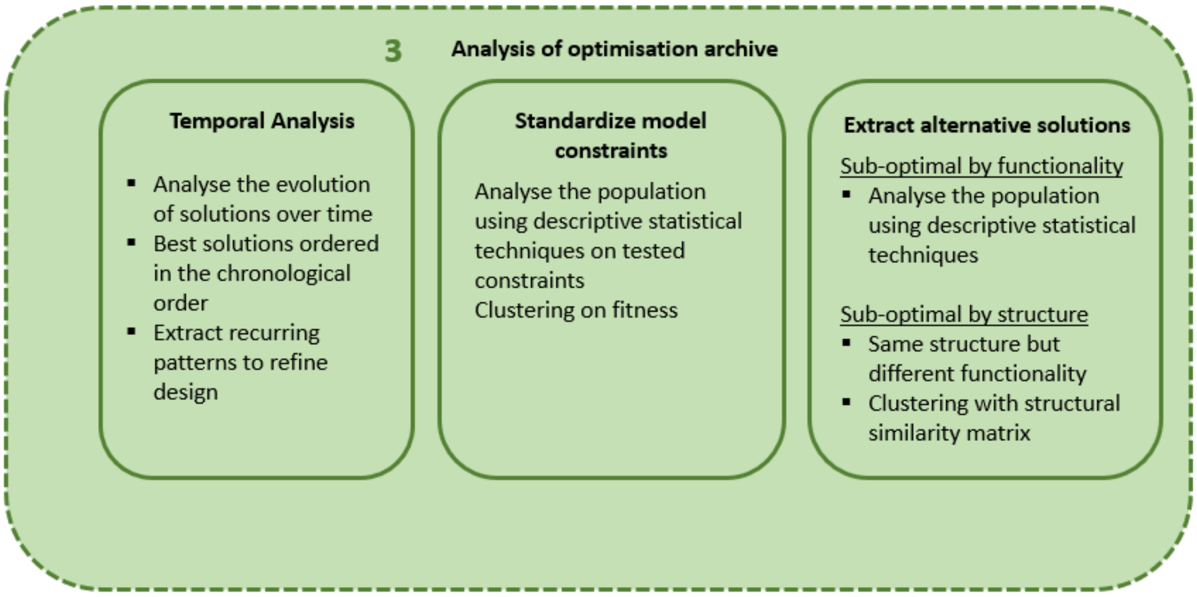

3 Analysis of optimisation archive
Temporal Analysis
Analyse the evolution of solutions over time
Best solutions ordered in the chronological order
Extract recurring patterns to refine design
Standardize model constraints
Analyse the population using descriptive statistical techniques on tested constraints
Clustering on fitness
Extract alternative solutions
Sub-optimal by functionality
Analyse the population using descriptive statistical techniques
Sub-optimal by structure
Same structure but different functionality
Clustering with structural similarity matrix

Figure 3.4: Analysis of optimisation archive

Information in the optimisation archive can be processed in different ways to obtain mainly three outputs. The three outputs are extracting alternative solutions, observe the evolution of solutions and to standardize model constraints.

1. Extract alternative solutions

   Alternative can be defined in mainly two ways; behaviourally different and behaviourally similar but structurally different. It is important to identify the form of alternativeness expected from the analysis as the analytic methods will be different for each approach.

   - Behaviourally different solutions

     Behaviourally different solutions can be identified by sorting the solutions based on the parameter values that are related to the behaviour of the model. Since the optimisation process will be optimizing the behaviour, these parameter values refer to the fitness value (decision variable) of the optimisation algorithm. Ordering the parameters will give an indication of the best and worst solutions. Further, solutions can be clustered based on the parameter values (i.e. decision variable of the optimisation program) to identify groups of solutions with similar behaviour. Grouping solutions based on their behaviour is useful when there are a large number of solutions in order to identify equivalence classes in behaviour.

   - Behaviourally similar but structurally different solutions

     In order to categorise solutions based on structural similarity a similarity measure should be defined to distinguish between two structures. This quantitative measure can be used to cluster the solutions in to groups of structural similarity.

   Work presented in this thesis primarily focuses on solutions which are behaviourally similar but structurally different. The behaviourally similar solutions falls into a equivalence class. However, in certain scenarios alternative solutions may also refer to a sub-optimal solution (behaviour slightly less than the optimal/best behaviour observed). Sub-optimal solutions are important when the structure of the best solution is infeasible to implement in real life (due to higher cost or infeasible modifications). In such cases availability of sub optimal solutions will provide a range of decision options to biologists.

2. Temporal analysis

   The temporal analysis is mainly focused on observing the structural evolution of solutions overtime i.e. within a single run. In a given optimisation run how the structure of the initial solution has changed into the best solution within a single run is analysed in this section. The benefit of this analysis gives the opportunity to identify the sub structures in models which remain constant during evolution. For larger models, when there are a high number of parameters that needs to be changed, limiting the number of parameters that is allowed to change will improve the optimisation time.

Identification of these sub structures takes place within the exploitation phase of the optimisation process (i.e. within a single run). However, in order to confirm the accuracy and consistency (i.e. not a one-off occurrence) of the sub structures, more instances need to be explored. Therefore, temporal analysis will have more significance and accuracy in the exploration phase.

3. Standardize model constraints

Data in the optimisation archive can be used to identify standard values for parameters. In this approach a value for a certain parameter is allowed to change in a plausible minimum and maximum range. Next examine the values which have been used by models which were successful solutions. This will give and indication of the true minimum and maximum values required for the specific parameter and the information could be used to derive algorithms to determine the value for parameters.

Figure 3.5 is a representation of the completed methodology.

3.2 Importance of having a methodological approach to identify alternative solution

Compared with the existing use of memory-based optimisation approaches discussed in Chapter 2 approaches, the methodology presented in this chapter is different due to the following reasons.

Mostly the memory based schemes used in optimisation approaches are used to introduce diversity in the population when selecting a better initial solution for a new run in order to avoid premature convergence. However, the memory scheme has a different definition in the work presented in this thesis. Optimisation history (or memory) is a collection of all the unique solutions that were encountered in the heuristic search within a single run. This collection of solutions make a population of solutions. In a usual optimisation, the history is discarded once the optimal solution is reached. However, a methodology is proposed to extract valuable information from the history in order to improve computational design and reduce the time

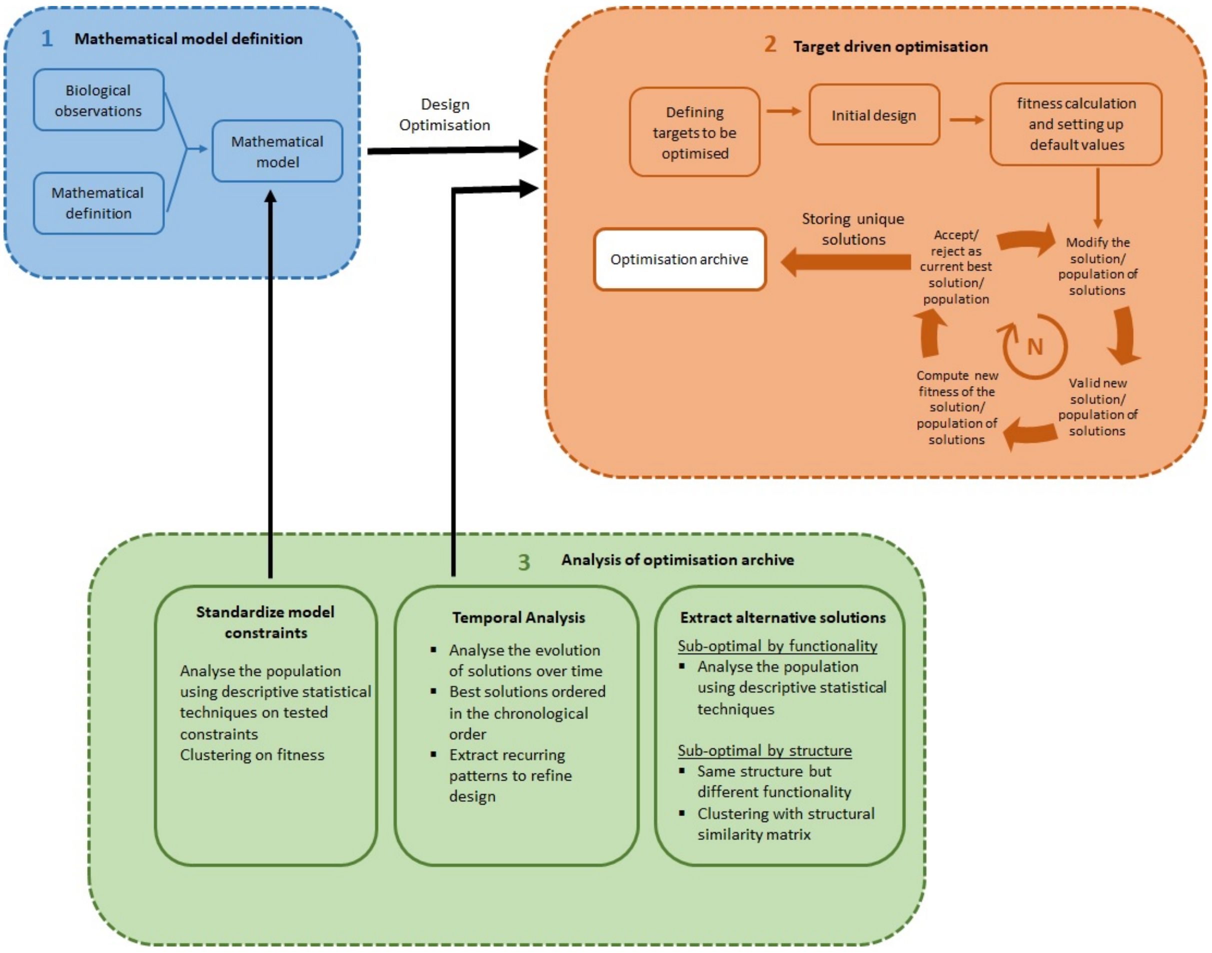

1 Mathematical model definition
Biological observations
Mathematical definition
Mathematical model
Design Optimisation
2 Target driven optimisation
Defining targets to be optimised
Initial design
fitness calculation and setting up default values
Modify the solution/ population of solutions
Valid new solution/ population of solutions
Compute new fitness of the solution/ population of solutions
Accept/ reject as current best solution/ population
N
Storing unique solutions
Optimisation archive
3 Analysis of optimisation archive
Standardize model constraints
Analyse the population using descriptive statistical techniques on tested constraints
Clustering on fitness
Temporal Analysis
Analyse the evolution of solutions over time
Best solutions ordered in the chronological order
Extract recurring patterns to refine design
Extract alternative solutions
Sub-optimal by functionality
Analyse the population using descriptive statistical techniques
Sub-optimal by structure
Same structure but different functionality
Clustering with structural similarity matrix

Figure 3.5: Methodological approach to identify alternative solutions from the history of a computational optimisation program

taken to run optimisation algorithm for the same type of models.

One factor that determines the time spent on searching for the optimal solutions is the size of the solution space. When the solution space increase, the time spent to reach the optimal solution increases as well. The solution space gradually increases when a combination of multiple input parameters are involved in forming a valid solution (in the optimisation process). In such instances, solutions with certain combinations of input parameters are not useful in the overall search process for the optimal solution. The lower fitness values of these unfavorable combinations delay the optimisation algorithm from reaching the optimal solution. Temporal analysis is useful in such instances as it helps to reduce the solution space by constraining the variability.

Through temporal analysis, we can identify sub structures of a model that could be kept constant in order to achieve a better fitness value. By adding more constraints to the model, the search algorithm will only focus on creating solutions that are beneficial towards reaching a better fitness. This eventually reduces the size of the solution space and thus, time taken to search for the optimal solution. Therefore, limiting the variability through temporal analysis will eventually be helpful in executing the optimisation algorithm faster.

Summary

This chapter presented a methodology that enables the extraction of alternative solutions from a computational optimisation output which is used to generate optimal solutions. The methodology consists of three main steps. First a mathematical model of the complex system should be defined. In the second step computational optimisation is applied to obtain the optimal solution for a predefined set of target characteristics. In the second step an optimisation archive is created which contains the history of the optimisation program. In the third step the optimisation archive will be analysed to obtain three types of insights; alternative solutions (behaviourally similar but structurally different solutions), standardising model constraints and evolution of solutions through temporal analysis. The aim of the methodology is to improve computational design process. When applying computational optimisation, the main focus is on the final solution. However, there can be instances when the optimal/best solution is not feasible to implement. In such scenarios alternative solutions (behaviourally sub optimal, behaviourally similar but structurally different) provide feasible options. Speciality of the proposed methodology is that it utilises the output from the optimisation program to identify alternative solutions.

This is a general methodology which could be applied to any instance of computational optimisation being used to improve design of complex systems. The methodology is validated by applying it to three case studies; mathematical expressions, GEM models and DNA walker circuits. These case studies will be discussed in chapter 4,5, and 6 respectively.

Chapter 4

Application 1: Analysing the search history of GEP optimisation to identify alternative mathematical expressions

Introduction

This chapter illustrates how the optimisation-history analysing methodology described in chapter 3 can be employed to analyse the search history of an optimisation search program to identify alternative solutions. The optimisation problem discussed here is focused on searching for the optimal mathematical expressions using Gene Expression Programming (GEP) optimisation technique. Conclusions and limitations of methodology has been discussed at the end.

4.1 Descriptor

The effective employment of the optimisation-history analysing methodology was assessed based on one model case study and two biological case studies. This chapter presents the first case study where the methodology is applied to evaluate alternative solutions in a population made of mathematical expressions generated by Gene Expression Programming (GEP) optimisation techniques.

An alternative solution is defined in this thesis as a solution which is structurally different but behaviourally similar. The focus of the methodology was to analyse the search history generated from an optimisation program to identify alternative solutions. In this sense, mathematical expressions were an ideal example as the population was made of only structurally different yet behaviourally similar solutions. Therefore, this case study can be considered a model case study. Whereas, other case studies provide a more realistic solution space which composed of varying degrees of alternativeness attributing to different combinations of structural and behavioural similarities.

As highlighted in Chapter 1, the main research question I intend to answer through the research is "When using computation optimisation to improve system design, how can alternative solutions be efficiently found by analysing the history of the optimisation approach?". Through the analysis of history of the GEP optimisation approach I aim to answer the following questions:

- Can alternative solutions be found by analysing the optimisation history?
- Can the alternative solutions be grouped based on their structure?

The next section provides a brief introduction to the concepts used in the chapter followed by the application of the methodology to the example scenario.

4.2 Introduction to mathematical expressions and GEP optimisation

4.2.1 Mathematical expressions

Mathematical expressions consists of mathematical operators and operands. In the work presented in this thesis only the four main operators, addition (+), subtraction (-), multiplication (*) and division (/) were used. Operands refer to the the quantity on which an operation is applied. Base 10 numerals (i.e. 0, 1, 2, 3, ..., 9) were used as operands. The number and type of operators were used to differentiate the structure of the mathematical expressions.

The selection of mathematical expressions as a model case study was done based on the following reasons:

- The concept of alternative solutions (i.e. structurally different but behaviourally similar) could be presented clearly with the use of mathematical expressions. For example if the target number is 12 there are numerous ways to obtain 12 using numbers between 1 and 9 and the four main operators addition (+), subtraction (-), multiplication (*) and division (/). e.g. $2*6$, $3*4$, $6+6$, $2*3+3+3+3$, $9+3$, $3*3+3$, $6/2+9$, $6+1+1+1+1+1$ and etc

- A mathematical expression can be depicted as a tree structure. Therefore, it was easier to compare solutions on structural similarities. Further, the structure was similar to the types of data being looked at in the other two biological case studies; GEM models and DNA walker circuits providing an a suitable reference.
- The methodology was focused on analysing search histories generated by optimisation programs in order to obtain alternative solutions. Availability of an existing implementation of an evolutionary program based on genetic algorithm that could generate a mathematical expression for a target value and predefined set of operators and operands (or operators) reduced the overhead of having to implement a program from scratch.

4.2.2 Gene Expression Programming

Gene Expression Programming (GEP) is an optimisation algorithm similar to genetic algorithm. GEP initiates with a populations of individuals, selects them according to a specific fitness, and introduces genetic variation using one or more genetic operators such as mutation, transposition, and recombination. The difference between GEP and genetic algorithm is the representation of individuals. GEP encodes individuals as linear string of fixed length (i.e. chromosomes) and these chromosomes later get expressed as nonlinear entities of different sizes and shapes (i.e. expression trees). Therefore GEP is also known as a genotype/phenotype algorithm. Consider the following algebraic expression:

$$\sqrt{(a+b)\times(c-d)} \tag{4.1}$$

This can be presented as a diagram as follows (see 4.1).

The optimisation process begins with an initial population. The population consists of randomly generated chromosomes. Then the fitness of each individual is evaluated. The individuals are then selected according to fitness to reproduce with modification. This creates a new generation. The individuals of this new generation are, in their turn, subjected to the same developmental process: expression of the genomes, selection based on fitness, and reproduction with modification. The process is repeated for a certain number of generations or until a solution has been found. An in-depth explanation on the optimisation approach and chromosome representation can be found in (Ferreira 2001).

GEP was implemented by Cândida Ferreira. Cândida Ferreira presents this as a commercial software suit called GeneXproTools (Ferreira 2010). It is offered as a 30 day free trial version.

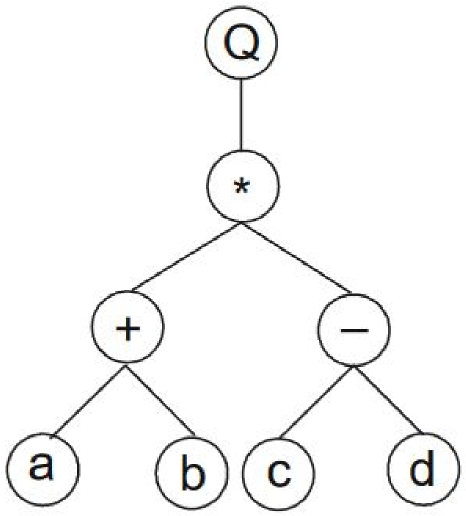


Figure 4.1: Representation of an individual chromosome as an expression tree. In this example a chromosome represents a mathematical function where "Q" represents the square root function. Tree is read from left to right and from top to bottom.

For my work I used a a free and open source implementation of GEP named "grep4" developed by Jason Thomas is available in Google Code Archive (`https://code.google.com/archive/p/gep4j`). There were couple of implementations of GEP in Java and Python. I opted for the Java implementation as I have prior knowledge in coding in Java.

Therefore, this case study was a suitable example as a proof of concept. Application of the methodology to analyse the alternative solutions in the selected solution space will be discussed in the following sections.

Application of the methodology

4.3 Step 1: Mathematical Model Definition

For the purpose of being used in an optimisation program, a mathematical expression was converted to an expression tree (see figure 4.1). Hence, the mathematical expression will be in the form of a binary tree. GEP uses a special format called the GEP gene to encode the mathematical expression as an expression tree.

4.4 Step 2: Target Driven Optimisation

In the second step of the methodology, optimisation is applied on the model in order to select the best solution and optimisation archive is created. The optimisation method used is gene expression programming (Ferreira 2001).

"grep4" program is capable of generating mathematical expressions using four operators; addition, division, multiplication and division. Inputs for the program are

- range of numbers to be used in an expression
- length of the mathematical expression (total number of operators and operands)
- target/ fitness value (expected answer from the expression)

The output of the program is the solution that has the highest fitness value. The algorithm terminates as soon as it finds a correct solution (when the fitness value of the current solution is equal to the target value specified). Therefore in order to obtain different solutions it is required the program to be executed repeatedly. A certain degree of reverse engineering on the code was required to obtain a population of solutions.

A range of experiments were carried out with the "grep4" program. Different target values considered for example 12, 24, 36 and 60. These target values were selected as there are several different ways in which operands and operators could be combined to obtain these targets. The preliminary constraints are as follows:

- Gene length – 20 (it allows maximum of 4 operators and 5 operands in one mathematical expression)
- Operand range -100 to +100
- Operators used subtraction (-), addition (+), multiplication (*), division (/). As the program could to run on one or many operators at a given time, several tests were run on different combinations of operators. For instance each of the four operators on their own, multiplication-division, multiplication-division-addition, addition-subtraction, multiplication-subtraction and etc
- Population size was changed for each operator combination. (As one run of the program resulted only in one solution, the program was re-run for n number of times to create a

population of size n). 50, 100, 200, 300, 400, 500, 1000, 1500, 2000, 2500, 3000 are the population sizes looked at.

Initial testing criteria

- Target value – 12, 24, 36 and 60
- Operator used – Multiplication
- Operands used – positive numbers from 2 to TargetValue/2
- Population size - 1000

12, 24, 36 and 60 was selected as the target value as there are several different solutions for each target value. This can be related to alternative models that results in the same behaviour. E.g.

- 12 – 6*2, 3*4
- 24 – 2*12, 6*4, 3*4*2, 6*2*2

In the initial testing phase I decided to use only multiplication. The program supports all four operators (+, -, * and /) though. The reason I omitted the target number from the operands is that most of the time the algorithm picked the exact value from the list of operands as it resulted in the highest fitness value compared to the target and the program did not result in a mathematical expression. Also for the time being (1* Target value) was disregarded too. Hence it was more meaningful to specify the operands from 2 to [Target value/ 2].

During the testing phase I observed that when the population size is large (e.g. 1000) the algorithm had to iterate less number of generations to find the solution. When the population size is smaller (e.g. 100, 50) it took large number of generations to get to the solution. Hence I set the population size to 1000 in order to reduce the execution time.

For each of the target value the optimisation algorithm was run for 500 and 1000 iterations to obtain the all possible expressions and plotted graphs to see the frequency of different solutions for each target value. When there are large number of possible solutions for a target value (e. 60g) the algorithm had to be repeated for a greater number of times (e.g. 1000) to observe all the possible solutions. I did a basic analysis of the output using R.

4.5 Step 3: Analysis of Optimisation Archive

The final step of the methodology was to analyse the optimisation archive in order to group the alternative solutions. As mentioned in the Descriptor section, the main purpose of the analysis was to identify alternative solutions from the optimisation history and cluster the solutions (mathematical expressions based on their structural similarities).

4.5.1 Identification of alternative solutions

Figure 4.2 represents the alternative solutions for fitness value 12 that were discovered by analysing the optimisation archive. As mentioned in the previous section optimisation archives were created for fitness values 12, 24, 36 and 60. For presentation purposes I will use the fitness value 12 as an example. Fitness value 12 means when a mathematical expression is evaluated, it will result in 12.

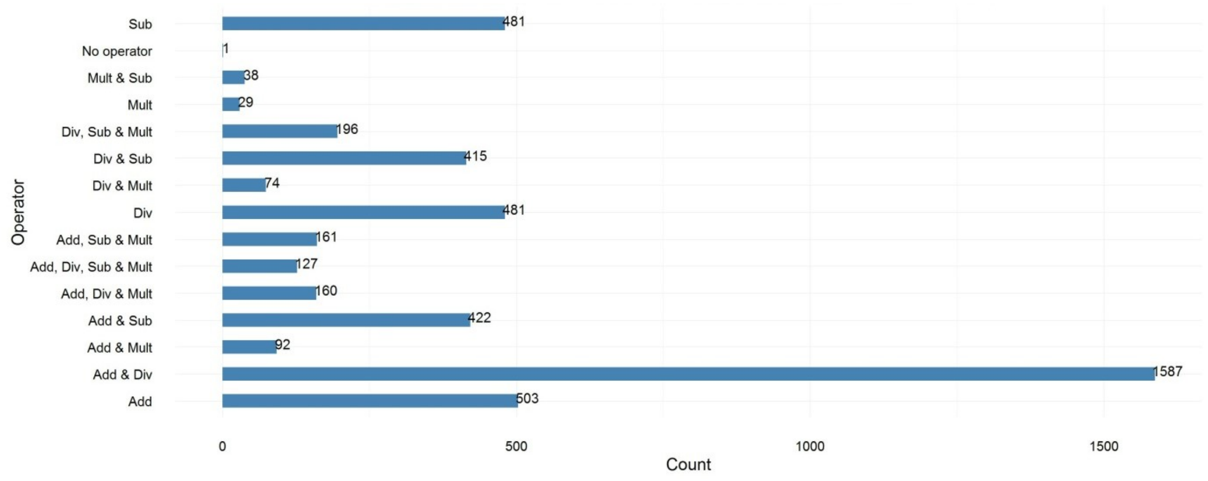


Figure 4.2: The figure represents the number of unique expressions (x axis) which evaluates to 12 and their structural differences based on the mathematical operator being applied (y axis). Operators being used are; Sub (subtraction), Add (Addition), Mult (Multiplication), Div (Division) and a combination of more than one operator. No operator indicates an equation was not used to achieve 12, instrad it directly picked the operand. These expressions were observed within 500 runs of the optimisation algorithm.

According to the figure 4.2, there are several alternative solutions based on the combinations of operators (represented in x axis) used in the expression. The count represents the number of mathematical expressions with each combination of operators. In this comparison operands have been not considered. Operand range used was -100 to +100 which resulted in an infinite set of possible solutions that was difficult to analyse. Hence I chose one operator (multiplication), smaller set of operators (2 - target value/2) and a pre-define set of target values (12, 24, 36 and 60 for the optimisation program which simplified the analysis due to smaller solution space.

Figure 4.3 represents the number of mathematical expressions (x axis) generated by the optimisation program. All these expressions evaluates to 12 and the operator used is multiplication and operands are numbers between 2 - 6. The GEP algorithm has been able to identify all the the possible alternative mathematical expressions that could be constructed under the specified operand and operator constraints.

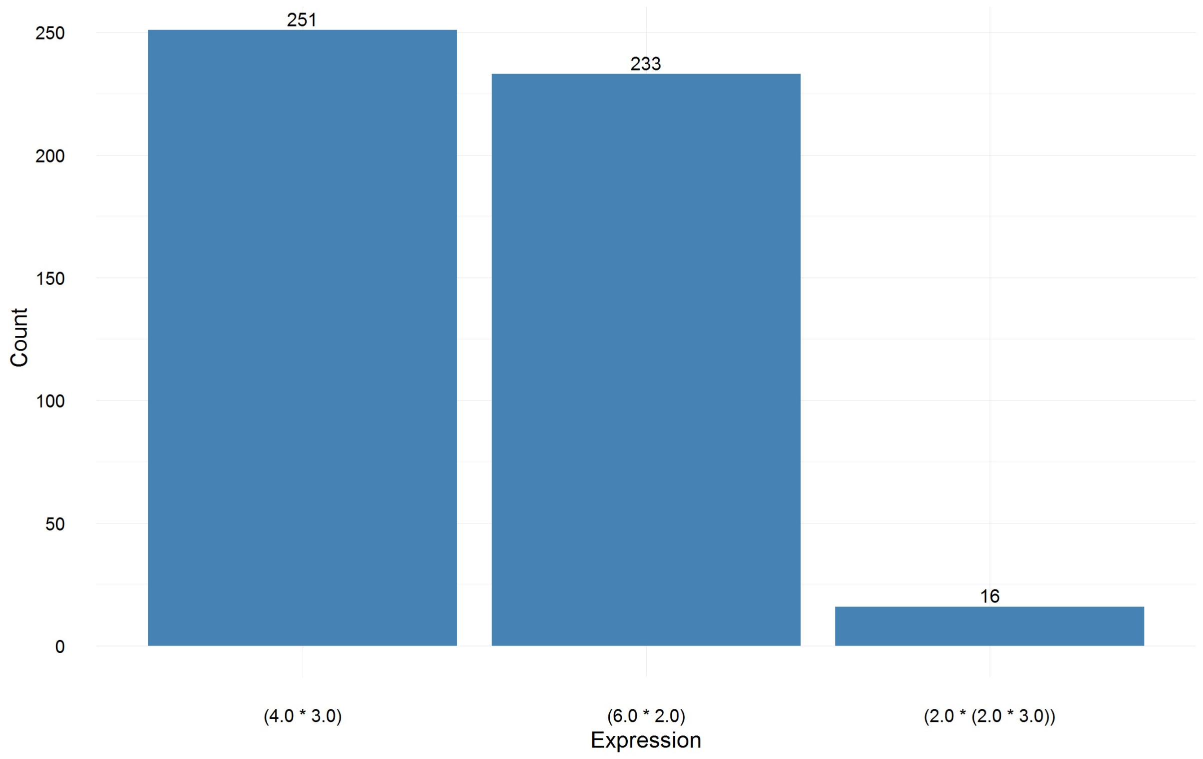


Figure 4.3: The figure represents unique mathematical expressions that evaluates to 12 (x axis) against the the number of occurrences of an expression within the 500 runs (y axis). All mathematical equations use the same operator multiplication and operator range of 2 to 6. These expressions were observed within 500 runs of the optimisation algorithm.

Similarly figures 4.4, 4.5 and 4.6 represent the total solution space the number of times a particular expression was observed during the 500 runs of the optimisation algorithm. One of the research question set for the first example application is "Can alternative solutions be found by analysing the optimisation history"? This analysis has answered the first research question. Using four different target values (12, 24, 36 and 60) I have demonstrated that alternative solutions can be found by analysing the optimisation history.

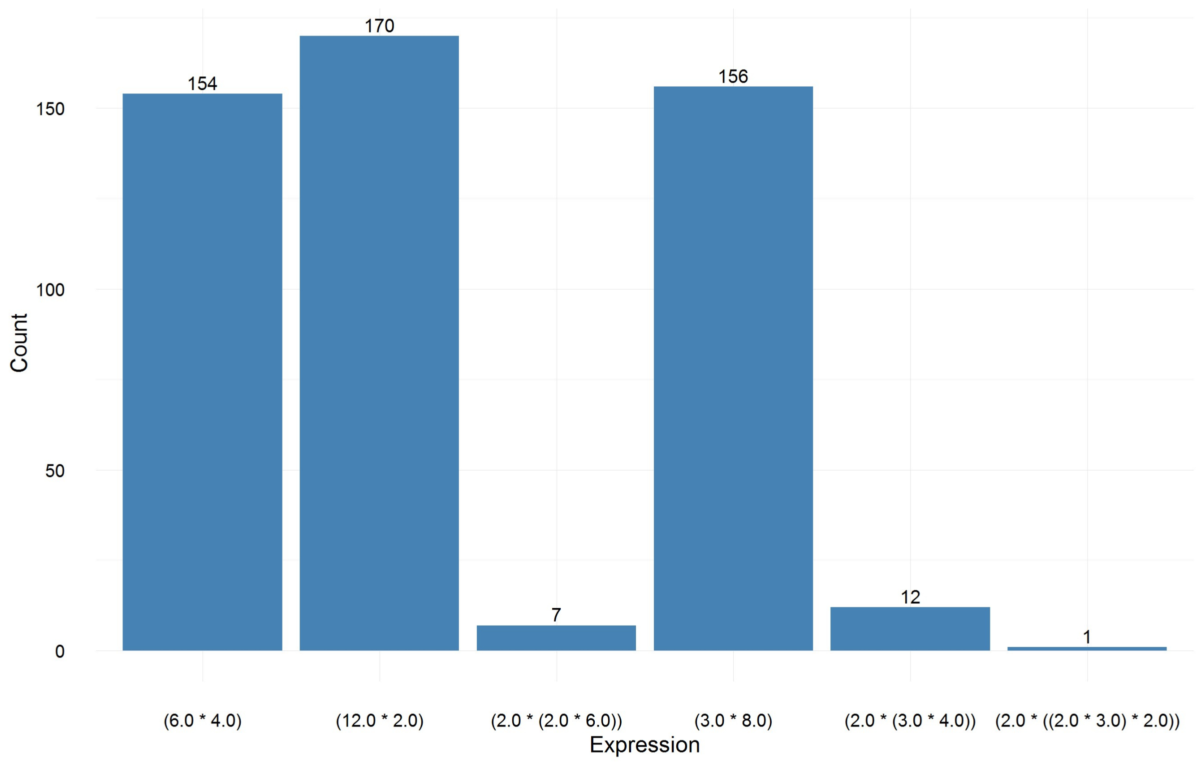


Figure 4.4: The figure represents unique mathematical expressions that evaluates to 24 (x axis) against the the number of occurrences of an expression within the 500 runs (y axis). All mathematical equations use the same operator multiplication and operator range of 2 to 12. These expressions were observed within 500 runs of the optimisation algorithm..

4.5.2 Clustering of solutions by structure

As mentioned in the Descriptor section, the second research question I intended to answer through the analysis was whether the alternative structures could be clustered based on their similar properties?. The main objective for comparing mathematical expressions was to identify behaviourally similar but structurally different expressions.

Structural similarity in this comparison means it should distinguish between expressions by the number of alterations that should be made to convert from one expression to the other. For example, as shown in figure 4.7 after expressing the mathematical expression as binary trees converting expression “A” to “B” requires 6 alterations; (insertion of 4 nodes, updating the value of 2 nodes from “2” to “1”. Therefore expression “A” is very different from expression “B”. However converting expression “B” to “C” requires only 2 alterations (updating the node labels “1” to “2” and “x” to “+”).

In order to compare binary trees of mathematical equations I used the R package graphkernels. Mathematical expressions were expressed as unidirectional binary graphs. However, the

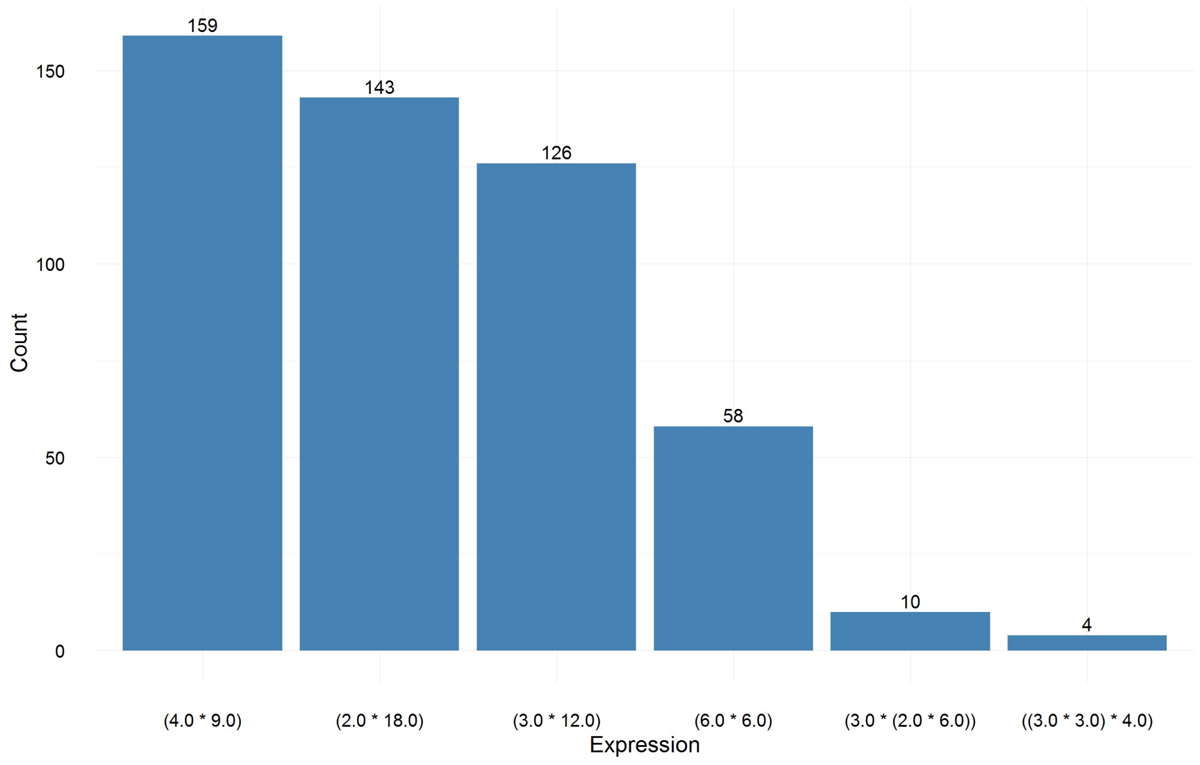


Figure 4.5: The figure represents unique mathematical expressions that evaluates to 36 (x axis) against the the number of occurrences of an expression within the 500 runs (y axis). All mathematical equations use the same operator multiplication and operator range of 2 to 18. These expressions were observed within 500 runs of the optimisation algorithm.

results produced by the graphkernels method were inaccurate as it compared only the numeric values of the node labels and did not take the connections (operator) between nodes into consideration. For example, graphkernels method evaluated two expressions; $(3 + 3)$ and $(3 * 3)$ which equates to two different final values; 6 and 9 respectively, to be structurally similar.

Since graphkernels was not an effective tool for structural comparison in this scenario, I explored the possibility of applying edit distance to solve the problem. Therefore I looked at the existing literature whether edit distance has been modelled for trees and whether there are already existing implementations of the algorithms in R. An algorithm to compute tree edit distance has been published by Kaizhong Zhang and Dennis Shasha. However, the algorithm was implemented in Python. This library is called zss and using the simple distance() method edit distance can be computed. There was no R implementation for this algorithm. I decided to use this algorithm for my comparison as it had the required functionality implemented.

Implementing tree comparison was done in Python. Simple distance() in library zss did not take graph isomorphism in to account. Therefore trees constructed from expressions $(2 * 3)$ and

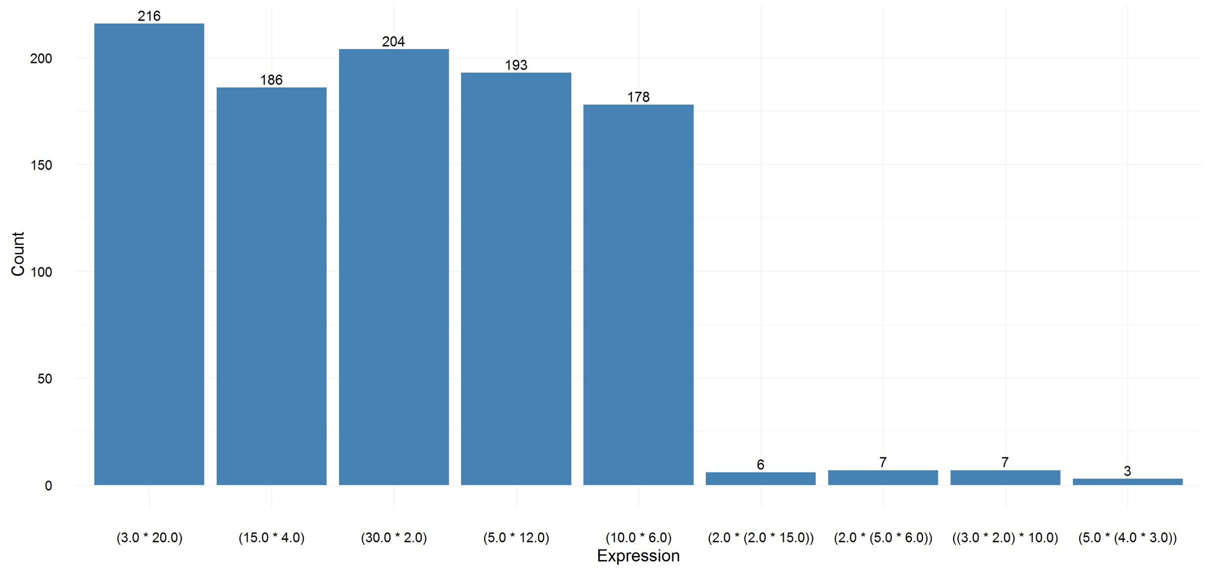


Figure 4.6: The figure represents unique mathematical expressions that evaluates to 60 (x axis) against the the number of occurrences of an expression within the 500 runs (y axis). All mathematical equations use the same operator multiplication and operator range of 2 to 30. These expressions were observed within 500 runs of the optimisation algorithm.

$(3 * 2)$ were treated as two different expressions resulting in an edit distance of 2. Because of this first I had to implement a function that will build a tree in a standard format regardless of how it is presented in the expression. So it will build the same tree for $(2 * 3)$ and $(3 * 2)$. (Source code for this is available in the GitHub folder).

Standard format:

- Each level was ordered so that the value in the left hand side node was smaller than the right hand side node value.
- Operators were given a number between 1001 and 1004 and were sorted accordingly. All the expressions were built using numbers between 0 and 100. Therefore there numeric representation of operators did not mix up with the operands.
  - "+" - 1001
  - "–" - 1002
  - "*" - 1003
  - "/" - 1004
- When the nodes in one level were both similar the node values of the next level was taken

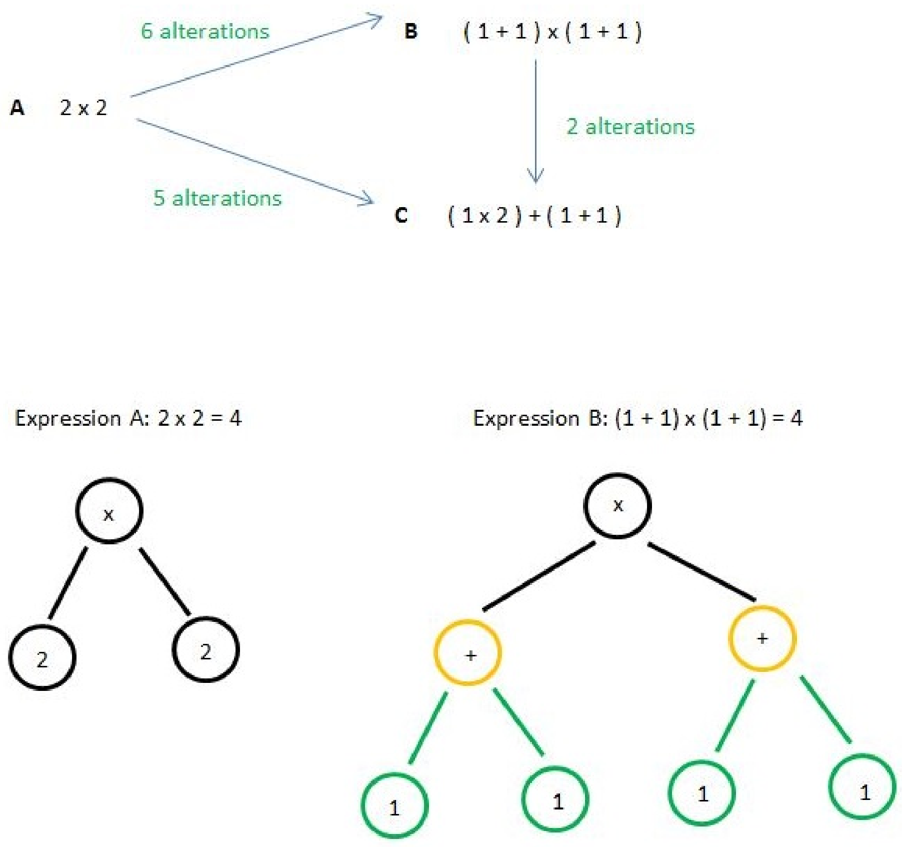


Figure 4.7: Structural comparison of mathematical expressions

into consideration when ordering the nodes. For example the expression $(4+5)x(3+2)$ will be built in the following manner.

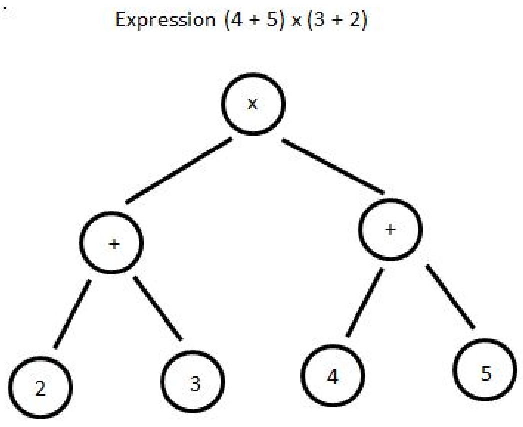


Figure 4.8: Representation of the example expression (4 + 5) x (3 + 2) as a tree structure

Since a large solution space was required for clustering, I chose the alternative solutions generated by the GEP optimisation for the target value 60 when all four operators were turned on and operand range was -100 to + 100. Next the pair-wise distances between expressions were computed using simple distance(). Afterwards this distance was converted into a dissimilarity measure and the expressions were clustered using hierarchical clustering. Figure 4.9 is the dendrogram obtained for hierarchical clustering. As shown in the dendrogram (figure 4.9) there is a clear separation between the types of expressions. Shorter expressions are grouped together and longer ones are grouped separately. Further Silhouette index confirmed the the optimal number of clusters for the clustering as 2 with a Silhouette index of 0.6.

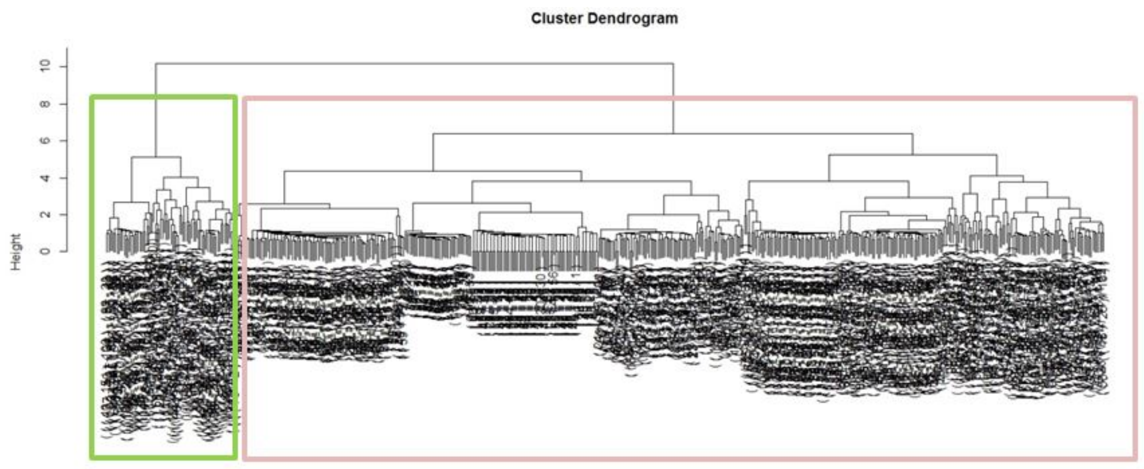


Figure 4.9: Clustering of mathematical expressions which has a target value of 60, operand range of -100 to +100 and operators addition, subtraction, multiplication and division. These alternative solutions were extracted from the optimisation history of the GEP program.

Next, the smallest cluster was analysed further. The reason for selecting the smallest cluster was because it was impossible to view the node lables in the dendrogram since there are over 100 solutions being clustered. Hence the smallest cluster was selected to examine the similarities between members within the cluster and the differences compared to the larger cluster. The smallest cluster composed of all expressions that are of the same length. However what I noticed was as shown in Figure 4.10 that the groupings were not exactly meaningful. The reason for this is edit distance computation weighs update, delete and insert operations equally. Therefore it does not differentiate between significant changes in the expressions of similar lengths. In order to overcome this problem I had to introduce different weightings for insert, delete and update for nodes in a tree. However, fine tuning the distance measure was not carried forward since it was not the main goal of the analysis. The main aim of the analysis was to prove that the alternative solutions could be clustered based on structural similarities.

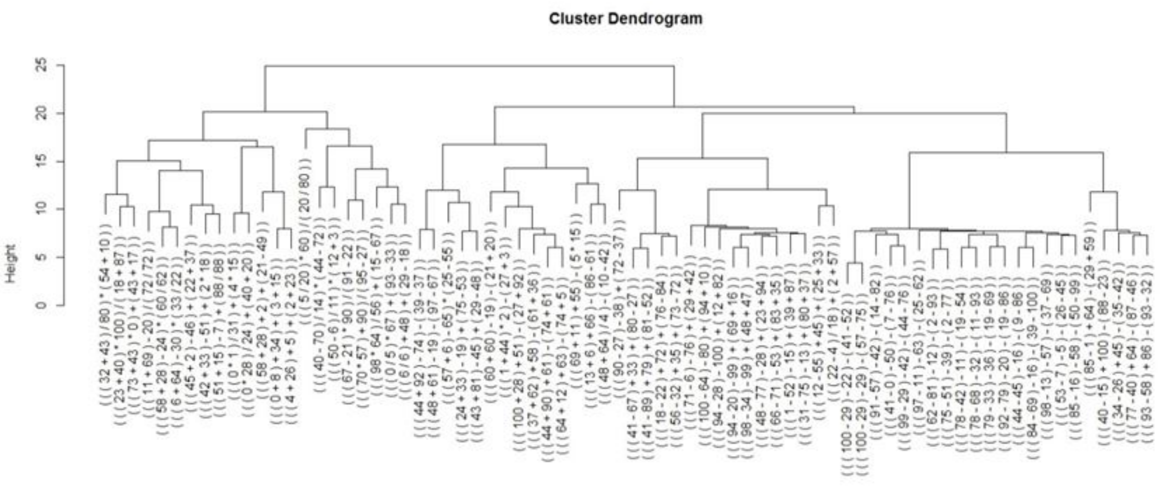


Figure 4.10: Analysis of the smallest cluster observed in the clustering of the alternative solutions for target value 60 in figure 4.9

4.6 Discussion on the results

In order to validate the methodological approach to identify alternative solutions by analysing the history of an optimisation algorithm was applied to an example case study. In step 3 of the methodological approach to identify alternative solutions by analysing the history, I have proposed 3 ways in which the optimisation archive could be analysed to improve the model design (i.e. tree structure of a mathematical expression in the first example application), namely extract alternative solutions, standardize model constraints and temporal analysis.

The goal of the analysis presented in this chapter was to verify whether alternative solutions could be identified by analysing the history of the optimisation program and whether the solutions could be clustered based on the structural similarities. Hence the first example application only demonstrated the use of optimisation archive to extract alternative solutions in the step 3 of the methodology. I have demonstrated above in which ways alternative solutions could be extracted and how they could be clustered based on the structure. Since a mathematical expression should evaluate to a specific target value, the concept of sub optimal solutions is not applicable to this example. For example, when the target value is 12, a mathematical expression that evaluates to 11 or 13 is not a suitable replacement. Therefore, all solutions that were considered in this example were alternative designs based on structure (behaviourally similar but structurally different). Further, standardizing model constraints were not required since the binary tree structure used to represent mathematical expressions was a standard model.

Finally, temporal analysis to optimize the tree structure could not be applied in this example because the expression structure changed based on the operators and operands turned on during optimisation process.

Summary

In this chapter the optimisation-history analysing methodology which was defined in chapter 3 was applied to a case study which uses computational optimisation to generate mathematical expressions. In the first phase of the methodology I have used a predefined mathematical model which is used to model mathematical expressions. Mathematical expression was defined as an expression tree and encoded as a GEP gene. In the second phase gene expression programming (GEP) genetic algorithm was applied to generate the optimal mathematical expression for a specified criteria. The optimisation archive was created by combining the history of different runs of the optimisation program. In the third phase alternative solutions were extracted by analysing the optimisation archive. Further clustering was used to group the solutions based on their structural similarities. This example was used as a proof of concept for the existence of behaviourally similar yet structurally different alternative solutions. Mathematical expressions was the most suitable example. Hence Temporal analysis and standardisation of model constraints were not demonstrable through this example. In the next chapter the methodology will be applied to case study from biology where computational optimisation has been used to optimise the design of genomic metabolic models of E.coli bacteria.

Chapter 5

Example Applications 2: Alternative solutions for GEM models using genetic algorithms optimisation

Introduction

This chapter describes the application of the optimisation-history analysing methodology described in chapter 3 to analyse the history of an optimisation search program designed to obtain the optimal design for a genome scale model of bacteria metabolism. A genome scale model (GEM) is a mathematical representation of a cell that models the metabolic activity (interaction of all metabolites, reactions and genes) for a given organism. The organism being modelled in this work is E.coli bacteria. The optimisation algorithm used for optimisation of the models is a genetic algorithm. Model definition, applying target driven optimisation and analysis of the optimisation archive to identify alternative solutions will be discussed in this chapter.

5.1 Descriptor

The second case study presented in this chapter focuses on employing a genetic algorithm to optimise the design of a GEM model. Optimisation-history analysing methodology will be applied to identify alternative solutions by analysing the optimisation history. The example case study presented in this chapter is the first example with links to computational design in synthetic biology.

As defined in chapter 3, the methodology consists of three phases.

- Step 1: Mathematical model definition

  The first step of the methodology is to define a mathematical model to mimic the behaviour of the complex system. The complex system being modelled in this case study is the metabolic activity of an E. coli bacteria. Section 6.2 provides a brief description of the model which is being used for computational optimisation. The GEM model being used in this case study is a pre-define model of the E. coli bacteria which was implemented by the Palsson group (Monk et al. 2013).

- Step 2: Target driven optimisation

  The second phase of the methodology is the application of computational optimisation to improve the design. The GEM model depicts the metabolic behaviour of E. coli bacteria and a genetic algorithm was used to create a model which yields a higher metabolic activity. The optimisation program was developed by a fellow research student in the team and I have used the history to populate the optimisation archive. Section 6.3 describes the details of the optimisation program and the nature of the optimisation archive.

- Step 3: Analysis of the optimisation archive

  Finally the optimisation archive is analysed in order to identify alternative solutions. Alternative solutions in this work refers to behaviourally different solutions and/or behaviourally similar yet structurally different solutions. The analysis is presented in section 6.6.

As mentioned in chapter 1 the research question I intend to answer through my research is "When using computation optimisation to improve system design, how can alternative solutions be efficiently found by analysing the history of the optimisation approach?" In the second example application where a genetic algorithm is used to optimise the biomass production of GEM models, the main research questions I aim to answer are the following questions:

- Are there duplicate solutions in a generation?
- How does a particular gene get activated (or deactivated) across generations?
- How does the gene activity (percentage of active genes in a model) vary in a generation and how does the activity progress across generations?
- How does the biomass in each model progress through the generations
- Are there any groups of models with the same biomass value but different gene sequences?

- Is there a grouping between models when clustered based on the gene activity and biomass?
- Is there a common core of genes for behaviourally similar alternative solutions?

5.2 Introduction to synthetic biology and GEM models

5.2.1 Synthetic Biology and Re-engineering of Bacteria

Synthetic biology can be simply defined as designing and fabrication of biological components and systems for useful purposes. Today synthetic biology (including genetic engineering) underlies a multi billion dollar industry offering solutions to some of the most intractable problems. The ability insert new combinations of genetic material into (or remove unfavourable genetic information from) micro organisms, animals and plants offers novel ways to produce valuable small molecules into proteins; provides the means to produce plants (Van Den, Van Damme, et al. 2013) and animals (Niidome and Huang 2002) that are disease resistant; tolerant of harsh environments, and have higher yields of useful products (Cameotra and Makkar 1998); and provides new methods to treat and prevent human diseases.

There are two main disciplines in synthetic biology; the design and fabrication of biological components and systems that do not already exist in the natural world (creating artificial life) and the re-design and fabrication of existing biological systems. For example in order to synthetically produce Artemisinin the scientists built a new metabolic pathway in yeast by assembling 10 genes from 3 organisms. This attempt is an example of re-designing of existing biological systems (Yeast). [See (Benner, Yang, and Chen 2011) for further details on synthetic biology].

Bacteria are one of the most extensively used organisms in synthetic biology. The main reason is genome of most of the bacteria have been completely sequenced and partially annotated. This gives the researchers the ability to perform comparative analysis on genomes which would help in identifying the evolutionary patterns, physiology and ecological adaptabilities of different organisms. This helps in deciding on which species and how they could be used in the re-engineering process.

Subsequently several special characteristics of bacteria which make them more easier and economical to be used in genetic manipulations have been mentioned below (Snyder et al. 2003).

- Haploid: Bacteria are haploid (have only one copy or allele of a gene). This gives the advantage of identifying cells with a particular type of mutation.

- Short generation time: Bacteria can reproduce within a very short period of time. For example some *E. coli* strains can reproduces every 20 minutes under ideal conditions. Short reproduction time of an organism gives the ability to do more experiments.
- Asexual reproduction: Bacteria reproduce asexually (cell division) hence all the progeny will be genetically identical to their parents and each other. This is useful in increasing the number of bacteria with a specific mutation rapidly and eliminate extra resources used for cloning.
- Colonies can be grown in Agar plates: Bacteria are small in size and this gives the ability to generate a colony of billions of individuals in a smaller space.

There are mainly two ways in which bacteria can be engineered (mutations can occur). The first method is by direct interventions. In this method mutations are created artificially by following gene editing methods.

- Gene knock-in: adding a new gene to the bacteria
- Gene knock-out: removing or replacing an existing gene
- Gene knock-up: expression of one or more of the genes are increased. This method can result in increasing the activity of reactions of a bacteria.
- Gene knock-down: expression of one or more of the genes are reduced. This could result in reducing the activity of reactions which are controlled by the altered genes.

The second method is mutation driven engineering. Here bacteria is either exposed to radiation or bacteria are allowed to naturally build mutations by putting them in an environment and allowing them to sharing of genetic information.

Direct intervention is the best and fastest way to get intentional mutations because we have control over the mutations of the bacteria. However direct intervention could be done for genes which have been annotated (knows the functionality of the gene). In contrast mutation driven engineering is slow and we do not have control over the mutations that would occur in bacteria. However the random mutations might result in potential mutations which have not been discovered before. It is more suitable when working with genes which have not been annotated. More details can be found in (Primrose and Twyman 2013).

5.2.2 Computational Models of Bacteria Metabolism

A bacteria model is a computational representation of bacteria's metabolic functions. A set of equations describing metabolic reactions: enzymes, substrates and products are referred as a model. Each reaction is associated with one or more enzymes in a logical manner and it is represented as a network or a graph showing interactions between enzymes and metabolites. These metabolic equations can be qualitative (measured in terms of network topology), or quantitative (measured in terms of reaction rates and metabolite concentrations).

These computational models are intended to be used in the design process; use the network model to predict organism's behaviour and optimise the model to desired target behaviour. Hence the optimised model can be used as a computational design; going backwards from reactions to the genes responsible for the enzymes (Reactions ->logical gene formula ->enzymes / proteins ->RNA ->DNA) which can act as guidelines for bioengineering implementation. Publically available whole genome-scale models (GEMs) of metabolism have been constructed by a group in the USA for 55 fully sequenced Escherichia coli and Shigella strains (Monk et al. 2013). These models are in SBML (Systems Biology Markup Language) (Hucka et al. 2003) format and are based on the K-12 MG1655 strain (KEGG GENOME: Escherichia coli K-12 MG1655). Their most recent reconstruction, iJO1366, accounts for 1,366 genes (39% of functionally annotated genes on the genome) and their gene products. Each model consists of on average 2300 metabolites and 2700 reactions and the total genome consists of 4000 genes. In addition models are being constructed at Brunel from in-house data; in total there will be over 200 such models.

Application of the methodology

5.3 Step 1 - Mathematical definition of a GEM model

As mentioned earlier a genome-scale model (GEM)is a mathematical representation of a cell that models the metabolic activity (interplay of all metabolites, reactions and genes) for a given organism. GEM models are used to depict both human and bacteria behaviour. These models have typically been developed for Flux Balance Analysis (FBA) and are constraint-based. (Flux balance analysis (FBA) is a mathematical method for simulating metabolism in genome-scale reconstructions of metabolic networks (Orth, Thiele, and Palsson 2010)).

The particular organism which has been modelled in the GEM model used in this research is the K-12 strain of E. coli (*Escherichia coli*). An E. coli bacteria has over 4000 genes or which around 1400 are involved in metabolism. Further a model comprises of around 3000 reactions and yield about 2300 metabolites (Gilbert et al. 2019). The main GEM model of E. coli K-12 strain used in here was built by the Palsson's group (Monk et al. 2013).

Figure 5.1 is a petri net illustration of the E. coli K-12 GEM model generated using Snoopy (Heiner et al. 2012) software.

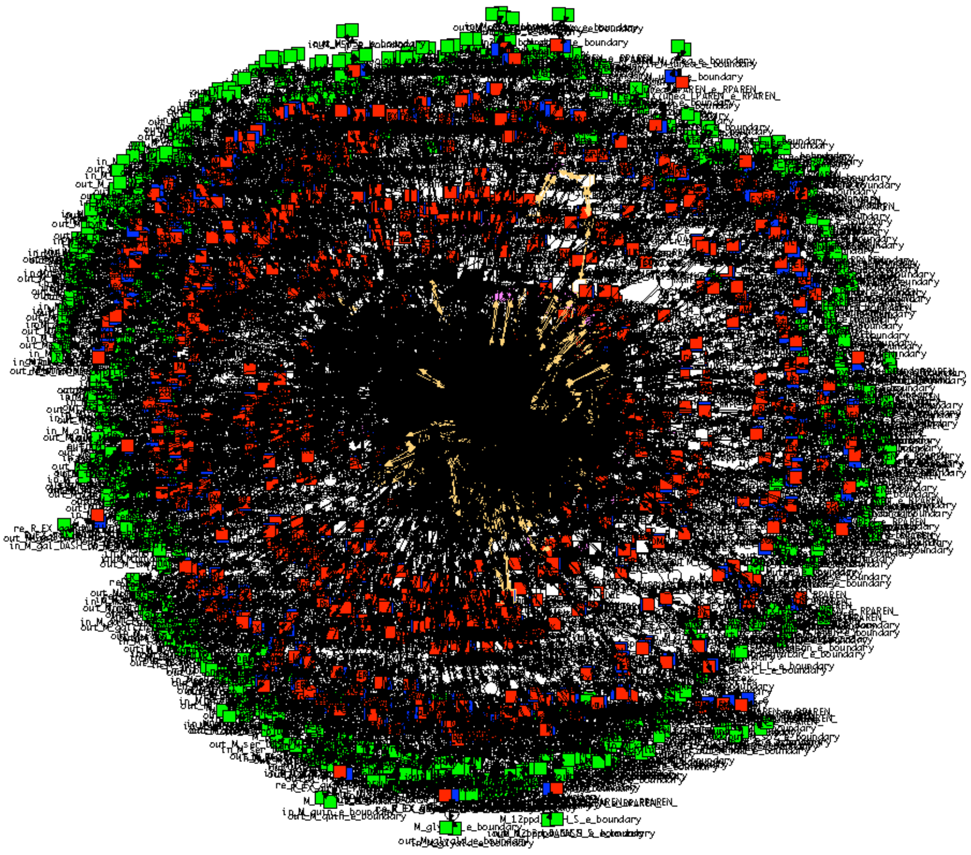


Figure 5.1: This is a Petri net representation of an E. coli K-12 genome scale metabolic model (GEM) from Palsson's group layout generated with Snoopy.

5.4 Step 2 - Target driven optimisation with genetic algorithm

A fellow doctoral researcher's, Bello Suleiman, research was focused on developing a genetic algorithm to optimise the behaviour of GEM models. This sections adopts his work and the optimisation archive was generated using the output from this optimisation runs.

5.4.1 Use of genetic algorithm to optimise GEM model based on growth

Mr Suleiman has employed a genetic algorithm to optimise the design of GEM models. The objective function was aimed at improving the models so that it maximizing the growth of a bacteria. The growth of a bacteria is measured through the amount of biomass a model generates in a FBA. Therefore, when predicting growth in a GEM model, the objective is biomass production, the rate at which metabolic compounds are converted into biomass constituents such as nucleic acids, proteins, and lipids (Orth, Thiele, and Palsson 2010).

According to his design each metabolic model was represented in form of a gene sequence. A gene sequence was converted into a model using a gene-reaction logic table. The gene-reaction logic table has information about the reactions and the respective genes that should be presented in order for the reaction to be activated. Certain reactions require only one gene to be presented; some reactions need more than one gene and some reactions can be activated if either of the required genes is presented.

The genetic algorithm was launched with a population of randomly generated gene sequences of bacterial metabolic models. Each gene sequence was consisted of a fixed number of genes. The gene sequence is a binary string; '1' representing the gene is turned off and '0' representing turned off. Through gene recombination and mutation, a new population of gene sequences were generated. These gene sequences were then converted into a bacteria metabolic model using a gene-reaction logic table. Each model was then simulated using the FBA simulation and the first 450 best models (model with the highest biomass value) were selected. The gene sequences of these models were then again fed to the genetic algorithm as the parents for the next generation. This process was iterated for 250 times (generations) until the population resulted in the highest biomass value.

5.4.2 Generation of the optimisation archive

As shown in figure 5.1 a GEM model resembles a graph data structure. However, the analysis of alternative solutions was focused on looking at the linear data (number and types of genes and reactions, and biomass value) associated with the structures. This means when comparing one GEM model to another GEM model I would not compare it as a graph structure. I would use numerical values such as number of common genes or reactions between two models, biomass value and compute a Euclidean pair-wise dissimilarity distance. Hence the optimisation archive consisted of mainly two types of data.

- Chromosome composition (genes) of each model - genes were recorded as binary data; '1' the gene was present and '0' indicated the absence of the gene, for each model in each generation. Therefore in total there would be Gene sequences for of all the 450 models in 250 generations (450 x 250 models).
- Behaviour of a model - Reactions and the flux value (reaction rate) of all the reactions in a model. The number of reactions in a model varies for each model based on the active genes (each reaction is associated with one or more genes being turned on).

5.5 Step 3 - Analysis of the optimisation archive

5.5.1 Exploratory Analysis of the optimisation archive

Observations from the analysis of gene sequences

A model was composed of 500 genes. There were mainly two groups of genes in a model. Genes which were essential for biomass creation of the model was in the first group and these genes were always turned on. The second group consisted of 139 genes that could affect the biomass production (increase or decrease) and these genes were allowed to turn on and off during optimisation. The optimisation program was in fact utilised to identify the best combination of genes that would increase the biomass production from the second set. Therefore the analysis is mainly focused on the second group of genes.

During the analysis of gene sequences the following observations were made.

- Within a generation, there were no duplicate solutions with the same gene sequence
- Initially most of the 139 genes were deactivated and as the the biomass production increases with the number of generations, more genes were activated and remained turned on. Also certain genes were permanently deactivated over the course as well. Visualising the activation and deactivation of genes using a heatmap was quite useful to observe the pattern. Figure 5.2 is a visualisation of the first generation (Generation 0) genes and it can be clearly seen the randomly activation of genes. However in the last generation (generation 249) a significant number of genes remain turned on for all the models which can be seen in figure 5.3. Figures 5.4 and 5.5 are illustrations of the activity of individual genes.

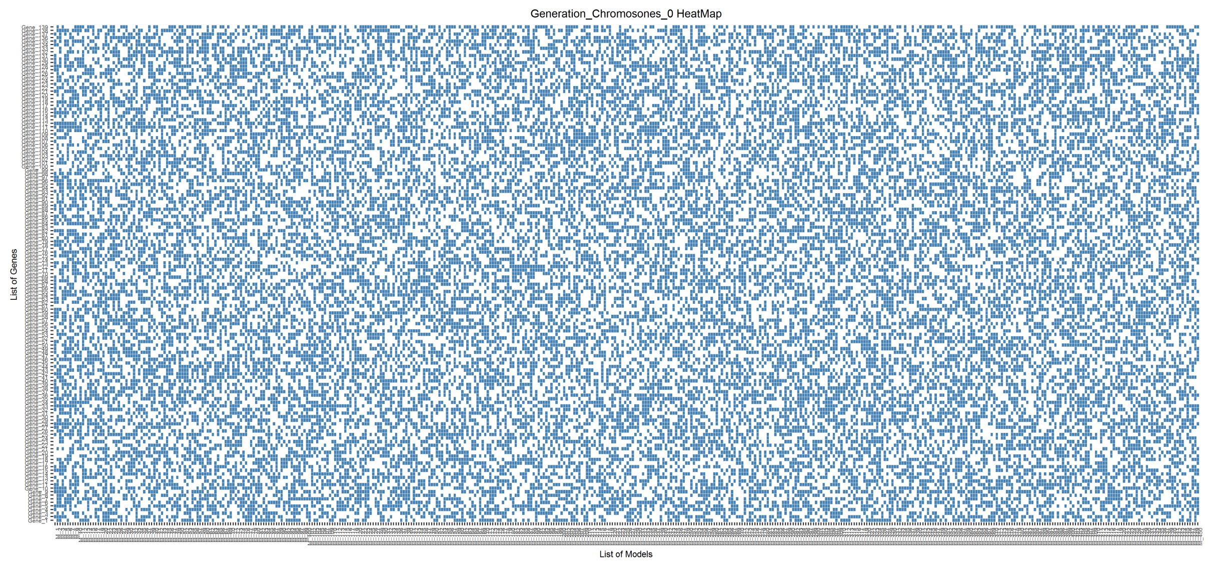


Figure 5.2: Heatmap of activated (blue) and deactivates (white) genes at generation 0. X axis represents the list of models and the y axis the individual genes. Genes have been activated randomly.

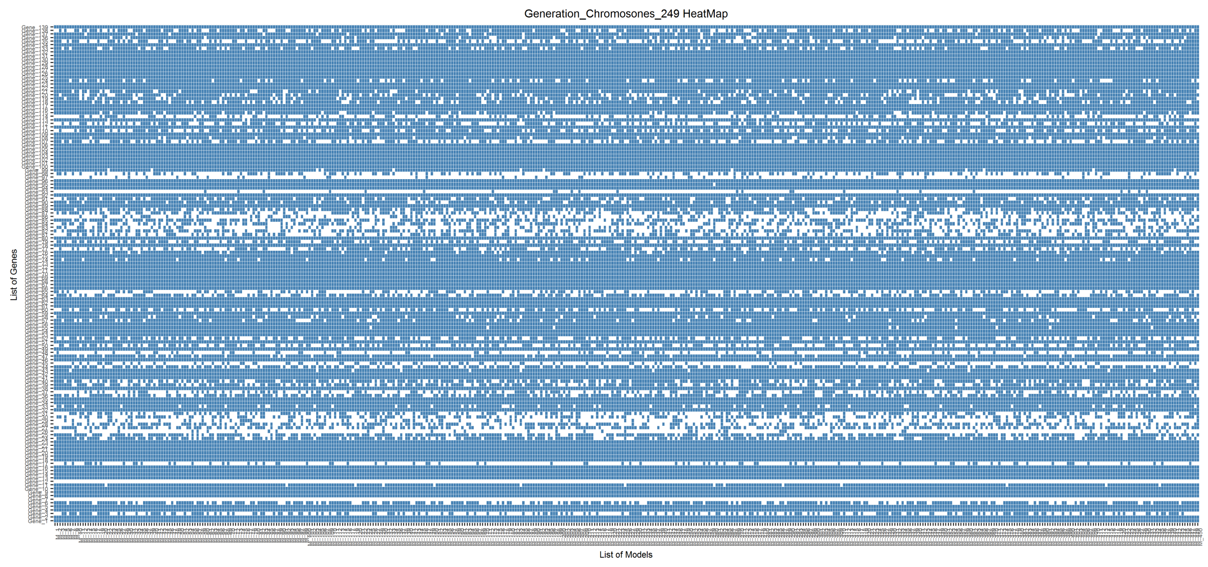


Figure 5.3: Heatmap of activated (blue) and deactivates (white) genes at generation 249. X axis represents the list of models and the y axis the individual genes. There is a clear pattern in activation/ deactivation in certain genes

- As the gene activity (the number of active genes as a fraction of the group of 139 genes) increased, the biomass production increases as well. Figure 5.6 represents the gene activity and biomass production at generation zero. It can be see that the average gene activity is about 0.5 with a high variability within models in generation zero. Also the biomass production is zero. Figure 5.7 illustrates the gene activity and biomass production at

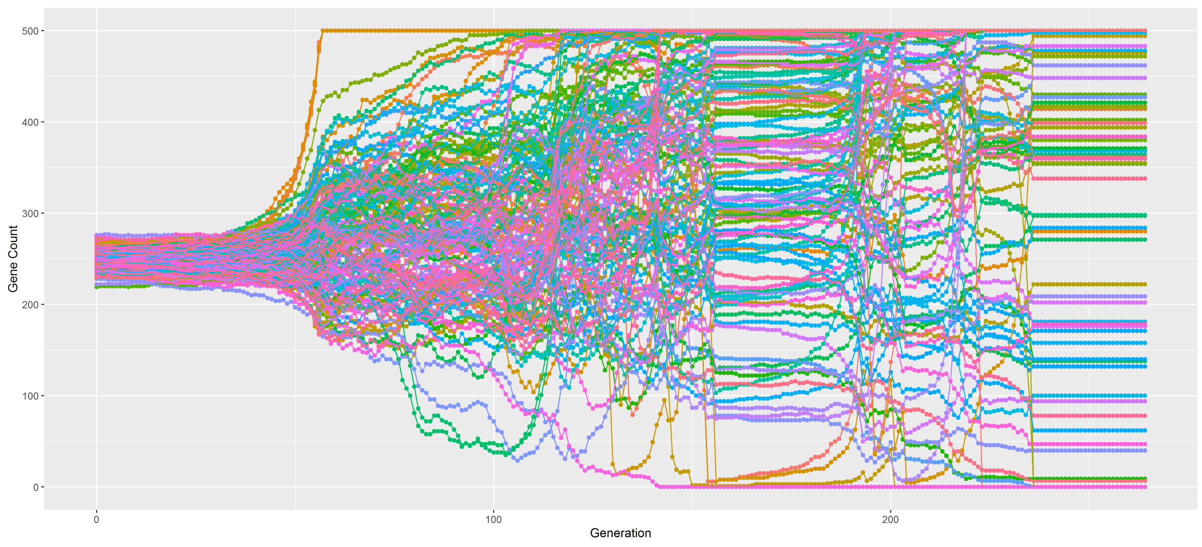


Figure 5.4: The graph represents how the number of activated genes in the models varies as the optimisation progresses. x axis represents the generation no (1 - 250) and y axis represents the number of active genes in the models in the current generation. At the beginning there is less genes activated in the models (which results in a low biomass value) and gradually the number of active genes increases (causing biomass to increase in value). Finally towards the end of the optimisation process gene variability in models becomes constant..

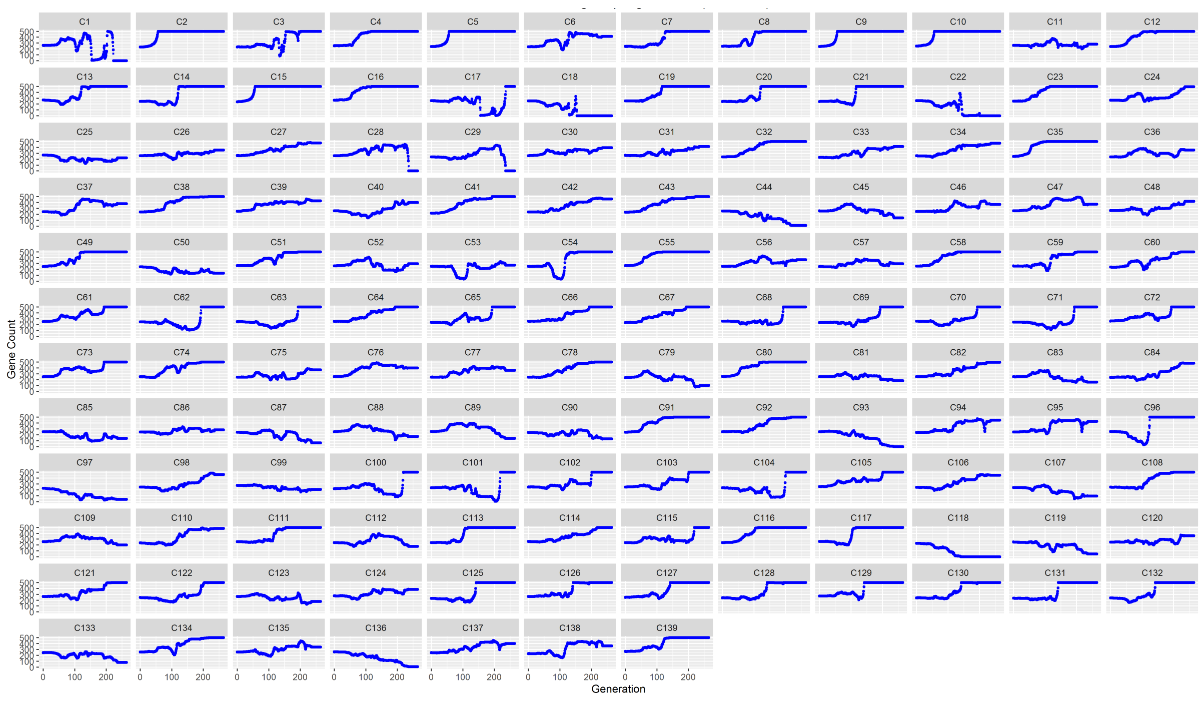


Figure 5.5: Gene activity over the 249 generations for each of the 139 genes

generation 140. Biomass production has gone up significantly and there models which produce varying degrees of biomass. Also the average gene activity has moved up to about 0.75 while variability in activity across models have reduced. Figure 5.8 is a visualization of the last generation and biomass for all models have become constant. Also the average gene activity has slightly increased.

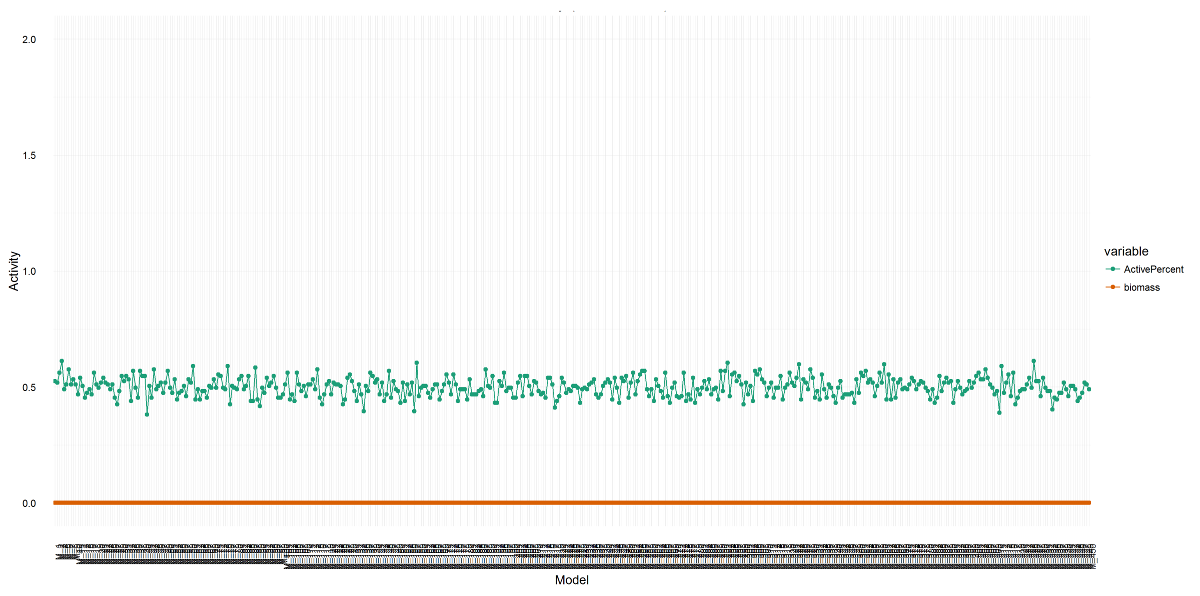


Figure 5.6: Gene activity (in blue) and biomass production (in orange) for each model at generation 0. The average gene activity is about 50% with a high variability across models and biomass generation is zero

- Clustering of models based on their gene composition was performed for each generation as well. The pair-wise similarity measure used to compare two models was the number of common active genes between a pair of models. The main purpose of clustering was to determine whether a grouping based on structure (gene composition) could be used to distinguish between different groups of bacteria models and utilise the insights to improve the optimisation process and computational design process. Figure 5.9 illustrates the grouping of models based on the gene composition.

Observations from the analysis of reactions

In a GEM model, the number of reactions are determined by the type of active genes. Hence the number of reactions in models changed often. During the analysis of gene sequences the following observations were made.

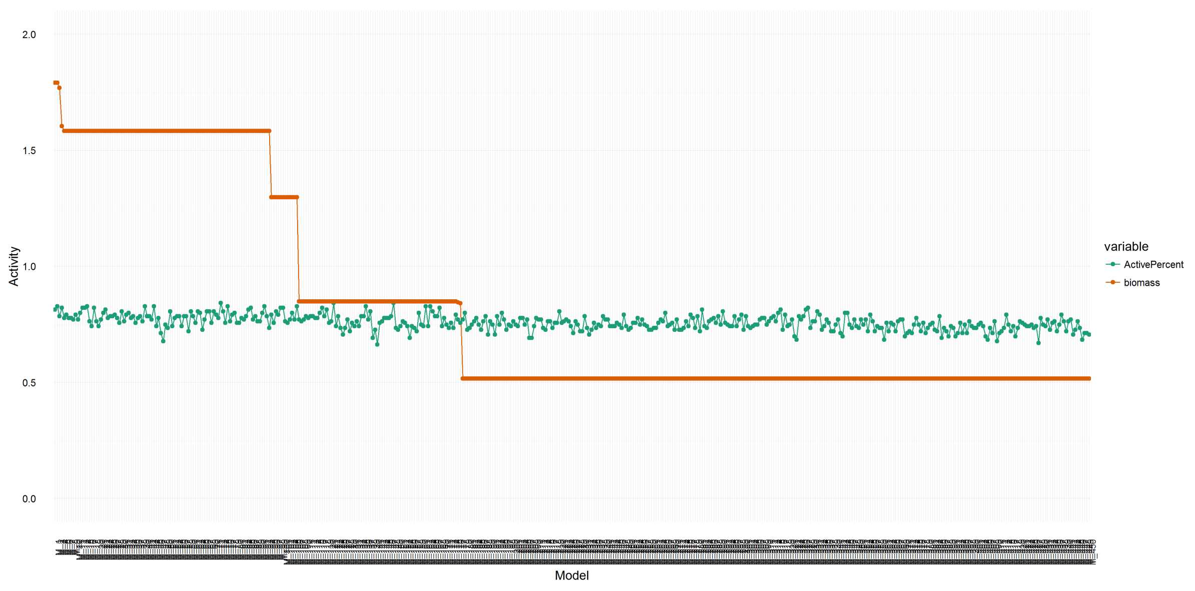


Figure 5.7: Gene activity (in blue) and biomass production (in orange) for each model at generation 140. The average gene activity is about 70%.

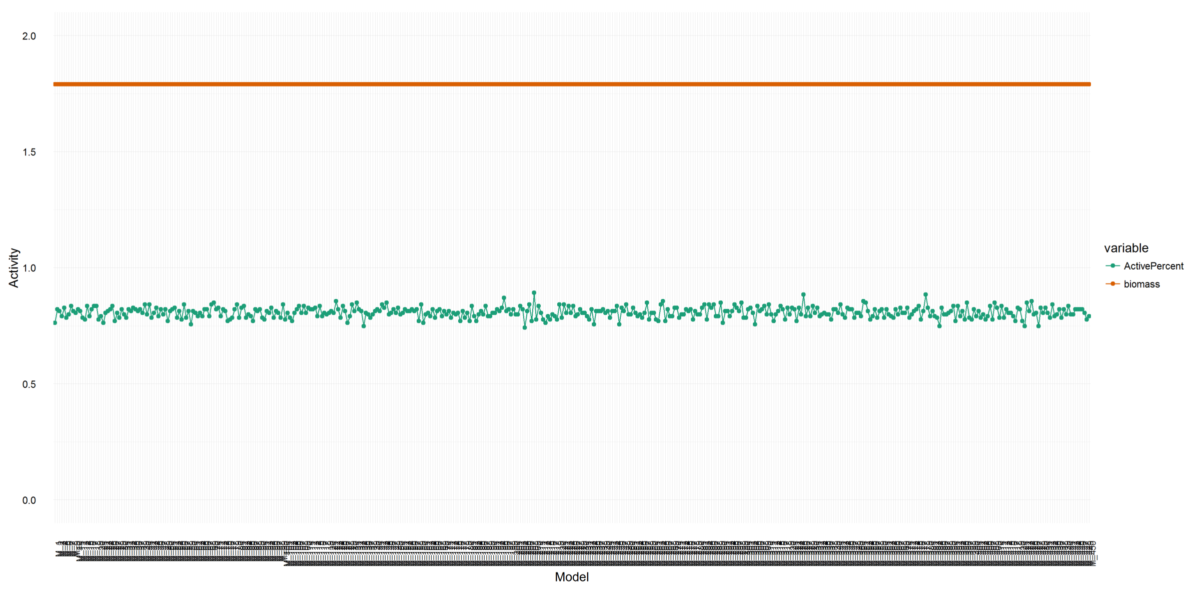


Figure 5.8: Gene activity (in blue) and biomass production (in orange) for each model at generation 249. The average gene activity is about 80%

In the initial generations most of the gene sequences did not result in working models (no biomass value). The number of the non-models gradually decreases and from the 50th generation onwards models started to generate a biomass. Biomass for all the models in the first 50 generations remains same at value 0. Then the biomass value gradually increases and reaches a

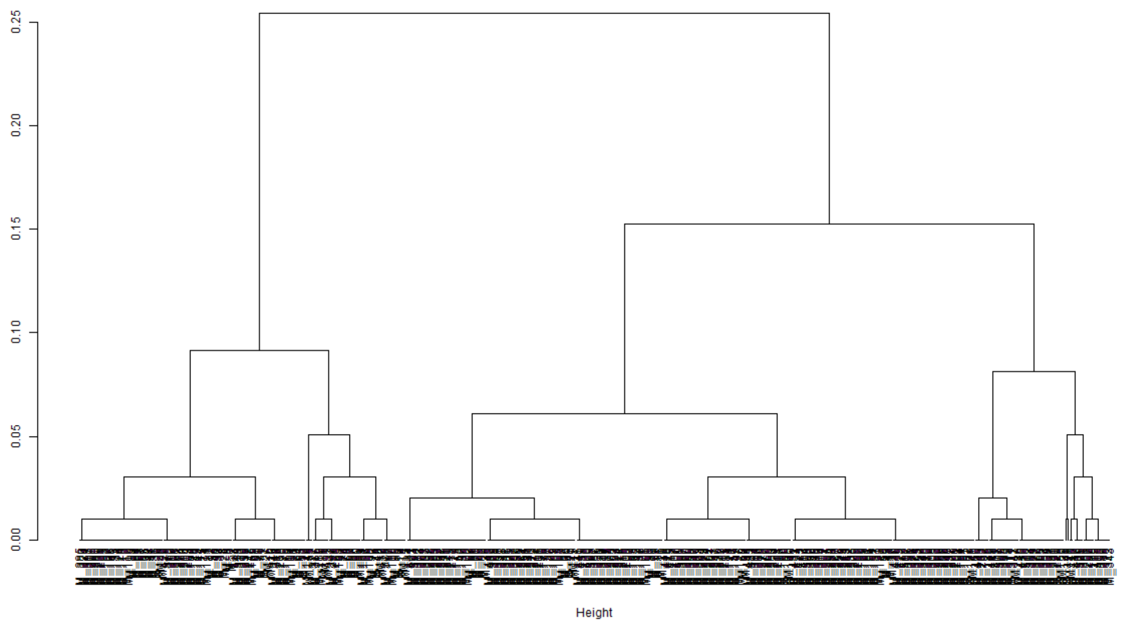
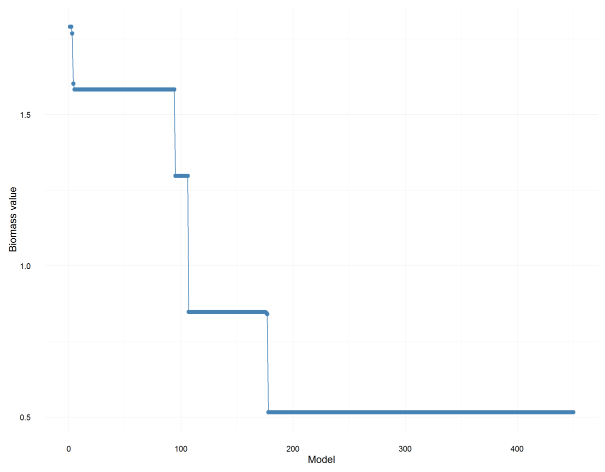
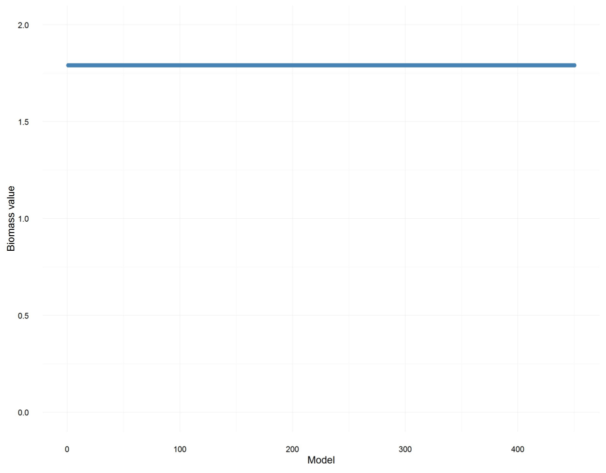


Figure 5.9: Clustering of models at generation 140 based on pair-wise similarity (the number of common active genes between a pair of models)

maximum if 1.79. From generation 152 to 249 all the models have the same biomass value 1.79 (see figure 5.10).



(a) Biomass activity in Generation 140 - 7 distinct values of biomass values $(1.79, 1.76, 1.60, 1.58, 1.29, 0.84$ and $0.51)$ observed in this generation



(b) Biomass activity in Generation 249 - only one biomass value is observed in this generation 1.79

Figure 5.10: Comparison of biomass activity in generation 140 and 249 of the optimisation process

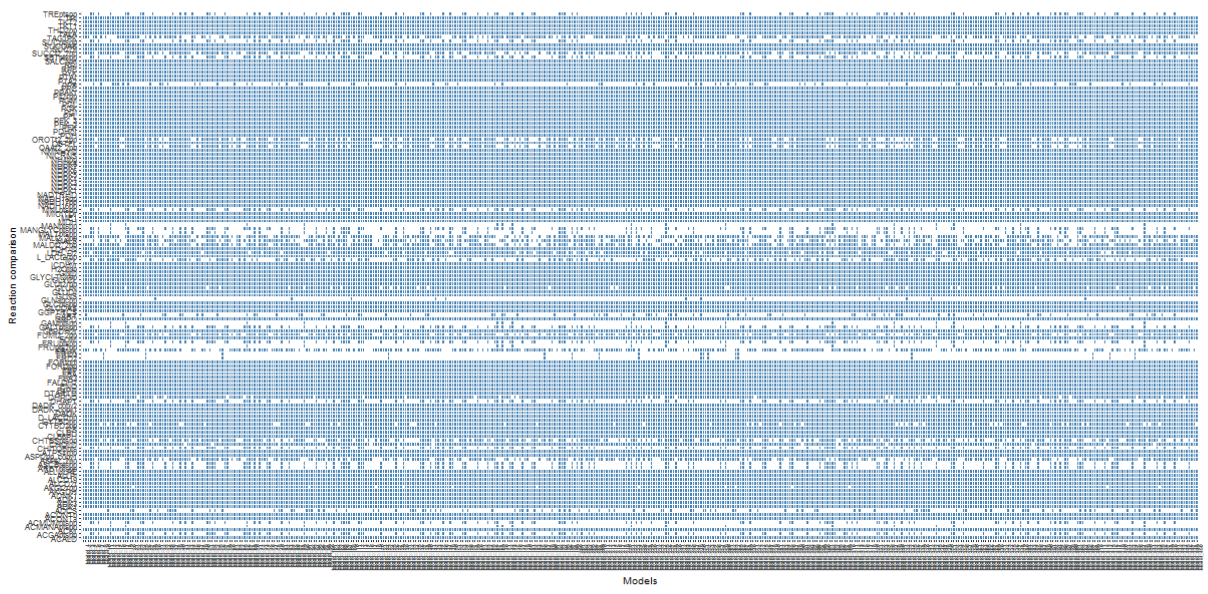


Figure 5.11: Reaction profile at generation 249. The active reactions are highlighted in blue and inactive reactions as blanks

5.5.2 Identifying alternative designs of GEM models

As mentioned earlier there are mainly two types of alternative solutions which are considered in this thesis.

1. Behaviourally similar but structurally different solutions
2. Behaviourally sub-optimal (output is lower than the best solution)

The first type of solutions are a equivalence class based on behaviour. However, in certain instances the best solution might be costly to implement or it might contain modifications that are impossible to perform in real life. Therefore, the biologists would benefit from having an alternative solution that might be slightly worse in performance compared to the best solution but easy and fast to implement. In such scenarios sub-optimal alternative solutions are ideal. Further there could be equivalence classes based on behaviour within these sub-optimal solutions.

In order to demonstrate both aspects of alternative solutions a generation 140 was picked as an example. Generation 140 bears evidence of all the biomass changes took place within a single run of the genetic algorithm. Figure 5.12 illustrates the different levels of biomass values observed at generation 140. The slightly varying biomass levels indicates the existence of sub optimal solutions.

Next the models in generation 140 were clustered based on Biomass similarity, number of

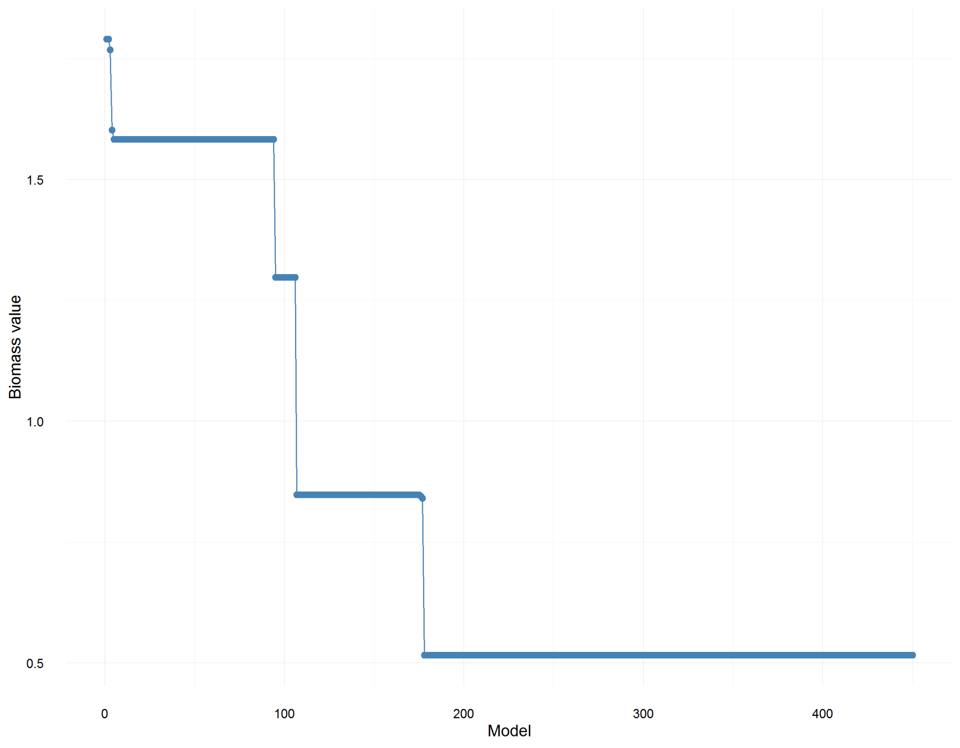
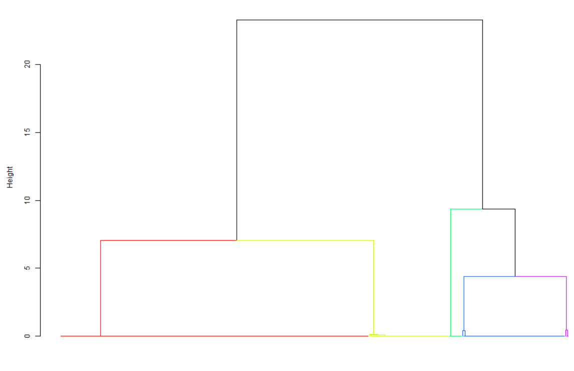


Figure 5.12: Biomass activity in Gen 140

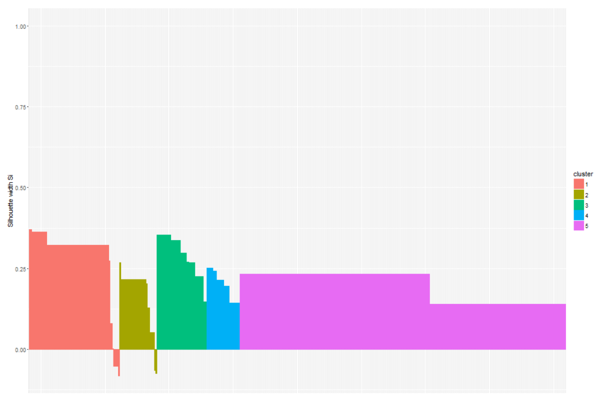
reactions and number of common reactions. Hierarchical clustering method was used to cluster and validation of goodness of clusters were done using Silhouette index. The clustering results in 5 clusters which was validated using a Silhouette value of 0.22. This information is illustrated in figure 5.13.

The following table 5.1 includes the details of the elements in each cluster. The 'common core (genes)' is the number of genes common in all the GEM models in a particular cluster. Average biomass value and the size of common core (genes) supported the claim that alternative solutions (different behaviour and different structure) exists within the solution space. Hence these alternative solutions are sub-optimal with a stratification in biomass value. The availability of solutions with varying degrees of behaviour (the average biomass production varies only slightly between the groups) can benefit the design process by providing sub optimal in the unlikely event of the optimal solution is not feasible to implement.

Further, models in a single cluster provides evidence for the structural alternativeness (solutions with same behaviour yet different structure). While all GEM models are unique in structure (composition of genes and reactions), models within a group share a common core. This common common core is different across clusters. Figure 5.14 is an illustration of the



(a) Dendrogram for hierarchically clustering based on biomass - number of clusters 5



(b) Silhouette plot for the clustering - average silhouette width 0.45

Figure 5.13: Clustering of GEM models in generation 140 based on biomass

common core genes in each cluster. While most of the 139 genes are present in all the clusters there are differences between clusters as well.

Cluster No	No of models in each cluster	Common core size (genes)	Average biomass	Minimum biomass	Maximum biomass
1	3	93	1.78326	1.76866	1.79056
2	91	53	1.58327	1.58306	1.60258
3	12	65	1.29750	1.29750	1.29750
4	71	38	0.84763	0.84063	0.84778
5	273	36	0.51625	0.51625	0.51625

Table 5.1: Clustering of Generation 140 models based on biomass production resulted in 5 clusters. The tables contains the details of elements in each cluster.

Figure 5.15 is an illustration of clustering the GEM models in generation 140 based on the pairwise common reactions. Hierarchical clustering was applied and the goodness of clusters were validated using Silhouette index. When clustered based on structure it results in 5 groups as well which is validated by a Silhouette index of 1. The differences in the core reactions in each of the clusters are presented in figure 5.16. While all models share a certain number of reactions, the common reactions between clusters are different across the 5 groups.

5.6 Discussion on the results

As mentioned in the Descriptor the main research question answered through the research is can alternative solutions be found by storing and analysing the history of an optimisation approach? In the example of a genetic algorithm being applied to improve the biomass production of a GEM

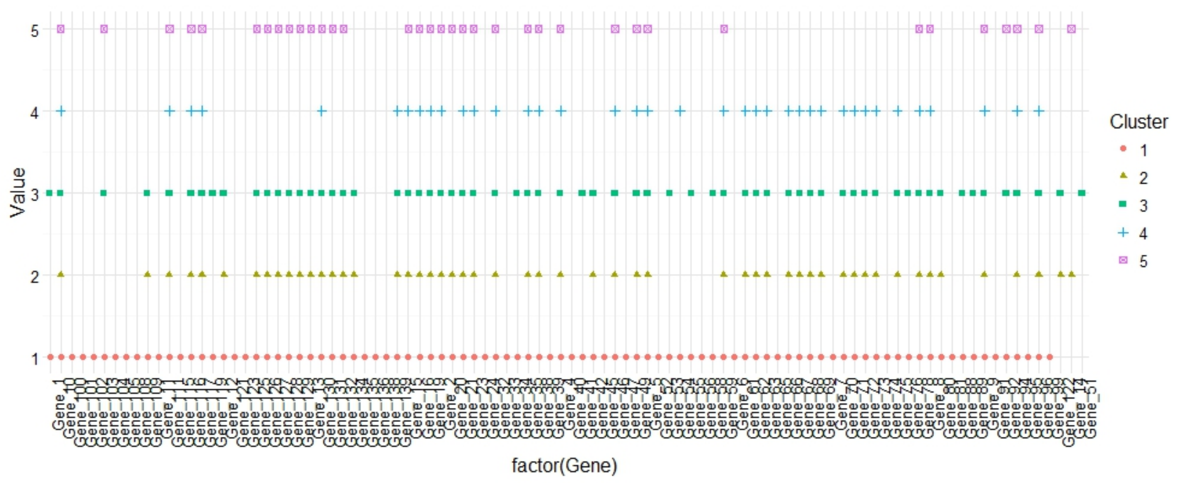
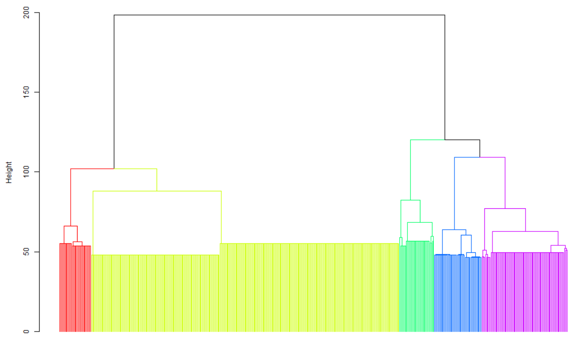


Figure 5.14: Presence of common core genes within each cluster. The GEM models in generation 140 was clustered based on biomass similarity which resulted in 5 clusters. The x axis presents the 139 genes and the y axis indicates the number of the cluster

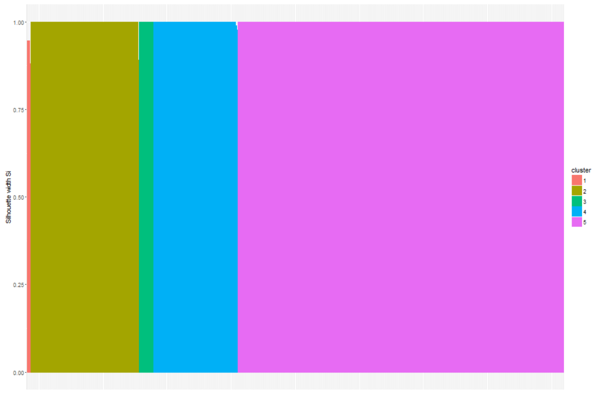
model of E.coli bacteria, several questions were answered during the analysis of the optimisation archive step. As mentioned in chapter 3, analysis of the optimisation archive consists of three sub outputs. Availability of these three outputs depends on the problem the methodology is being applied.

The first output is extracting alternative solutions. Prior to extracting alternative solutions, an analysis was done on different types of model data collected in the optimisation archive. It was found that all models in the 250 generations have a unique gene sequence to one another. Therefore there were no duplicate solutions. Genes are activated randomly at the beginning (see figure 5.2), however, as the optimisation process progresses and the certain genes remain turned on with the increasing biomass values (see figure 5.3). Also the number of genes turned on as the optimisation progresses from generation 0 to 249 (see figure 5.4. The reason for this is the active genes determine the number of active functions. As the number of active functions increases the biomass production increases as well. In generation 0, on average 50% of the genes in each model were activated (5.6) and at the end of the optimisation on average 80% of the genes were turned on for models with the highest biomass value (see figure 5.7).)

The goal of the optimisation program is to optimise the biomass production of GEM models. As shown in figure 5.6 at the beginning of the optimisation program biomass is zero. As the optimisation progresses the new genes get activated resulting in increase production of biomass. The highest biomass value recorded was 1.79 and all models in the final generation produces the same biomass (see figure 5.7). Since there were no duplicate gene sequences within a gener-



(a) Dendrogram for hierarchically clustering common reactions



(b) Silhouette plot with a Silhouette value of 1

Figure 5.15: Clustering of GEM models in generation 140 based on common reactions. The Silhouette index resulted in a value of 1 when the number of clusters was 5.

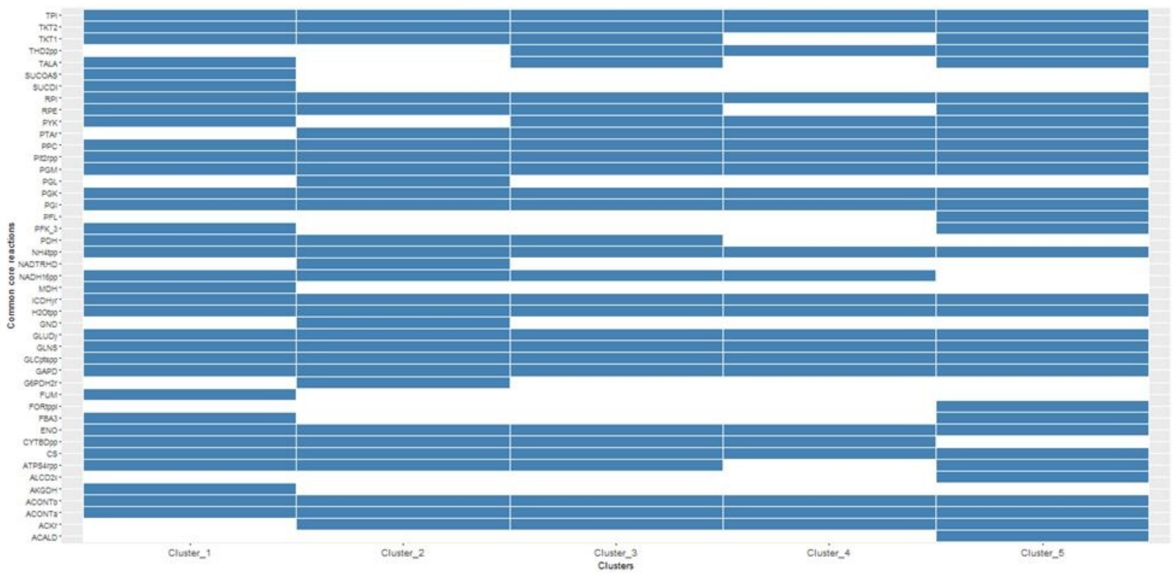


Figure 5.16: The heatmap presents the availability of core reactions within the models in each cluster.

ation, the GEM models in the last generation are behaviourly similar yet structurally different alternative solutions.

While the biomass values were more constant at the beginning (see figure 5.6) and at the end generations (see figure 5.8), more stratification of biomass values were observed at the middle generations (see figure 5.7). This is an interesting observation because it proves that more interesting sub optimal solutions get buried during the optimisation process because the focus is only on the optimal solution which is observed at the end. Hence storing the optimisation history and further analysis provides meaningful insights. Therefore, these sub optimal solutions are valuable in a scenario where the GEM model with the highest biomass output is infeasible to implement.

Next, the models in each generation were clustered based on the gene composition (the number of common active genes between a pair of models - see figure 5.9), common reaction composition (the number of common active genes between a pair of models - see figure 5.15a) and biomass of the models (see figure 5.13a). Even though there are 250 generations, I have chosen generation 140 as the model generation for illustration purposes because the biomass values have a good stratification which is a representative of all the biomass values observed during the 250 optimisation generations. Further, clustering was done with hierarchical clustering. The reason to opt for hierarchical clustering method for the analysis in this thesis was it was a simple clustering method and sufficient to accurately group the solutions based on the euclidean distance measures. Groupings obtained by clustering based on gene composition and reaction composition were not useful in connecting structural differences to biomass. However, clustering models on biomass revealed interesting information in identifying the common core for a given set of models (see figure 5.16).

Clustering (based on biomass) of the models in Generation 140 revealed that there were 5 distinct clusters. The number of clustered was justified by an average Silhouette value of 0.45. After analysing the gene composition of the models in cluster I was able to identify a common core for each cluster/biomass value (see figure 5.14). Common core is all the genes that are active in the models in a given cluster. Figure 5.14 represents the genes activated in each core (for each cluster) out of the 139 genes that are allowed turn on and off during the optimisation approach. Similarly each cluster has a common reaction core (common reactions available in all the modules in the respective cluster) which is presented in figure 5.16. Given the differences in behaviour it is highly unlikely that a global viable core (a model that would produce the respective biomass only when the core genes are turned on) can be achieve. Hence the local common cores work as a representative of each cluster.

The second output of the methodology is using the insights obtained from analysing the optimization archive to standardize model constraints. Through clustering of solutions, I was able to identify a common core of genes (genes that were always turned on when the model produced the respective biomass) for each biomass category. During the optimisation process 139 genes were allowed to randomly turn on and off. However, from the common core we can identify the genes that should be turned on for the whole duration of time. Since this reduced the number of genes that needs to experimented with are smaller better results could achieved easily and faster.

The third output of the methodology is temporal analysis. In temporal analysis the evolution of a particular solution is observed over the course of optimisation process. Temporal analysis was not performed for this example because there are no constant structures in the model design.

Summary

This chapter presented the second case study which was aimed at applying the optimisation-history analysing methodology defined in Chapter 3. The case study was aimed at analysing the optimisation history generated by a genetic algorithm optimising the behaviour of GEM models of metabolism of E. coli bacteria K-12 strain. The first step of the methodology was to create a mathematical model that imitates the behaviour of the complex system, bacteria metabolism. An existing mathematical definition of the system was adopted in this step. The second step was application of target driven optimisation to improve the design and compile the optimisation archive. I adopted the work done by a fellow researcher to compile the optimisation archive. Optimisation output of a genetic algorithm was used to create the optimisation archive which composed of the behaviour of the bacteria (biomass production) and structure (gene and reaction composition of models). Finally the optimisation archive was examined to identify alternative solutions. The analysis resulted in alternative solutions which were alternative by both behaviour (both behaviourally and structurally different) and structure (behaviourally similar yet structurally different).

Alternative solutions of GEM models identified in this example scenario are computational design solutions. An optimisation approach was used to observe how change in activity of genes affect the metabolic behaviour of the bacteria. However, these designs need to be constructed physically in order to verify the computationally predicted behaviour is similar to the observed physical behaviour. In a usual synthetic biology modelling process, the differences in the predicted and observed behaviour is fed-back to the design process in order to improve the computational design approach. Therefore alternative computational solutions (design) can be used to drive the engineering of alternative physical designs. In this case the alternative solution acts as an alternative design template (or blueprint).

Compared to the first case study the second case study looked at a different optimisation algorithm. In the first case study gene expression programming optimisation was applied while in the second example a genetic algorithm was used. Also the structure of data being analysed were different. Mathematical expressions in the first example case study were graphs (binary trees) where as the GEM models were viewed as liner data. The third case study which is presented in the next chapter aims at applying the methodology to identify alternative solutions in an solution space composed of DNA walker circuits constructed from the output of a simulated annealing optimisation.

Chapter 6

Example Applications 3: Alternative Solutions for DNA Circuits using Simulated Annealing Optimisation

Introduction

This chapter illustrates how the optimisation-history analysing methodology described in Chapter 3 can be employed to extract alternative solutions from a solutions space consisting of DNA Walker circuit layouts. A DNA walker circuit is a biochemical circuit built using DNA strands. The circuit layout can be viewed as an undirected graph drawn on a two dimensional Cartesian plane. Simulated annealing optimisation algorithm was used to generate the best circuit layout for a specified set of objectives. A brief introduction to DNA walker circuits, implementation of simulated annealing optimisation algorithm to generate optimal DNA walker circuit layouts that satisfies predefined objectives and a method to compare the layouts will also be discussed in this chapter.

6.1 Descriptor

The third case study which was used to test the effective usage of the optimisation-history analysing methodology is explained in this chapter. The example presented in this chapter focuses on employing a computational optimisation algorithm to obtain an optimal design layout for a DNA circuit. Methodology will be applied to analyse the optimization history generated from the optimisation program in order to identify alternative solutions (structurally different but behaviourally similar) and improve the design process using the insights from the history.

Unlike the first example, analysing the optimisation history of improving mathematical expressions structure, DNA circuit layout provides a more realistic solution space composed of varying degrees of alternativeness attributing to different combinations of structural and behavioural similarities.

The three main phases of the methodology are;

- Step 1: Mathematical model definition

  The first phase of the methodology is to define a mathematical model for the problem. A DNA walker circuit is a biochemical circuit built using DNA strands. DNA walker circuits can be used for a number of applications including precise arrangement of matter at the nano-scale and the creation of smart bio-sensors. Similar to silicon circuits, DSD systems are able to evaluate non-trivial mathematical functions. Further, experiments have shown that DSD systems can simulate logic circuits (Dannenberg et al. 2013). These capabilities of DNA walker circuits pave the means of DNA being used as a digital data storage in the future. DNA walker circuit technology is still in it's early experimental stages and circuit layout approach being looked at in this thesis is a fairly new approach proposed by Professor Gilbert and Professor Heiner. Section 6.2 provides a detailed description on the DNA circuit layout (model definition) and the examples which will be looked at in the case study. The main focus of the study is to obtain an optimal design for a given layout and a set of alternative designs.

- Step 2: Target driven optimisation

  In the second step of the methodology, computational optimisation is applied on the layouts to select the best solution. Once this step is completed the optimisation archive will be created which will be analysed to identify possible alternative solutions.

  As the DNA circuit layout is novel, this would be the first instance computational optimisation techniques has been applied to obtain the optimal design. Hence, mainly two optimisation algorithms; hill climbing and simulated annealing were used to identify the most effective algorithm. Sections 6.3, 6.4 and 6.5 includes a detailed description of formulating layout optimisation problem for DNA walker circuits and application of the algorithms.

- Step 3: Analysis of the optimisation archive

  The third step of the methodology is to analyse the optimisation archive in order to identify the alternative solutions. Section 6.6 has a detailed explanation of the insights extracted from analysing the optimisation history. During the analysis mainly three insights were obtained which could aid in improving the design process of DNA walker circuits.

1. Obtained alternative DNA walker design layouts which are behaviourally similar yet structurally different
2. Identified optimal core design elements in layouts by observing the evolution of solutions over time. Core designs are an arrangement of nodes (e.g. Init, Fork, Norm 1 and Norm 2 nodes in Toy layout) that remains constant in each optimal design. These core designs can be used as fixed blocks when designing larger circuits. Further having fixed location for nodes could reduce the number of locations which needs to varied in the optimisation algorithm decreasing the execution time.
3. Finalized design constraints (relationship between number of nodes and maximum length and width of the layout) for DNA walker circuit layouts

6.2 Step 1 - A Brief Introduction to DNA Circuits

6.2.1 Nanotechnology and DNA

The prefix *"nano-"* is a familiar term used in the metric system that refers to a scale of size. *"nano"* is derived from the Greek work *nannos* meaning the dwarf. In the metric system nano is used used to denote a factor of 10^{-9} (0.000000001 or one billionth). Therefore nanotechnology can be defined as "the understanding and control of matter at dimensions of roughly 1 to 100nm, where unique phenomena enable novel applications" (Allhoff, Lin, and Moore 2010). At present nanotechnology research has become highly and increasingly integrative across multidisciplinary knowledge sources. For example medicine (surgery, immunology) , engineering (telecommunication, electrical and electronic), chemistry (drug discovery) and physics (particles, nuclear) are few major disciplines where nanotechnology is applied to solve problems (Porter and Youtie 2009).

DeoxyriboNucleic Acid (DNA) is present in all living cells and it is the carrier of genetic information. DNA engineering is a well known research discipline where level of gene expression is modified in order to optimize the nature of gene products and improve cellular performance. However nano-technological purpose of DNA engineering specifically aims at creating specific topologies, shapes and arrangements of secondary and tertiary structure to manipulate the spatial and temporal distribution of matter (**seeman1999dna**).

6.2.2 DNA Walker Circuit

A DNA walker circuit is a biochemical circuit built using DNA strands. A DNA walker circuit has mainly two parts; a DNA Strand Displacement (DSD) circuit and a DNA walker. A DSD circuit is created by tethering of DNA strands to a rigid lattice (origami tile). These tethered strands are called anchorages. A DNA walker is a free-moving single DNA strand which is capable of traversing along the lattice performing a computation.

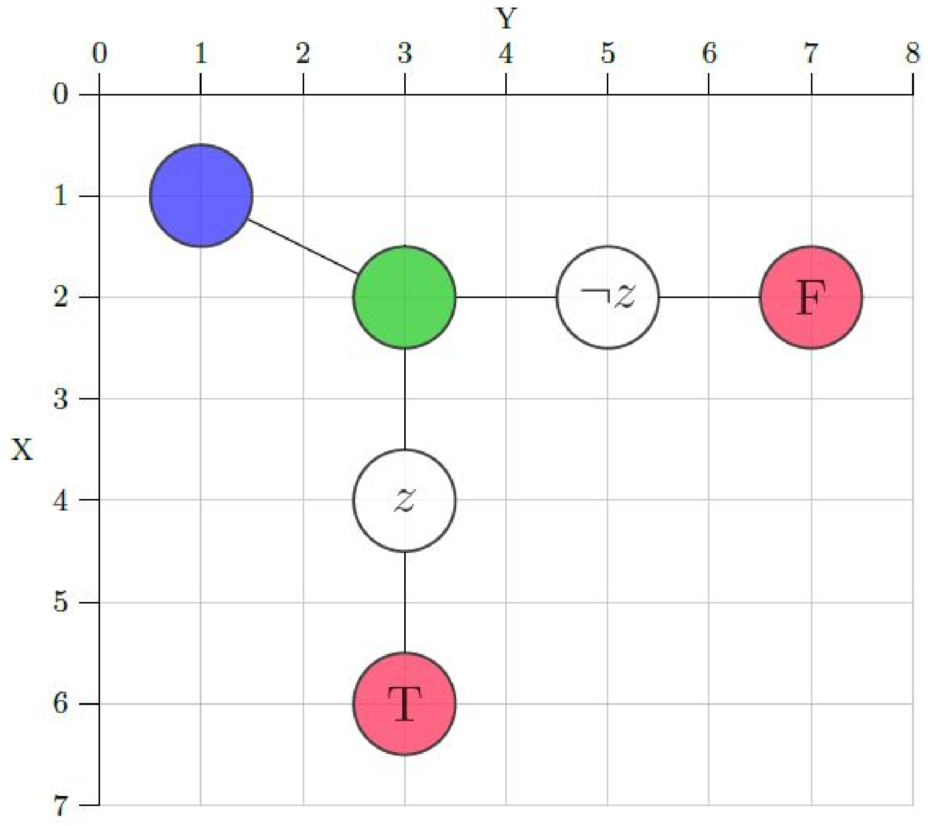


Figure 6.1: An illustration of the "Toy" circuit with the use of coloured Petri nets. Colour code: blue – INIT, green – FORK, red – FINAL; uncoloured – NORM

Figure 6.1 represents how a DNA circuit looks like to a modeler. However, in the real life this DNA circuit looks like any other DNA strand. DNA strands which are expected to react (a strand with DNA circuits and DNA walkers) are first placed in a mixed chemical solution. The reactions between the walker and the circuit are controlled by enzymes.

While there can be DSD circuits with free moving DNA strands, tethering of DNA strands (nodes in the circuit in figure Figure 6.1) give the ability to impose constraints on the reaction between DNA strands by preventing the otherwise free moving stands to react in unintended ways (Qian and Winfree 2011). The manner a walker traverse along the circuit is called DNA strand displacement. Figure **??** is an illustration of the DNA displacement process. A single free moving DNA strand (walker: green colour strand) first binds to a partially double-stranded (black and red colour strands), then releases (or displacing) the originally bound strand (red colour strand). This process is called DSD and Qian and Winfree have further discussed about several instances of how DSDs are used experimentally to simulate logic gates (Qian and Winfree 2011). A simulation of the binding of a walker to the DNA circuit can be found in `https://youtu.be/42FCzoJt8Pg` as well.

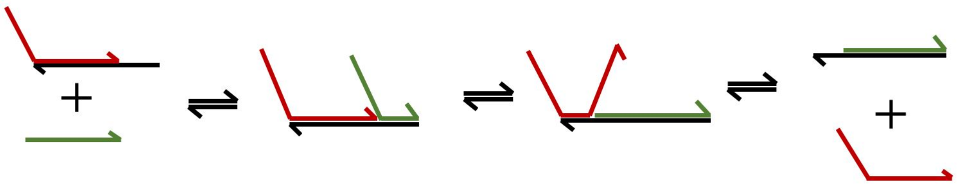


Figure 6.2: An illustration DNA strand displacement: A single free moving DNA strand (walker: green colour strand) first binds to a partially double-stranded (black and red colour strands), then releases (or displacing) the originally bound strand (red colour strand). (This illustration was inspired by the work published in "Control of DNA Strand Displacement Kinetics Using Toehold Exchange" (Zhang and Winfree 2009))

DNA walker circuits have been further studied by Dannenberg and his research group and they have introduced a modelling approach for walker circuits based on experimental data (Dannenberg 2016). However, in this thesis I will be looking only at the DNA walker circuit layout.

6.2.3 DNA Walker Circuit Layout

The DNA walker circuit layouts which are discussed in this thesis have been adopted from the work published by Prof Gilbert and Prof Heiner (Gilbert, Heiner, and Rohr 2018). There are four circuit layouts namely *Toy*, *Toy0*, *Toy1* and *Toy2*. These circuits are designed to evaluate a Boolean function over n input variables. Below figure 6.3 is an illustration of the layout of the circuit named *Toy* which is elaborated in Prof Gilbert and Prof Heiner's work (Gilbert, Heiner, and Rohr 2018).

As shown in figure 6.3 a DNA walker circuit can be viewed as an undirected graph. The graph can be considered as a binary tree where the walker evaluates a Boolean function while it walks along the branches of the tree. If the walker reaches the red node in the horizontal branch, the result will be *false* and if the walker reaches the same coloured node in the vertical branch the

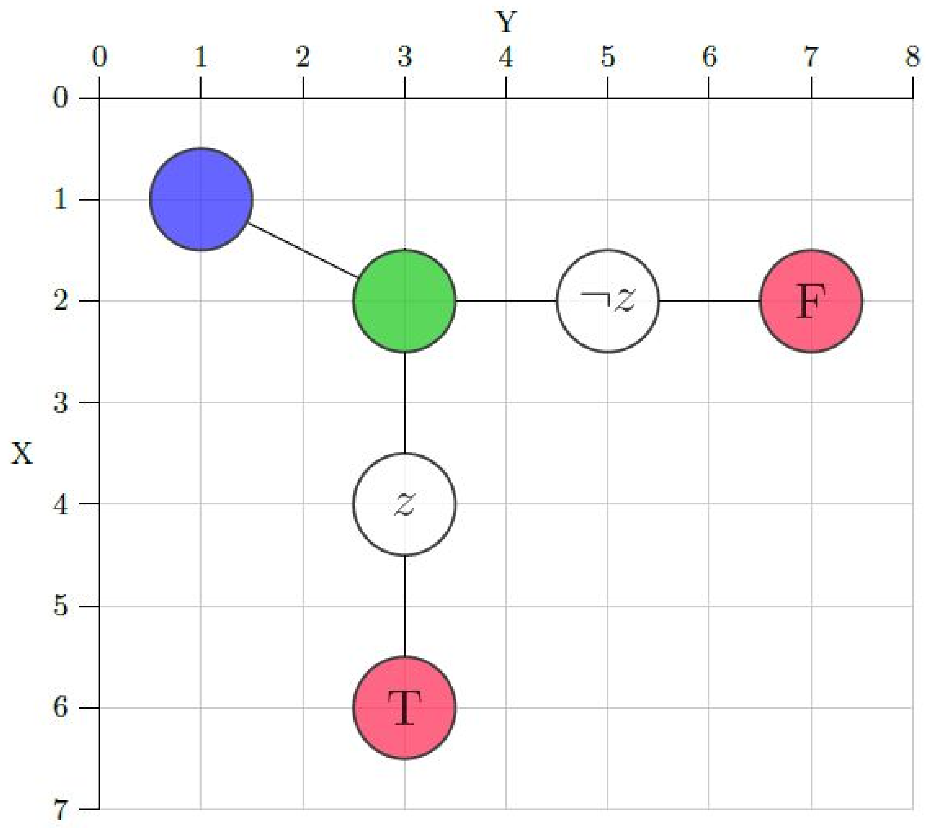


Figure 6.3: An illustration of the *Toy* circuit layout with the use of coloured Petri nets. Colour code: blue – INIT, green – FORK, red – FINAL; uncoloured – NORM

result will be evaluated to *true*. Vertices are referred to as anchorages (terms vertices, nodes and anchorages are used interchangeably in this text) and undirected edges represent possible walker steps. There are five different types of vertices based on their functionality. In the diagram the nodes have been colour coded according to their functionality. These vertices are laid out in a Cartesian grid of a fixed size which is determined by the size and shape of the tree.

6.2.4 Different Types of Anchorages and Their Behaviour in a DNA Circuit

There are mainly five types of vertices. Each of these types have their own characteristics and behaviour which will be explained in detail in the layout constraints section.

- *INIT* - Init (also refereed to as I) is the unique initial node (anchorage). This node denotes the entry point into the circuit for a walker. There is only one Init node in a circuit. When representing the layout as a coloured Petri net Init node will be encoded in blue.
- *FORK* - Fork (also refereed to as F) node denotes a junction where the circuit branches

into two sections; true or false. There can be one or more Fork nodes in a circuits. When representing the layout as a coloured Petri net Fork node will be encoded in green.

- *NORM* - Norm (or N) node processes a boolean expression. In a binary decision tree there are two types of norm nodes; True Norm vertex (also refereed to as $N1n$ where n represent the order of Norm nodes) and False Norm vertex (also refereed to as $N2n$). There can be more than one Norm node in a circuit. When representing the layout as a coloured Petri net Norm node will be encoded as an uncoloured node.
- *FINAL* - Final node denotes the position where the walker leaves the circuit. In a binary decision tree there can be only two Final nodes; True vertex (also refereed to as $F1$) and False vertex (also refereed to as $F2$). When representing the layout as a coloured Petri net Final node will be encoded in red.
- *JOIN* - Similar to a Fork node a Join node denotes a junction. Binary decision trees do not require join anchorages therefore none of the examples discussed in this thesis have join nodes.

6.2.5 Connectivity Between Anchorages in a DNA Circuit Layout

Each anchorage (node) has a designated set of connections. Both *INIT* and *FINAL* nodes have only one adjacent edge, *NORM* node will have two adjacent edges and *JOIN* and *FORK* nodes have three adjacent edges.

Thus the edge connections can be summarised as;

- *INIT* node has one adjacent edge that leaves the nodes
- *FORK* node has three adjacent edges; one enters the node and two leaves the node
- *NORM* node has two adjacent edges; one enters the node and one leaves the node
- *FINAL* node has one adjacent edge that enters the nodes
- *JOIN* node has three adjacent edges; one enters the node and two leaves the node or two enters the node and one leaves the node

For a given set of input values, a walker is supposed to go only to anchorages, where the evaluation of the label yields true. However as the walker walks along an undirected graph ideally it can start its journey at a *INIT* and traverse to any node or start at one of the *FORK* nodes. Therefore the traversal imposes several constraints on a walker's movement. A DNA walker can thus travel in one direction starting from *INIT* vertex to one of the *FINAL* vertices and it is not allowed to visit a node that has already been visited.

6.2.6 Simulation of DNA Walker Circuits

DNA walker circuits are tested in-silico. Prof Heiner and her research group has developed a Petri net analysis too named Marcie (Heiner, Rohr, and Schwarick 2013) and it is used to simulate a DNA walker circuit.

In order to simulate the circuit a CANDL specification of the layout needs to be created. A CANDL specification is a text file with the extension *.candl* and it includes the structure of the layout; a representation of the circuit in Cartesian form (x and y coordinates for every node in the layout), and characteristics of the layout such as grid size, node connectivity and distance measurements *(a template of a sample CANDL file is in Appendix 7.3.4)*. Next the CANDL specification is processed by Marcie using a specific set of commands. Marcie is used to perform a qualitative analysis on the circuit.

Marcie is able to perform the following tasks :

- Simulate the functionality of the DNA walker circuit
- Determine the type and number of leak transitions in the circuit
- Determine whether any dead states exists in the circuit

6.2.7 Research Question

When a walker is moving along a localized (tethered) DSD there is a possibility of the walker jumping to another branch of the decision graph. This undesirable movement results in an incorrect output called a leakage transition. Prof Gilbert and Prof Heiner have proposed a method to automatically identify leakage transitions, which allows for a detailed qualitative and quantitative assessment of circuit designs, design comparison, and design optimisation. As Prof Gilbert and Prof Heiner suggest the leakage in a walker circuit can be reduced by optimising the distance between two anchorages (nodes). Ideally this can be achieved by increasing the circuit area. However at the same time the circuit area should be minimised too.

As highlighted in Chapter 1, the main research question I intend to answer through the research is "When using computation optimisation to improve system design, how can alternative solutions be efficiently found by analysing the history of the optimisation approach?".

The goal of the work presented in this chapter is to explore different ways of laying out the nodes in a given circuit preserving the topology so the leakage transitions and circuit area are minimised (Gilbert, Heiner, and Rohr 2018). Following are the questions I intend to answer through my research:

- What is the optimal way to layout the circuit components preserving the node and arc connections that would minimize both the area occupied and the number of leaks?
- Can the components be laid in a different manner while the behaviour remains the same? (i.e. are there alternative solutions; different structure but similar behaviour, for a given design)
- Is there an inherent ranking present within the solutions?

Computational optimisation algorithms will be used to obtain the optimal solution. The methodology define in Chapter 3 will be used to analyse the optimisation history to locate alternative solutions and rankings.

6.2.8 Four DNA Walker Circuits Examples

In order to explore the different ways of laying out a circuit, four different types of circuits have been used. These layouts were designed by Prof Gilbert and they are inspired by the work published in (Gilbert, Heiner, and Rohr 2018). These layouts are referred to as *Toy layouts.* Table 6.1 contains details of the four layouts.

6.3 Step 2 - Formulation of the optimisation problem

6.3.1 Layout Optimisation Problem

As identified by Prof Gilbert and Prof Monika in their research (Gilbert, Heiner, and Rohr 2018), main challenges involved in designing a reliable walker circuit consisted of minimizing leakage transitions and area.

Leakage transition is the computation error which occurs when a walker jumps into another branch of the decision graph. Area is the area occupied by the layout. In general we can minimise the leakage transitions by increasing the distance between nodes. However it is also

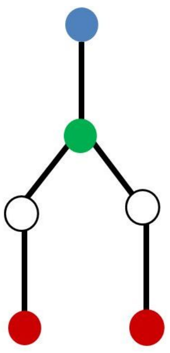
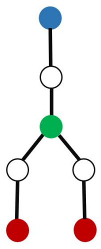
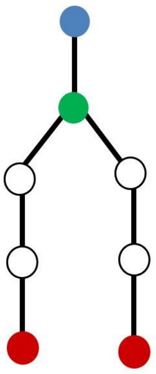
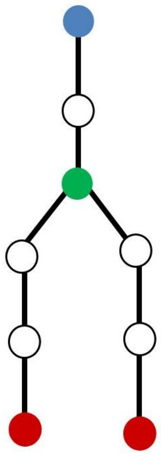
Feature	DNA layout			
	Toy	Toy 0	Toy 1	Toy 2
Layout				
No of nodes	6	7	8	9
Grid size	14 x 14	14 x 14	20 x 20	20 x 20
No of Init nodes	1	1	1	1
No of Fork nodes	1	1	1	1
No of Norm nodes	2	3	4	5
No of Final nodes	2	2	2	2

Table 6.1: Details of the four example DNA walker circuits

a requirement of an ideal circuit to have the least area possible. Hence the aim is to minimise leakage transitions while minimising the area of the circuit.

For example consider the Toy circuit model illustrated in 6.3. There are several hundred ways in which the nodes can be laid out preserving the circuit layout constraints. Therefore it is not possible to work it out by hand or mathematically compute the best arrangement of nodes which occupies the least area and have the least number of leaks. This suggests that the exploration for the best solution would be benefited by adopting a computational search mechanism. The search for the optimal circuit was done in two separate computational methods; exhaustive search and computational optimisation which are described in detailed in the following sections.

6.3.2 Exhaustive search by Prof Gilbert's Prolog program

Prof Gilbert implemented a Prolog program which exhaustively searched the complete solution space for the optimise solution. The program was developed in Prolog and it was able to generate all the possible solutions for a given model. After computing the fitness for all the solutions, the solution that had the least fitness value was selected as the best one. Prof Gilbert was able to exhaustively search the solutions spaces for Toy, Toy0, Toy1 and Toy2 models. Following table 6.2 provides a summary of the results obtained from Prof Gilbert's search.

Model layout	No of solutions	No of unique fitness values	Minimum Fitness	Maximum Fitness
Toy	263,920	140	40	11816
Toy0	786,943	225	60	17091
Toy1	394,002	817	780	35520
Toy2	640,609	1113	780	43512

Table 6.2: Number of solutions obtained from Prof Gilbert's exhaustive search for the four Toy models

For instance let's look at the solution space of Toy layout which was generated through exhaustive search. The 263,920 total number of solutions could be fitted into 140 unique fitness values. The minimum fitness value observed was 40 and the maximum fitness value was 11816. Distribution of fitness values of the total solution space is depicted in Figure 6.4. Further figure 6.5 is a representation of the distribution of leaks and area of the solutions. Similarly the complete solution space for Toy0 circuit layout consisted of 225 unique fitness values where the fitness values varied between a minimum fitness of 60 and a maximum fitness of 17091.

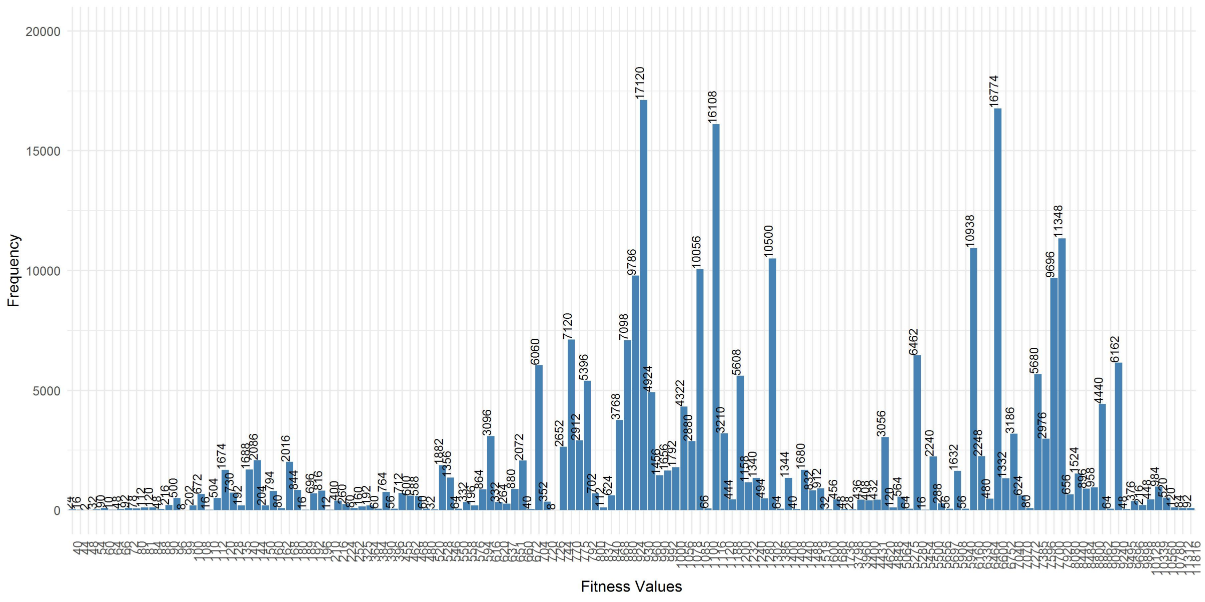


Figure 6.4: Distribution of fitness values in the total solution space generated for Toy layout through exhaustive search

While it is possible to identify the optimal circuit layout designs using Prof Gilbert's method, there are couple of drawbacks in his approach. The Prolog program has to first generate the total solution space and then evaluate each and every solution. This exhaustive search uses a lot of resources. As we are interested in an optimal solution it is a waste of resources to look

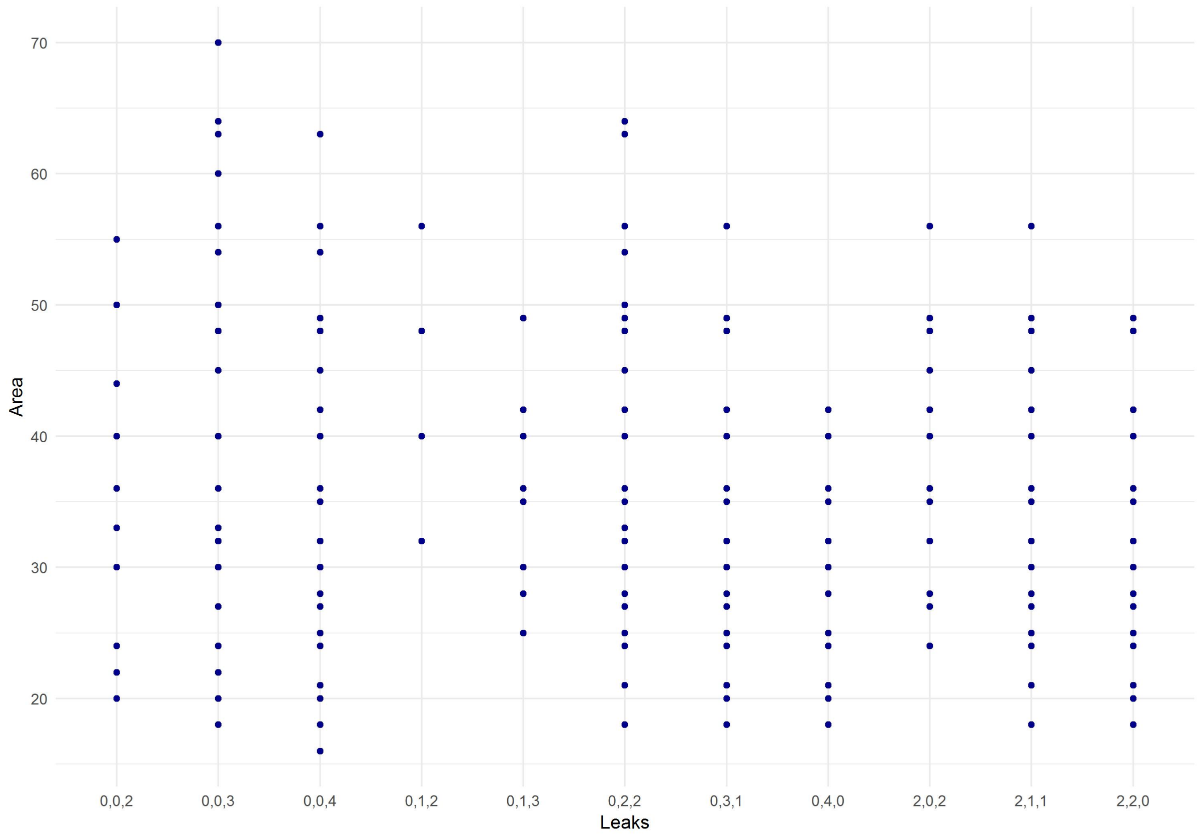


Figure 6.5: Distribution of area and leaks in the total solution space generated for Toy layout through exhaustive search

at all possible answers (for the Toy layout the first 10% of the best fitness values account to only 1388 out of the total number of solutions 263,920). For example, for the Toy circuit model even though there are 263,920 solutions in total, only 24 solutions have the minimum fitness. Further for larger circuit models it is impossible to perform an exhaustive search. For example the program crashed few times when trying to generate the solution space for Toy 1 and Toy 2 layouts given the difference is an addition of one node to each layout compared to its predecessor. Therefore an exhaustive search is an ineffective method. However the results of the exhaustive search was beneficial in implementing and validating the functionality of the heuristic search.

6.3.3 Computational optimisation of the layout problem

As an exhaustive search proved to be inefficient and impractical, especially as the layouts gets larger, a heuristic method was considered more suitable. A heuristic search provided the capability to explore the solution space in an efficient manner resulting in a good enough, if not the best, solution. Hence computational optimisation was used to explore the solution space.

As explained previously a DNA walker circuit can be viewed as an undirected graph. Graph layout problem is a well-researched area and are a separate category of combinatorial optimization problems whose goal is to find a linear layout of an input graph in such way that a certain objective cost is optimized (Díaz, Petit, and Serna 2002). The graph problem is mainly applied to three main categories of examples. The first example category is known as the facility layout problem (a.k.a plant/ machine layout problem). The optimisation process is focused on the placement of facilities (e.g., machines, departments) in a plant area, with the aim of achieving the most effective arrangement in accordance with some criteria or objectives under certain constraints, such as shape, size, orientation, and pick-up/drop-off point of the facilities (Hosseini-Nasab et al. 2018). A majority of the work on graph layout problem is based on facility layout problem using different optimisation techniques such as swarm optimisation (Rossetti, Macor, and Scamperle 2017), simulated annealing (Tayal and Singh 2018) and genetic algorithms (Raman, Nagalingam, and Gurd 2009).

The second example category can be considered as developing optimum design of truss type structures (a.k.a topology design) with respect to size, shape and topology design variables using optimisation (Tort, Şahin, and Hasançebi 2017), (Hasançebi and Erbatur 2002). The third category of example data used in graph layout optimisation is circuit layouts. These circuits can be integrated circuit designs (Berkens et al. 2012), (Hughes, Morton, and Monk 2007) or DNA walker circuits (Kawamata, Tanaka, and Hagiya 2009).

Optimisation of a DNA walker circuit discussed in this thesis relates to a method of optimizing an integrated circuit layout, wherein an initial DNA walker circuit layout is provided. A predetermined set of constraints (physical characteristics) are used to evaluate the structure of the circuit and a cost function is used to evaluate the goodness of a layout and layout are selected that optimize the goodness value, so that the circuit layout is optimized. Mainly two optimisation algorithms were used to in the heuristic search; random restart hill climbing and simulated annealing.

6.3.4 Defining a Fitness Measure for a DNA Walker Circuit

A "heuristic search refers to a search strategy that attempts to optimize a problem by iteratively improving the solution based on a given heuristic function or a cost measure" (Lu and Zhang 2013). Hence it was essential to define a cost function for a DNA walker circuit. The main reason for defining a custom cost function for the heuristic search described in this thesis is that there doesn't exist a standardised cost function to measure the goodness of DNA walker circuits based on the parameters that are being optimised.

As mentioned previously the aim of the heuristic search is to reduce the leaks of a DNA walker circuit and area occupied at the same time. Thus the fitness measure is defined as follows:

$$Fitness = (\text{short leaks} * 100 + \text{medium leaks} * 10 + \text{long leaks} * 1) * \text{area} \tag{6.1}$$

$$\begin{aligned}
\text{short leaks} &= \text{number of short leaks in the circuit} \\
\text{medium leaks} &= \text{number of medium leaks in the circuit} \\
\text{long leaks} &= \text{number of long leaks in the circuit} \\
\text{area} &= \text{area of the circuit}
\end{aligned} \tag{6.2}$$

Hence as per the above mentioned fitness measure a lower fitness value is considered as a better layout. In the fitness measure the leakage transitions are weighted by type as there is an order of significance for each type. Short leaks are more common and more prone to erroneous output and hence those leaks should be minimised more than the other two types. Medium and long leaks are rare and more tolerable than short leaks hence have assigned a lower weighting.

The area of the circuit is the rectangular area occupied by nodes in the Cartesian plane. Hence it is calculated as follows:

$$\text{Area of the circuit} = (Xmax - Xmin) * (Ymax - Ymin) \tag{6.3}$$

6.4 Optimising DNA Walker Circuit Layout Problem using Hill Climbing Algorithm

6.4.1 Hill Climbing Algorithm

Hill climbing algorithm is a local heuristic search technique used to optimise mathematical problems. As the name suggests the functionality of the algorithm depicts climbing to the highest point of a range of hills. If the solution space is represented as a range of hills, the algorithm will climb the hill until it reaches the highest hill top. The mechanism used to identify the highest top is to compare the current height with the previous best height.

The algorithm starting with an arbitrary solution for a problem, then attempts to find a better solution by making an incremental change to the solution. Subsequently a hill climber performs an iterative search by continuously moving in the direction of increasing fitness value to find the best solution to the problem. The algorithm terminates after a set number of iterations (Selman and Gomes 2006).

Following figure 6.6 is a representation of different states a hill climbing algorithm could reached during its search process and the values of the fitness function.

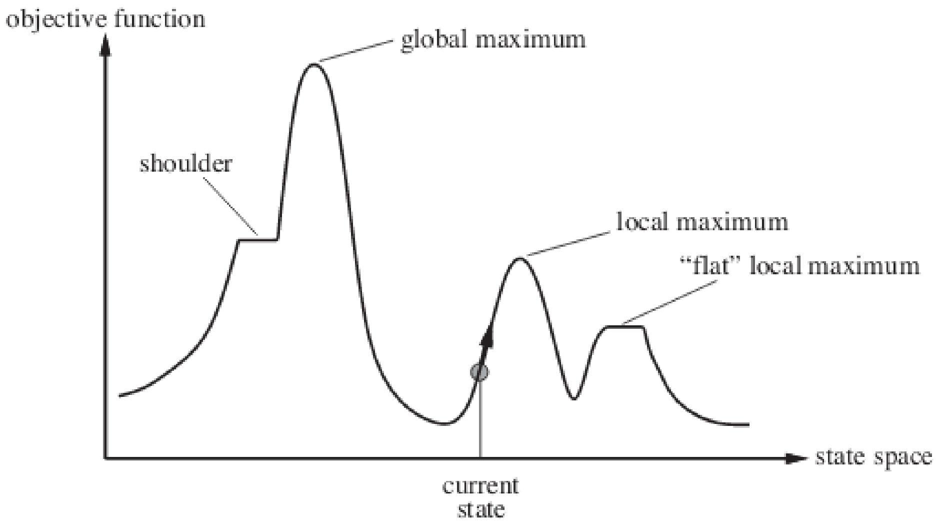


Figure 6.6: State space diagram for hill climbing algorithm: x axis represents the and the y axis represents the value of the objective function/fitness (`source:https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/`)

The point which is marked as the *current state* is the place where the algorithm is present at this moment. There are three main types of regions the algorithm can reach during its search process.

- Local maximum: a local maximum is the best state the algorithm can reach compared to the neighbours. However it is known as a local maximum as there is a state better than the local maximum (global maximum).
- Global maximum: the best possible state in the state space graph.
- Plateua/flat local maximum: a flat region where the objective function has the same value. It is impossible for a hill climbing algorithm to get out of a plateua as the direction of travel can not be decided with exactly similar values.

Hill climbing uses a greedy search approach; meaning at any point in the solution space, the search moves in the direction only which optimizes the cost function with the hope of finding the optimal solution at the end. Hence there is a possibility that the optimisation algorithm will stop at a local optimum as shown in the above figure. This is the basic hill climbing algorithm. There are several versions of the hill climbing algorithm with slightly different search mechanisms to overcome some of the drawbacks in the basic hill climbing algorithm.

- Simple hill climbing algorithm: This is the simplest hill climbing algorithm. The algorithm will evaluate the neighbour state and select the first state which optimises current cost and set it as a current state. Only one state will be checked at a time and if a better state than the current state is found, then the algorithm will move else it will remain in the same state. Following is the pseudo code for the simple hill climbing algorithm.

Algorithm 1 Simple Hill Climbing Algorithm

Input: A random solution
Output: Solution with the best fitness

```
 1: procedure SimpleHillClimber
 2:     $S \leftarrow$ Random Solution
 3:     $F \leftarrow$ Fitness($S$)
 4:     for $i \leftarrow 1$ to ITER do
 5:         $S' \leftarrow$ Small Change($S$)
 6:         $F' \leftarrow$ Fitness($S'$)
 7:         if $F'$ better $F$ then
 8:             $S \leftarrow S'$
 9:         end if
10:     end for
11:     return $S$
12: end procedure
```

- Random restart hill climbing algorithm: As mentioned earlier the algorithm will check one state at a time and if a state better than the current state is found, then the algorithm will move else it will remain in the same state. One drawback of simple hill climbing algorithm is that the algorithm can get on trapped at a local maximum or a plateua (flat local maximum) due to its traversal strategy. As a solution the initial search state can be reset to a randomly selected location and the search operation can be restarted giving the possibility of eventually reaching the global maximum.

- Steepest-Ascent hill climbing algorithm: The steepest-ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighbouring states of the current state and selects one state which maximizes the objective function. This algorithm consumes more time as it searches for multiple states (Arriaga and Valenzuela-Rendón 2012).

6.4.2 Implementation of Random Restart Hill Climbing Algorithm to Optimise DNA walker circuit layout

The aim of designing the optimal layout for a given DNA walker circuit was to come up with a layout(s) with the least number of leaks and area. As described in the previous section the main challenges faced in designing the optimal layout were the exponential number of permutations in topology and computational overhead which increased with the number of nodes in the circuit. Therefore a heuristic search was deemed appropriate for an efficient design process.

Random restart hill climbing algorithm was selected as a suitable optimisation technique due to few reasons. Firstly it was possible to transform the circuit layout problem into a mathematical programming problem as the number of leaks and area were quantitative parameters. Hence it was easy to define a fitness function required for the optimisation. The second reason for opting for hill climbing algorithm was its easiness in implementation. Using computational optimisation in a design process targeted to minimise the leaks and area in a circuit was a novel idea. Hence it was necessary to confirm whether the expected output (results from the exhaustive search) could be obtained from a heuristic search. Therefore the efficiency of implementation was the second reason for selecting hill climbing algorithm. Further the functionality of a genetic algorithm was not appropriate for the layout problem. In a genetic algorithm the mutation function combines portions of data (DNA) from two solutions and create a new solution. The idea of plugging two halves of separate layouts together to make a new solution did not seem reliable as there were many constraints involved in layout the components in a circuit.

Java was selected as the preferred programming language for the implementation as I had prior knowledge in coding with Java. Because the learning curve was minimal implementation of the optimisation program could be done fairly quickly. Implementation of he optimisation program consisted of two main parts; implementation of the random restart hill climbing algorithm and implementation of the DNA walker circuit layout constraints in order to generate and validate layouts.

Implementation of the random restart hill climbing algorithm

Implementation of the hill climbing algorithm was fairly straightforward. Below is the pseudo code (algorithm 2) for the random restart algorithm used for the optimisation. While the inner for loop is responsible for a single hill climbing search, the outer for loop is responsible for the restart function of another hill climber search at a random point in the search space. The variable *ITER* denotes the number of iterations the hill climbing search process was iterated within a single execution (number of states it visited in the solution space). Ideally the value of *ITER* depended on the size of the circuit and the number of iterations increased with the size of the circuit. *STARTPOINTS* variable was used to specify the number of times the hill climber was restarted. The preferred number of *STARTPOINTS* was set to 20. The more times the hill climber was restarted a wide range of local maximums could be observed however the value of *STARTPOINTS* was constrained by the execution time of one hill climbing search process. The execution time for one hill climbing process increased with the size of the circuit as the time taken for layout validation increased with the number of nodes in the circuit along with the number of iterations per process.

Algorithm 2 Random Restart Hill Climbing Algorithm

Input: A random solution
Output: Solution with the best fitness

```
procedure RandomRestartHillClimber
    for $x \leftarrow 1$ to STARTPOINTS do
        $S \leftarrow$ Random Solution
        $F \leftarrow$ Fitness($S$)
        for $i \leftarrow 1$ to ITER do
            $S' \leftarrow$ Small Change($S$)
            $F' \leftarrow$ Fitness($S'$)
            if $F'$ better $F$ then
                $S \leftarrow S'$
            end if
        end for
        return $S$
    end for
end procedure
```

Implementation of the DNA walker circuit layout constraints in order to generate and validate layouts

The main part of the programming effort was required to encode DNA walker circuit constraints into Java. The work was mainly on three functions.

1. Function: *GenerateRandomSolution* - This function was used to generate a random solution. To form a valid solution the nodes were placed on the Cartesian plane as per the constraint definition. Size of the Cartesian plane, number of nodes, order of appearance and type of each node, and the connections between each pair of nodes were given as input to the program.

   The algorithm starts from the first node (i.e. INIT); it randomly generates a pair of coordinates for the INIT node, and moves on to the next node according to the order specified. Next the algorithm generates all the possible locations the next node could be placed and picks a location randomly from the available nodes. If there are no available places for the node to be placed algorithm will start at with new initial position.

   This approach was an efficient way of generating layouts than randomly generating nodes and validating against constraints afterwards.

2. Function: *SmallChange* - In a hill climbing algorithm a small change needs to be made to the existing solution in order to decide the direction of travel. This function was used to introduce a small change into an existing solution (S). As per the algorithm 50% of the nodes in the layout were allowed to be moved to different places. Firstly the Java algorithm generates a random integer number between 1 and $n/2$ (n being the number of nodes in the circuit) to decide how many nodes will be changed. Next it will generate all the possible locations the selected node can move and randomly pick one from the available locations. If there are no available locations it will discard the changes and start fresh.

   Usually the small change function in a hill climbing algorithm is expected to perform the smallest possible change. In the circuit layout problem the smallest change is changing the location of one node. However changing one node at a time did not introduce much variability into the solution as the number of possible locations a node could be placed was minimal due to layout constraints. As a result there were a significant number of repeats in the hill climbing solution space and in all the 20 runs (each run with 5000 iterations) not only it could not reach the lowest fitness observed during the exhaustive search but also the variability in fitness values observed were minimal. Therefore multiple locations were allowed to change with one change operation causing the hill climber to make a significantly high jump from a lower fitness to a high fitness value. This proved to be an effective method because for the Toy model in the next 20 runs (with the improved small change function) 45% of the time the hill climber was able to reach a fitness value within the top 10% of the lowest values observed during exhaustive search.

3. Function: *LeakComputation* - This function was used to compute the leaks in a circuit. As described earlier the leaks in a circuit is necessary to calculate the fitness of that layout. Initially the leaks were computed using the Marcie simulator. However calling Marcie was computationally expensive. Due to the computational overhead it took nearly 2 hours for one run (with 5000 iterations) to complete. Therefore the leak computation algorithm was implemented in Java and it reduced the execution time of one run (with 5000 iteration) to 2 minutes.

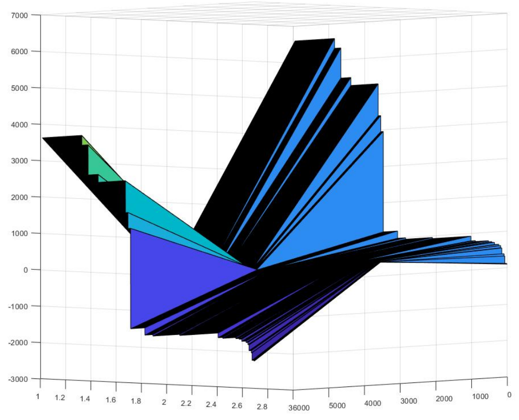
The next section is an analysis of the solutions obtained from the hill climbing algorithm.

6.4.3 Evaluation of results from Hill Climbing Algorithm

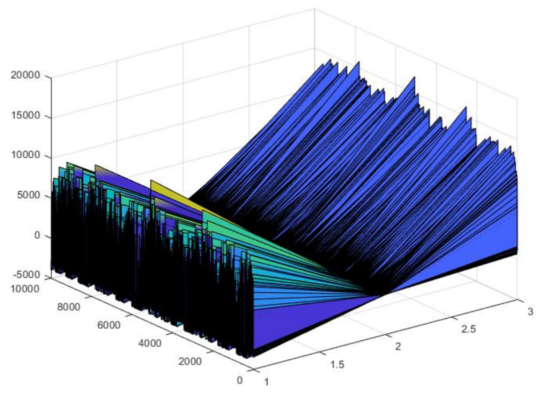
The main reasons an optimisation approach was used in the generation of possible layouts were due to the limitations and inefficiency in an exhaustive search. As described in the previous section a random mutation hill climbing algorithm was used to search the solution space for each of the four layouts. The results of the search and conclusions made are discussed in this section.

Initially the hill climbing algorithm was run on both Toy and Toy0 circuit layouts. For each run the hill climbing optimisation was repeated for 20 runs while each run having a set $STARTPOINT$ value of 5000 iterations. However the minimum fitness value observed during all the runs did not reach a value below 500 for both circuits in all the runs. As mentioned earlier the minimum fitness observed during the exhaustive search for Toy and Toy0 layouts were 40 and 60 respectively.

As the hill climbing algorithm could not reach the expected minimum fitness, the complete solution space for both Toy and Toy0 layouts were examined. A surface plot of the solution space was generated by combining all the variables in a layout (i.e. coordinates, area, fitness and leak information) and reducing it to two dimensions applying dimensionality reduction. This was done using MatLab software. Figure 6.7 is a representation of the solution space for both Toy



(a) Toy



(b) Toy0

Figure 6.7: Surface plots representing the total solution space of Toy and Toy0 layouts. The total solution space was obtained from the exhaustive search Proglog program.

and Toy0 layouts.

As depicted in the surface plot in figure 6.7 the distribution of solutions took the form of a valley instead of a hill. The main disadvantage of this form of a distribution is that the hill climbing algorithm will always stop in a local optimum instead of reaching a global optimum. Hence it was concluded that a hill climbing algorithm was not suitable for this problem. Simulated annealing optimisation technique was considered appropriate for the optimisation of layout designs as it has the capability of accepting slightly worse solutions in order to escape from being trapped in a local optimum.

The next section will discuss in detail on how simulated annealing optimisation technique was used to optimise the design process of DNA walker circuits.

6.5 DNA Walker Circuit Layout Optimisation using Simulated Annealing

6.5.1 Simulated Annealing Algorithm

Simulated annealing is a global optimisation technique which mimics the physical annealing process of a crystal structure (Kirkpatrick, Gelatt, and Vecchi 1983). Simulated annealing is capable of working with both discrete and continuous fitness values. The simulated annealing optimisation techniques is widely applied to solve problems in different disciplines such as the travelling salesman problem (Allwright and Carpenter 1989), designing electronic circuits (Wong, Leong, and Liu 2012), dynamically planning the traversal path for robots (Miao and Tian 2008) and in bioinformatics designing of protein molecules (Goodsell and Olson 1990).

A simple heuristic such as hill climbing, always traverse the solution space by searching for a better neighbour after better neighbour and stops when there are no neighbours that are better solutions than the current solution. This approach does not guarantee to lead to the best solution most of the time as the current best could be a local optima while the actual best solution would be a global optimum that could be different. The speciality of simulated annealing optimisation technique is that although the search mechanism prefer better neighbours, it is capable of accepting worse neighbours in order to avoid getting trapped in a local optima. Ideally a simulated annealing algorithm can find the global optimum if it is run for a long enough amount of time.

The following pseudocode represents the simulated annealing optimisation approach in it's generic form.

Algorithm 3 Simulated Annealing Algorithm

Input: A random solution S, starting temperature T
Output: Solution with the best fitness

```
1: procedure SimulatedAnnealing
2:     $S \leftarrow$ Random Solution
3:     $F \leftarrow$ Fitness($S$)
4:     for $x \leftarrow 1$ to ITER do
5:         $S' \leftarrow$ Small Change($S$)
6:         $F' \leftarrow$ Fitness($S'$)
7:         if $F'$ better $F$ then
8:             $S \leftarrow S'$
9:         else if AcceptanceProbability better Random$(0, 1)$ then
10:            $S \leftarrow S'$
11:        end if
12:        $T \leftarrow$ NewTemperature
13:    end for
14:    return $S$
15: end procedure
```

The annealing schedule and the acceptance probability plays a significant role in the simulated annealing search technique.

Annealing Schedule

Simulated annealing is inspired by the physical annealing process. The annealing schedule mimics the temperature variation which takes place in an actual annealing process. Initially the temperature (T) is set to a high value and then it is decreased at each step following an annealing schedule (the annealing schedule can be a custom function defined by the user). Final temperature the algorithm can reach is zero (T=0). Neighbourhood area in the solution space that the algorithm could search is determined by the value of the temperature. When the temperature values are high the search algorithm is expected to wander towards a broad region of the search space containing good solutions. As the temperature reduces (i.e. cools down) it drifts towards low-energy regions that become narrower and narrower; and finally move downhill.

Acceptance Probability

Acceptance probability defines the likelihood of making a transition form the current state to a candidate new state which is slightly worse than the current state. This feature prevents the search from becoming trapped at a local minimum that is worse than the global one.

The energies of the two states (i.e. fitness of the solutions) and the current temperature determine the acceptance probability. The acceptance probability function is usually defined so that the probability of accepting a move decreases when the difference in energies (i.e. fitness in the current state and new state) increases. This makes sure that small uphill moves are more likely than large ones. Also the temperature T plays a crucial role in controlling the sensitivity of transition between states. For higher values of T simulated annealing function is sensitive to large uphill variations and when T is smaller the function is sensitive to finer energy variations in states.

6.5.2 Design and Implementation of the algorithm

Transition from the hill climbing algorithm into simulated annealing algorithm was fairly straightforward. The only changes needed to be done was encoding the cooling schedule and acceptance probability. Simulated annealing program was implemented in Java as well.

a. Fitness function

The same fitness function which was used in the hill climbing algorithm was used for simulated annealing as well. The equation is a s follows.

$$Fitness = (\text{short leaks} * 100 + \text{medium leaks} * 10 + \text{long leaks} * 1) * \text{area} \tag{6.4}$$

The best solution in this context was the circuit layout which has the least number of leaks and minimum area. Hence a lower fitness value was considered a better fitness. Therefore the optimisation process can be considered as a minimisation optimisation.

b. Cooling schedule

As mentioned earlier simulated annealing heuristic search mimics the natural annealing process. The cooling schedule determines how the temperature is reduced in the consequent cooling cycles (i.e. iterations). The cooling schedule is composed of three main parameters; starting temperature, final temperature and the cooling factor.

c. Termination condition

The simulated annealing algorithm is programmed to complete all the cooling cycles (i.e. iterations) that were defined at the start of the algorithm. The number of cooling cycles were decided using an estimation method which is presented below. For a new circuit layout it would difficult to compute the best fitness it could achieve. The goal of opting to a heuristic search was to effectively search the solution space for the optimal or sub optimal solution. Hence it is impossible to impose a termination condition. Further, acceptance probability condition is programmed to accept solutions with the same fitness yet different structure revealing potential alternative solutions. Hence letting the algorithm complete the set number of iterations was found to be useful than imposing an early termination.

6.5.3 Estimation of the number of iterations (cooling cycles) for simulated annealing

Even though hill climbing was not an effective heuristic search mechanism that could look though the solution landscape of DNA walker circuit layouts, the algorithm was useful in determining an estimate for the number of iterations required for simulated annealing algorithm to be executed to reach an optimal solution.

The experiment was done on the first three layouts Toy, Toy0 and Toy1. For each of the layouts hill climbing program was set up to run for a long time; 30 minutes for Toy and Toy0 layouts and 2 hours for Toy1 layout as the number of iterations had to be increased to cover the possibility of a proportional relationship. With the fixed number of iterations the program was repeated 40 times for all three layouts. Afterwards the contingency plots were generated for all three layouts. Figure 6.8 is an illustration of the contingency plot generated for Toy layout by running hill climbing algorithm for 40 runs each run with 150000 iterations. A contingency plot represents the best fitness reached at each iteration when the optimisation algorithm is run.

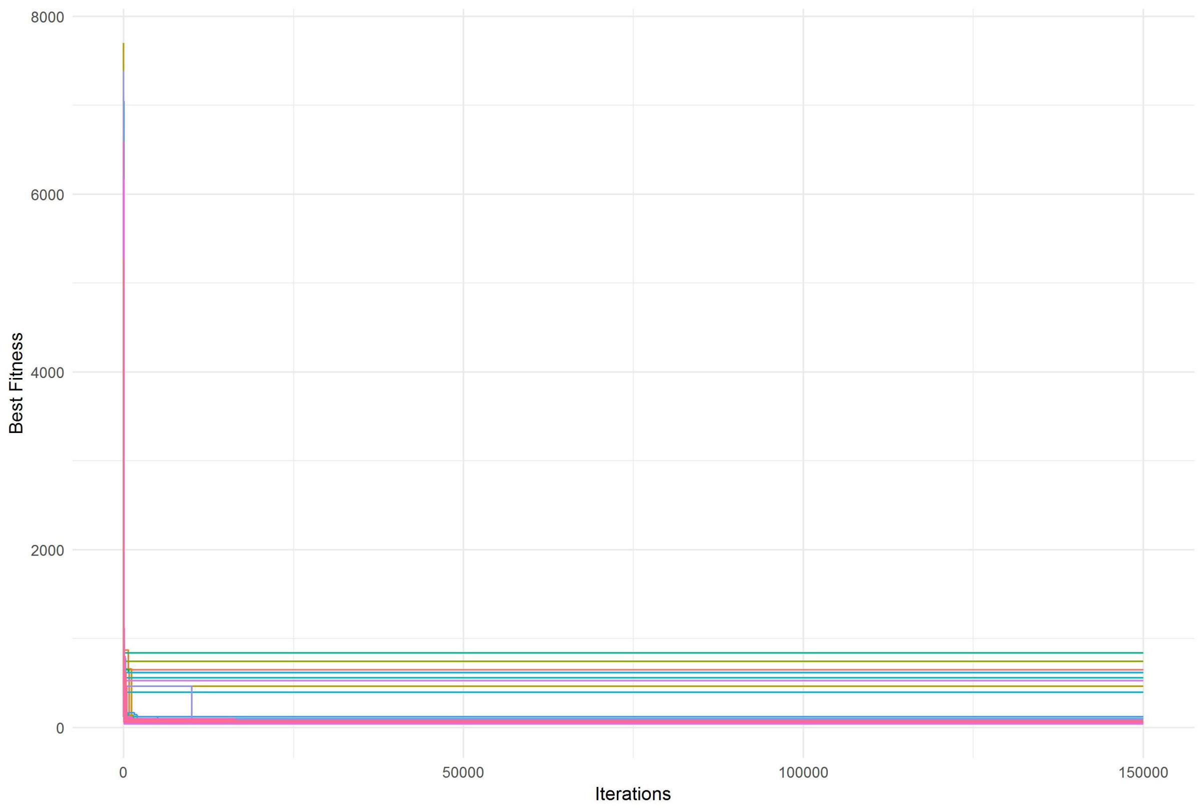


Figure 6.8: Contingency plots for Toy layout: The graph contains 40 contingency plots generated after applying Hill climbing algorithm on Toy layout for 40 times (runs). The x axis represents the number of iterations per run and the y indicates the the best fitness at each iteration.

By examining the contingency plot the maximum number of iterations that took to reach equilibrium was selected as the number of iterations required to reach the optimal solution for that particular layout. Following table 6.3 shows the maximum iteration point in which each layout has reach equilibrium. There exists a proportional relationship between the numbers; when the number of nodes in the layout is increased by one the number of iterations increases by 2.4 times. Hence for Toy2 layout we can estimate that the number of iterations should be $95147 * 2.4$ 228353. Also it includes the average execution time of a run for each layout.

Layout	No of Nodes	Equilibrium reached at	Relationship	Avg execution time per run (ms)
Toy	6	16419		3695821.5
Toy0	7	39985	$39985/16419 = 2.4$	2343345.2
Toy1	8	95147	$95147/39985 = 2.4$ $95147/16419 = 5.7$	8001359.3

Table 6.3: Equilibrium points for each toy layout - Hill climbing algorithm

Figure 6.9 depicts how the circuit area and leaks of solutions change over time during a single run of the program. The blue represents the beginning of the optimisation cycle and gradually the colour changes from purple to red. Red colour denotes the final solutions. According to the fitness function, the area and the number of leaks should decrease with the time. The figure represents this effect.

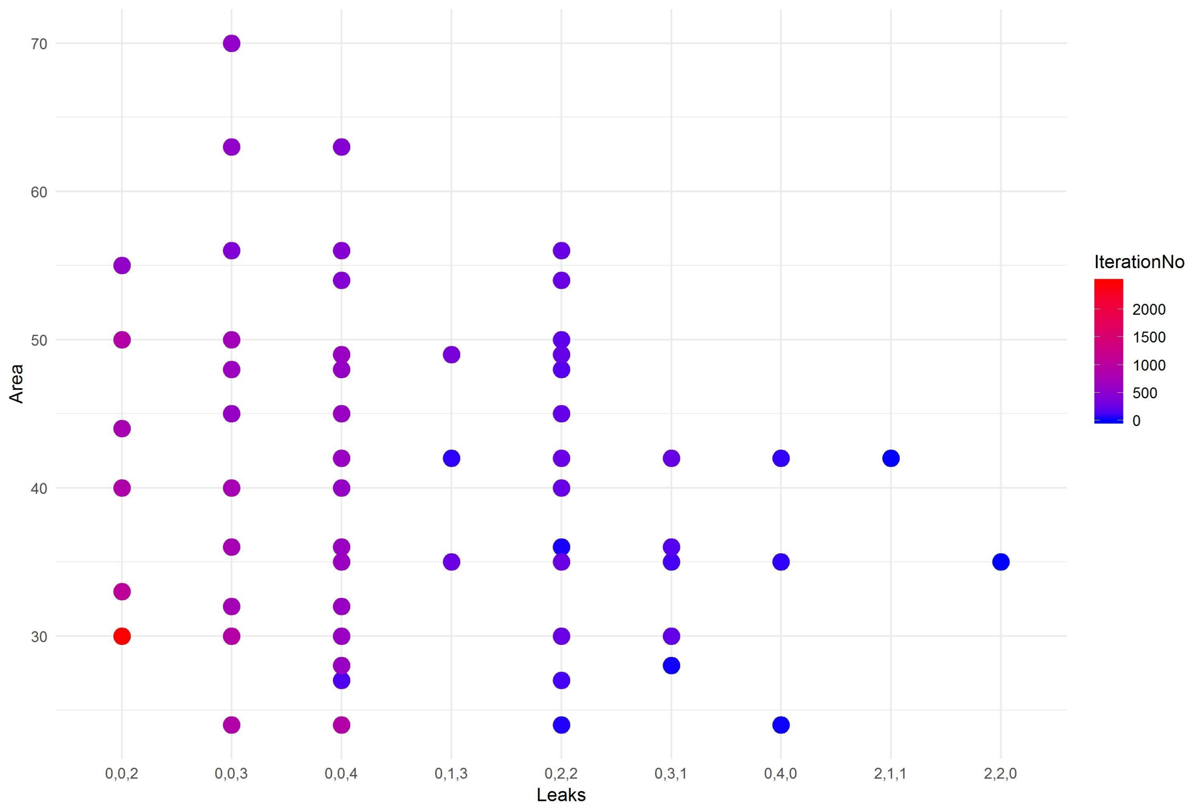
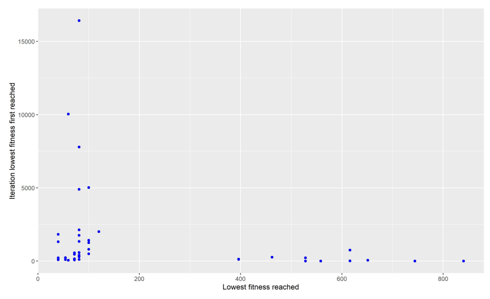
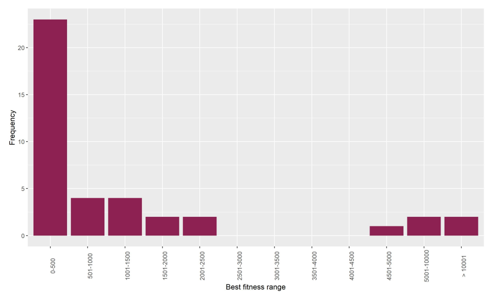


Figure 6.9: This figure represents how area and fitness changes over time as the simulated annealing algorithm optimises the solution. Blue colour represents the early stages of the optimisation process, then gradually the colour changes from purple to red. Red denotes the optimal solution for the run.

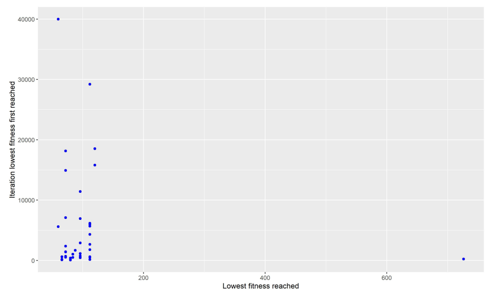
However, the execution time of one run was significantly high and in order to optimise the time, number of iterations needed to be further reduced. Hence, for each layout the lowest fitness observed at each run and the iteration in which it was reached was plotted in a scatter plot. As figure 6.10 represents, in all layouts, the lowest fitness was reached within the first thousand iterations majority of the times (70%).



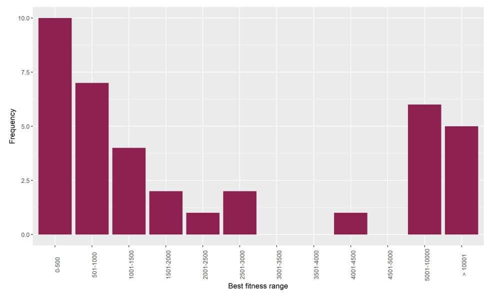
(a) Layout: Toy



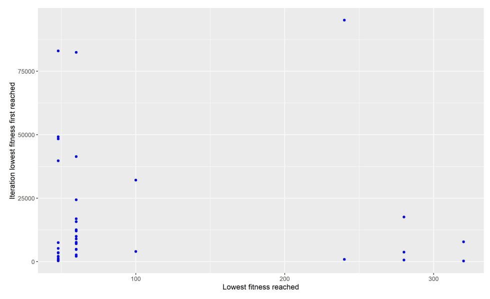
(b) Layout: Toy



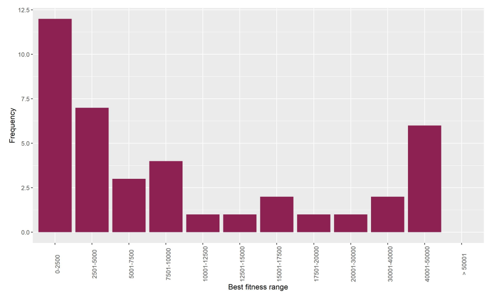
(c) Layout: Toy0



(d) Layout: Toy0



(e) Layout: Toy1



(f) Layout: Toy1

Figure 6.10: This figure presents the analysis of the relationship between number of iterations taken to reach the best fitness when applied Hill Climbing Optimisation Algorithm on layouts Toy, Toy0 and Toy1. Scatter plot in the LHS indicates the lowest/best fitness at each run against the iteration the it was first observed. Bar plot indicates the number of lowest fitness values observed within the a specified number of iterations

6.5.4 Evaluation of Results of Simulated Annealing and creating the optimisation archive

Considering all above factors the following number of iterations were used for each layout when running simulated annealing algorithm.

Circuit layout	Iterations	Runs
Toy	1500	20
Toy0	3600	20
Toy1	8700	20
Toy2	20000	20

Table 6.4: Number of iterations for each layout when applying simulated annealing algorithm

Simulated annealing algorithm was applied to the four DNA circuit layouts. Separate tests were carried out for the four different circuit layouts. As a result of the search, several kinds of circuit layouts were generated automatically.

For models Toy and Toy0 18/20 runs of the simulated annealing algorithm were able to reach the the lowest fitness value observed during the exhaustive search. The lowest fitness reached by the exhaustive search for Toy1 and Toy2 layouts, was 780. However the simulated annealing algorithm was able to generate layouts with a new lowest fitness value of 60. This proved that the simulated annealing process was effective in obtaining a layout with lowest fitness for a new circuit design.

The simulated annealing approach was beneficial in several ways for the design process of the DNA circuits.

- Efficiently exploring the solution space and reaching the optimal solution with the minimum fitness
- Identifying alternative solutions for the same fitness band
- Identifying the classification criteria for alternative solutions based on design requirements
- Establishing design constraints
- Improving future design by identifying sub-design patterns

Creation of the optimisation archive

Unlike genetic algorithms, hill climbing and simulated annealing algorithms do not have the concept of generations. Hence one run was considered as a population and the iteration number was used as a generation. Iteration number was used to impose an order on the solutions (order of appearance in time) which was helpful in observing the evolution of a solution over time. A text file was written during each run of the simulated annealing algorithm which contained the following:

- Iteration number (n)
- Fitness of the new solution
- Best (minimum) fitness value observed so far in the run
- Initial solution (layout) which is generated at the beginning of the current iteration

- New solution generated after making a small change to the initial solution in the current iteration
- Solution (layout) with the best fitness
- For each solution number of short, medium and long leaks, and area

For example, when simulated annealing algorithm was executed on the Toy layout, one run consisted of a population of 1500 solutions (1500 iterations). As the number of runs increases the populations became more diverse. However, each population was analysed individually as combining populations will affect the definition of data. Analysis of the optimisation archive is explained in the next section.

In addition to the above data a separate log was created to capture the alternative solutions generated during the a run. When simulated annealing algorithm searches for the optimal solution, at each iteration the fitness of the new solution was compared with the current best fitness (Elite fitness) observed. If the new fitness was less than the current best fitness, the best fitness and the corresponding solution were replaced with the new values. According to the implementation of the algorithm, this occurred only when the fitness value of the new solution is less than the current best fitness value (as this is a minimisation problem). During the execution, if an alternative solution (solutions with similar behaviour but different structure) was observed, it was lost. Hence, in order to capture the alternative solutions with current best fitness (elite fitness) values, a new clause was added to the program to record a solution with the new solution if the new fitness value is equal to the current best fitness value. A Java hashmap was employed to store the unique solutions for each fitness value. This information will be included in the population data which were mentioned in the previous paragraph as well, however, recoding alternative solutions on the go gives the additional benefit of promptly identifying alternative solutions. This is beneficial for larger circuits with longer execution times. If the design requirements agrees with a sub-optimal design, the execution could be terminated without having to wait for till the completion of the set number of iterations.

6.6 Step 3 - Analysis of the optimisation archive

As mentioned in the previous section the optimisation archive consists of populations of solutions created for each run. As proposed in the methodology, the archive was analysed mainly to gain the following three types of insights.

- Extract alternative solutions
- Temporal analysis - observe the structural evolution of solutions overtime to identify core design patterns
- Standardize model constraints

In order to explain the functionality, optimisation results from Toy layout is used in the following sections.

6.6.1 Identifying alternative solutions

Simulated annealing algorithm starts at a random place in the global solution space and reach for a local optimum solution within a specified number of iterations. Therefore, one run is not sufficient to make a decision on the global optimum. In order to get a better understanding of the total solution landscape, it is essential to perform multiple runs. In the DNA walker circuit layout design case study, simulated annealing algorithm was repeated for 20 times for each layout. When analysing the solution space in search of alternative solutions, all the solutions from all the runs for a given layout were analysed as one population. Figure 6.11 gives an overview of all the optimal fitness values observed during the 20 runs. The minimum optimal fitness reached by simulated annealing algorithm was 60. The other fitness values are 72, 81, 90, 96, 100, 120, 140 and 482. The most frequently observed fitness was 100 and it was observed 5/20 times.

An alternative solution can be interpreted in two different ways.

- Both structurally and functionally different - these solutions can be identified by ranking the solutions by fitness. Additionally, solutions can be chosen based on our design requirements. For example, Figure 6.12 represents the position of solutions based on area and number of leaks. If the designers have specific constraints on the area, types of leaks and the number of leaks allowed, solutions could be selected as well.

  Figure 6.13 represents the alternative solutions by area. In the total population there are 26 unique circuit area values and the frequency is the number of unique solutions with the same circuit area. However, these solutions have different fitness values and leak values. Figure 6.14 represents alternative solutions based on area.

  Figure 6.15 represents the alternative solutions by leak bands. In the total population there are 11 unique leak bands and the frequency is the number of unique solutions with

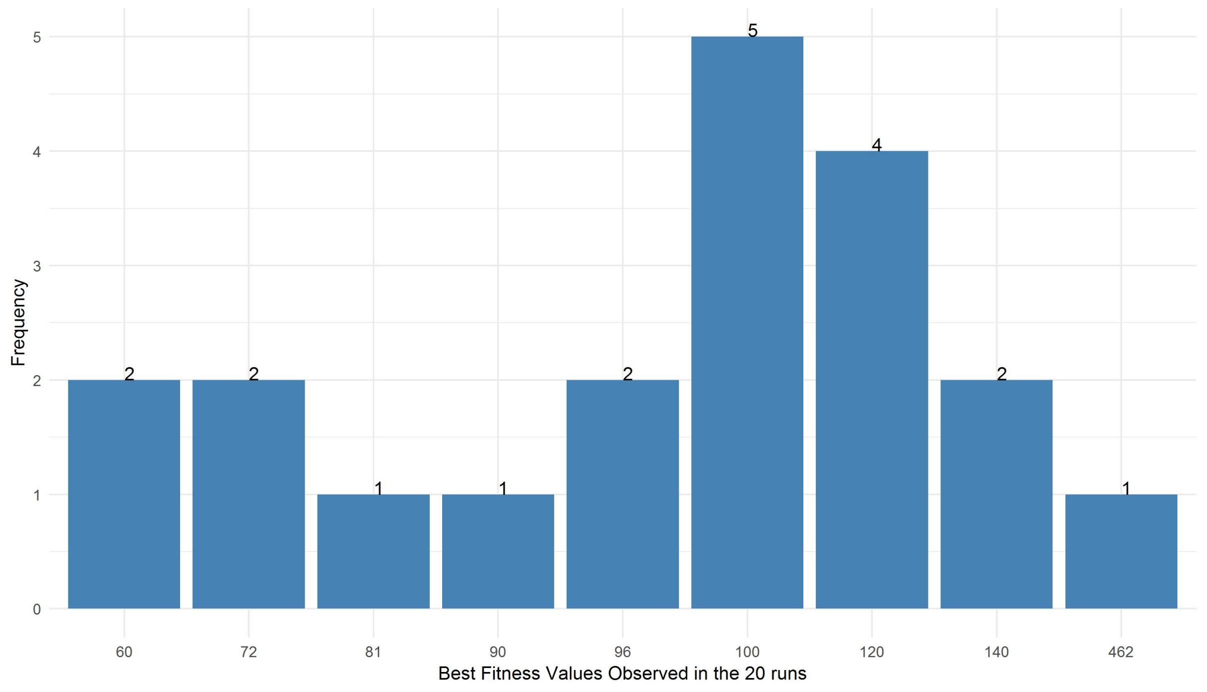


Figure 6.11: The figure represents the frequency of the best fitness values observed for the Toy layout when simulated annealing (SA) algorithm was executed. The data consists of fitness values observed for 20 runs of the SA each with 1500 iterations

the same leak ban. A leak band is a single value consisting of the number of small, medium and large leaks in order. For example $[0, 0, 2]$ means a circuit with 0 small leaks, 0 medium leaks and 2 large leaks. The alternative solutions by leak bands too have different fitness values and area values. Figure 6.16 represents alternative solutions based on leaks.

- Structurally different, yet behaviourally similar solutions

  The research work is more focused on identifying behaviourally similar yet structurally different solutions. Figure 6.17 represents the unique fitness values observed in the total population for Toy layout design.

  Figure 6.18 provides a cleared view of the top 20 fitness values observed.

  Figure 6.19 includes the alternative solution for fitness value 66. These solutions are behaviourally similar as they have the same fitness, number of leaks and circuit area. However the structure is different.

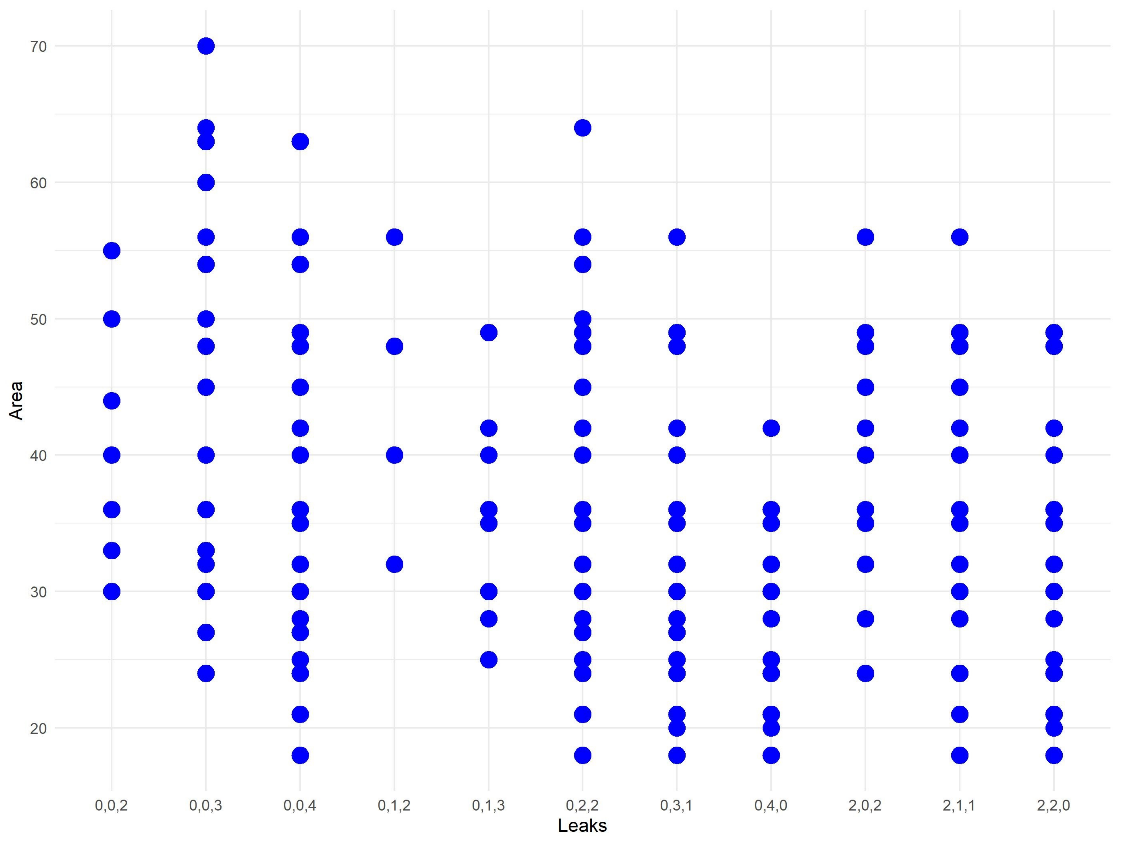


Figure 6.12: This figure represents where each solution fits in a area vs leaks landscape for all solutions observed in the total population of Toy layout. There are 11 leak bands: each band consisting of the number of leaks mentioned in the order of short, medium, large, in the whole population.

6.6.2 Establishing design constraints

Another main advantage of using a computational optimisation method was it gave a ability to fiddle with design constraints and verify constraints. For example the simulated annealing algorithm helped to create a new rule in determining a maximum and minimum grid size for the DNA walker circuit. Identifying this had a significant improvement in the solution space. Following is a detail description of how the constraints were checked.

Finalizing the layout constraints for Toy0 circuit

In my previous report I explained a case where DNA layout validation constraints inbuilt in Marcie (DNA circuit validation software) was in contradiction with the constraints Prof. Gilbert and I used to develop our DNA layout generation/ validation program. To enumerate the problem; a node of type “NORM” can have 2 or 3 short distances. However Marcie rejects the

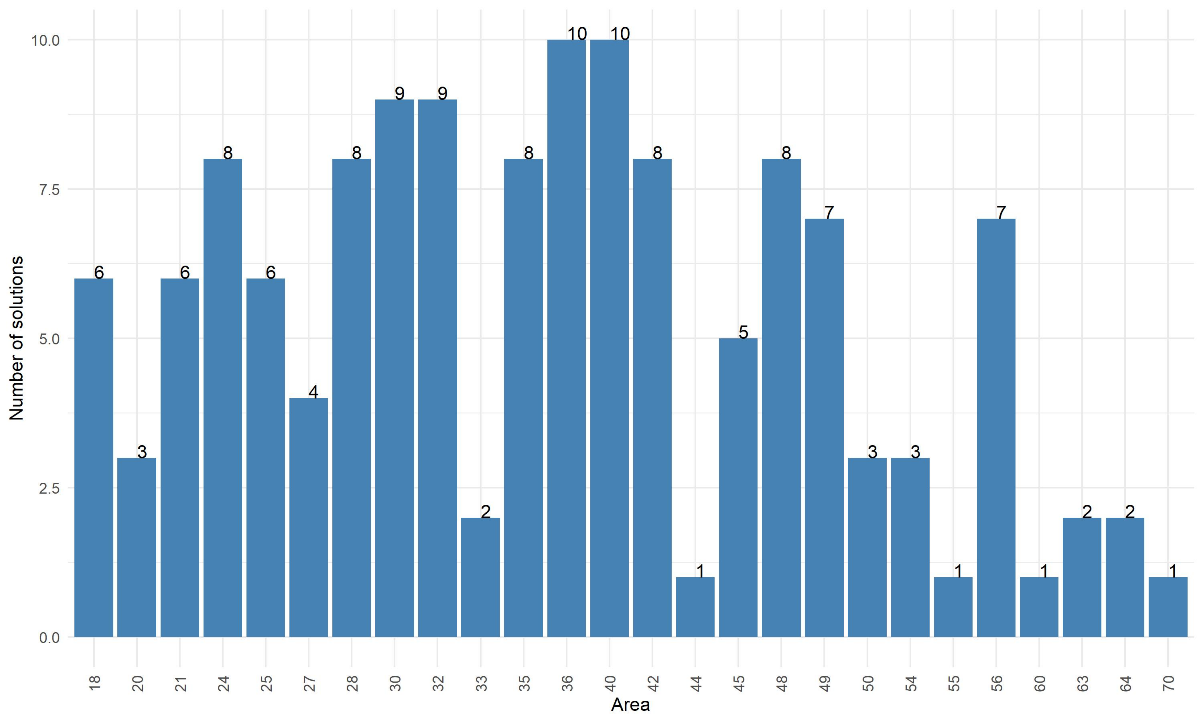


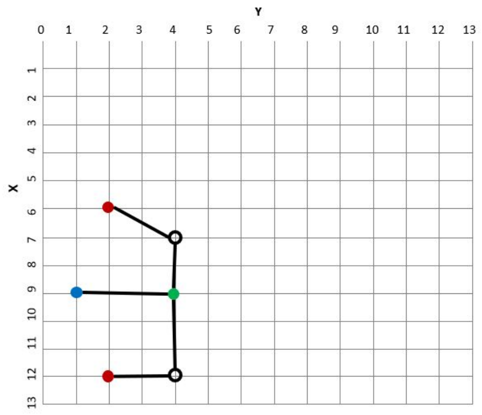
Figure 6.13: This figure represents the number of unique solutions observed for each circuit area in the Toy layout population. There are 26 unique area values with a minimum value of 18 and a maximum value of 70.

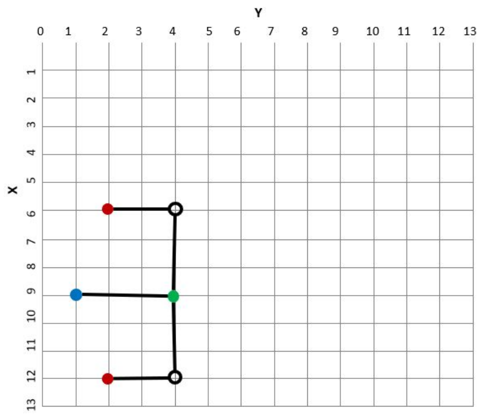
following layout (see figure 6.20) where the N0 node (of type NORM) has 3 short distances.

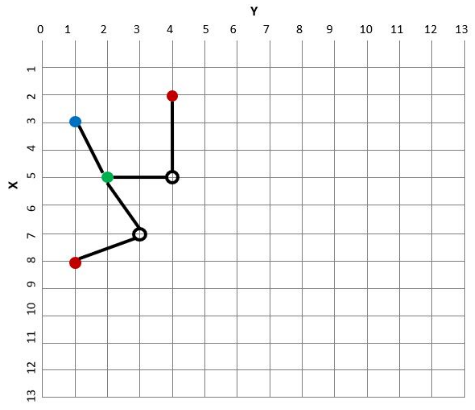
We consulted the collaborating team in Germany who developed Marcie software. As per their explanation DNA walker circuits are undirected graphs and the arcs between nodes do not exist in reality. Therefore when a walker takes a specific route the direction is determined by the distance between nodes. Further to obtain a useful layout a series of assumptions are made. One of them is that fork nodes have only one predecessor. Marcie's analysis simulates a DNA walker and walks around the net. In the incorrect layout Marcie reaches the fork node on two different paths (see figure 6.21) and this violates the aforementioned assumption. Therefore the walker cannot decide which one is the correct predecessor of the fork node. Hence it's an illegal layout.

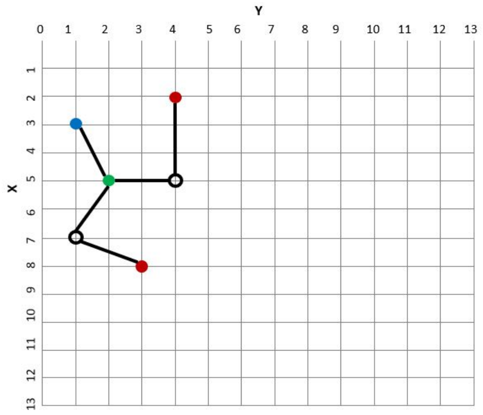
Finalizing the maximum grid size for each layout and formalizing how to compute it

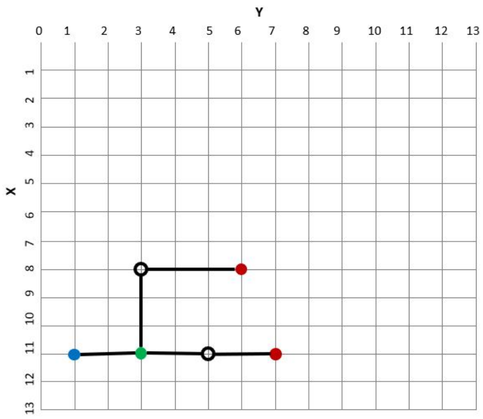
For the experiments carried on the DNA circuit layouts there was no standard grid size defined. It was a predetermined value. Therefore Prof Gilbert and I decided to experiment with the layout size to determine the maximum and minimum gird size for a layout. The goal of our search is


(a) Layout: 9194c474c262, Fitness: 396, Leaks:[0,2,2]


(b) Layout: 9194c464c262, Fitness: 72, Leaks:[0,0,4]


(c) Layout: 315273548124, Fitness: 3798, Leaks:[2,1,1]


(d) Layout: 315271548324, Fitness: 558, Leaks:[0,3,1]


(e) Layout: b1b383b586b7, Fitness: 420, Leaks:[0,4,0]

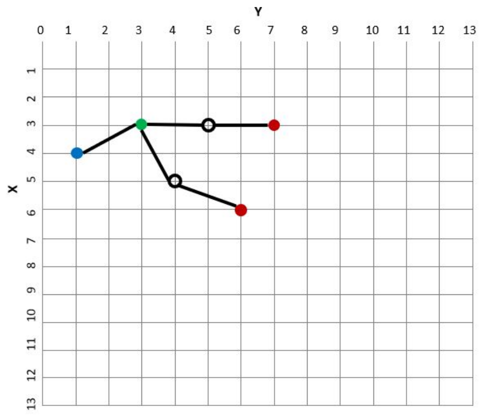

(f) Layout: 413335543766, Fitness: 3960, Leaks:[2,2,0]

Figure 6.14: Alternative solutions by area (both structurally and behaviourally different alternative solutions). These layouts have the same area of 18, however the fitness and leaks values are different to each other.

to identify all possible layouts for a given circuit and the search should not be dictated/ biased by the grid size. Therefore it is important to determine a maximum and minimum grid size.

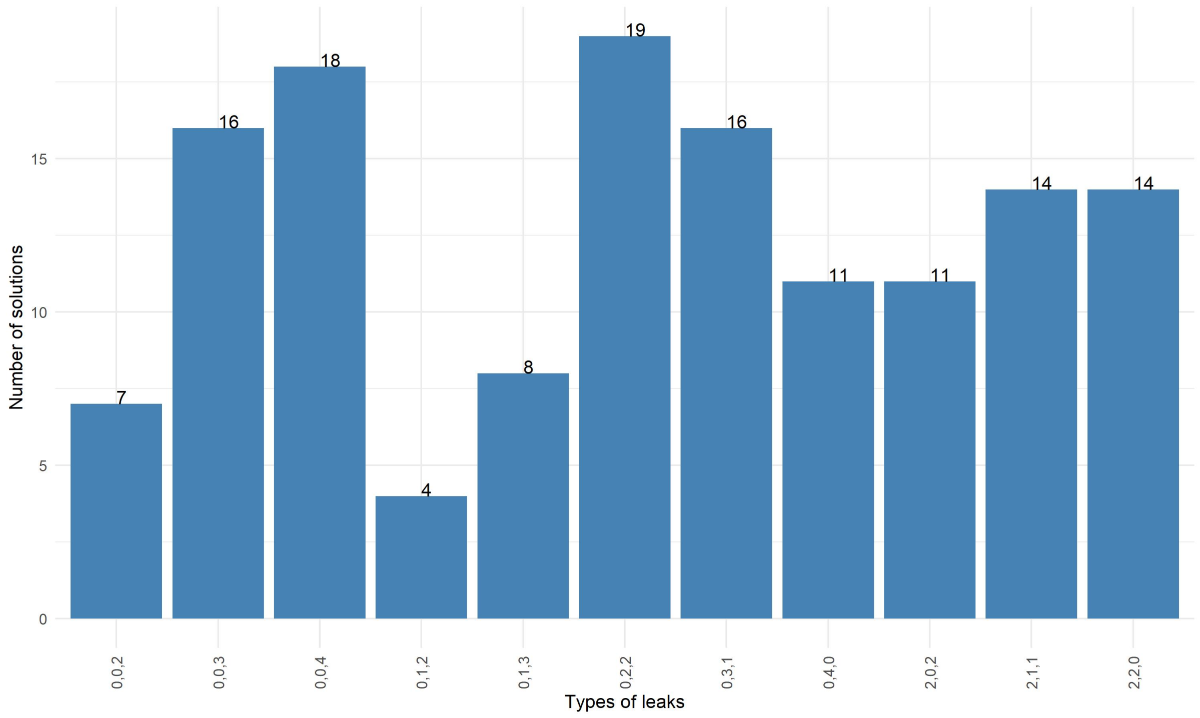


Figure 6.15: This figure represents the number of unique solutions observed for each leak band in the Toy layout population. There are 11 unique leak bands for the Toy circuit layout in the population observed.

The first constraint we imposed was one branch of the layout should be able to laid out on a straight line along one of the axis. For example let's consider the Toy layout. Toy layout has 6 nodes. And there are two branches labelled in orange and blue in figure 6.22. Therefore each branch has 4 nodes. The maximum short distance between two nodes according to Manhattan distance is 3.

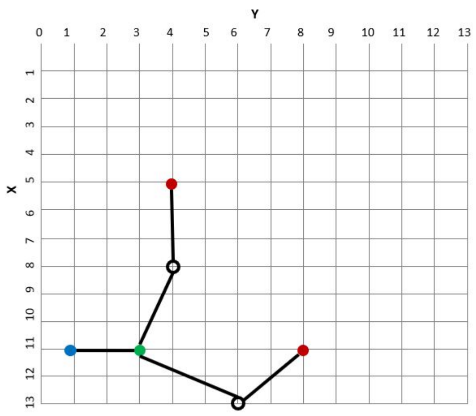
Therefore in order to lay out one branch in a straight line along any axis the maximum grid side should be;

$$Maximumdistance = (noofnodesinonebranch - 1) * 3 + 2 \tag{6.5}$$

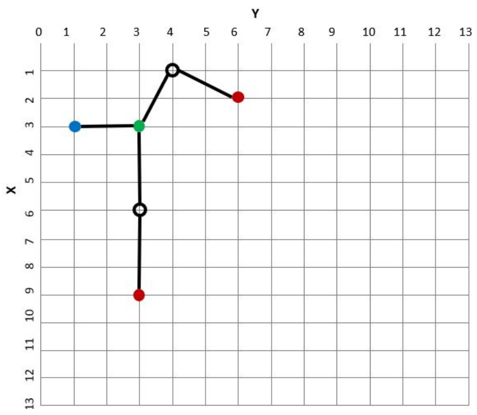
2 have to be added to the answer because nodes are not placed on the grid borders. Therefore for the Toy layout maximum grid size should be

$$Toy_{maxdist} = (4\breve{}1) * 3 + 2 = 11 \tag{6.6}$$

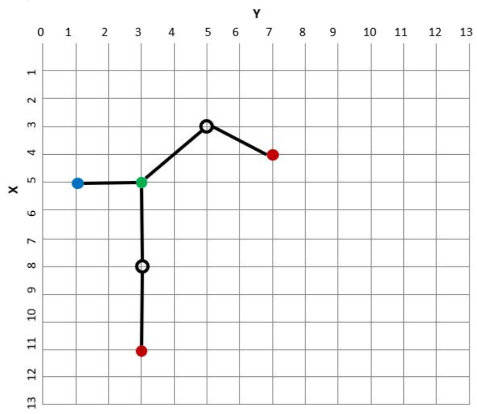
We experimented with the new grid size and the solution space increased significantly. Further we were able to obtain a new best fitness value for the Toy layout. Previously it was 100



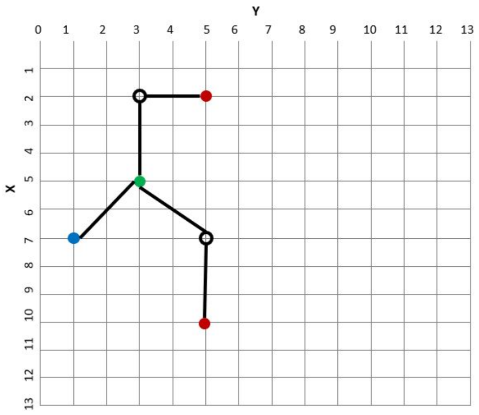
(a) Layout: b1b484d654b8, Fitness: 672, Area:56



(b) Layout: 313314632693, Fitness: 480, Area:40



(c) Layout: 5153358347b3, Fitness: 576, Area:48



(d) Layout: 7153237525a5, Fitness: 384, Area:32

Figure 6.16: Alternative solutions by leaks (both structurally and behaviourally different alternative solutions). These layouts have the same number of leaks of [0,1,2] (no short leaks, 1 medium leak and 1 long leak), however the fitness and area values are different to each other.

and the current best (lowest) fitness was 84. However when examining the layouts with the best fitness I observed the below layout.

I noticed that the FINAL nodes in the layout in figure 6.23 (left) could be further extended as depicted in the second figure (right). Applying the same rule for the maximum distance between two consecutive nodes, the branch to be laid out on a straight line the distance should be;

$$Toy_{maxdist} = (5\breve{\ }1) * 3 + 2 = 14 \tag{6.7}$$

Next we tested with the new grid size 14 * 14 and the solution space increased further. Also we tested by increasing the grid size further up to 15 but the solution space remained the same. This experiment helped us to standardise the method to determine the grid size for any given layout. Given a layout with uneven branch lengths; Maximum width: branch with the maximum number of nodes should be able to laid out on a straight line

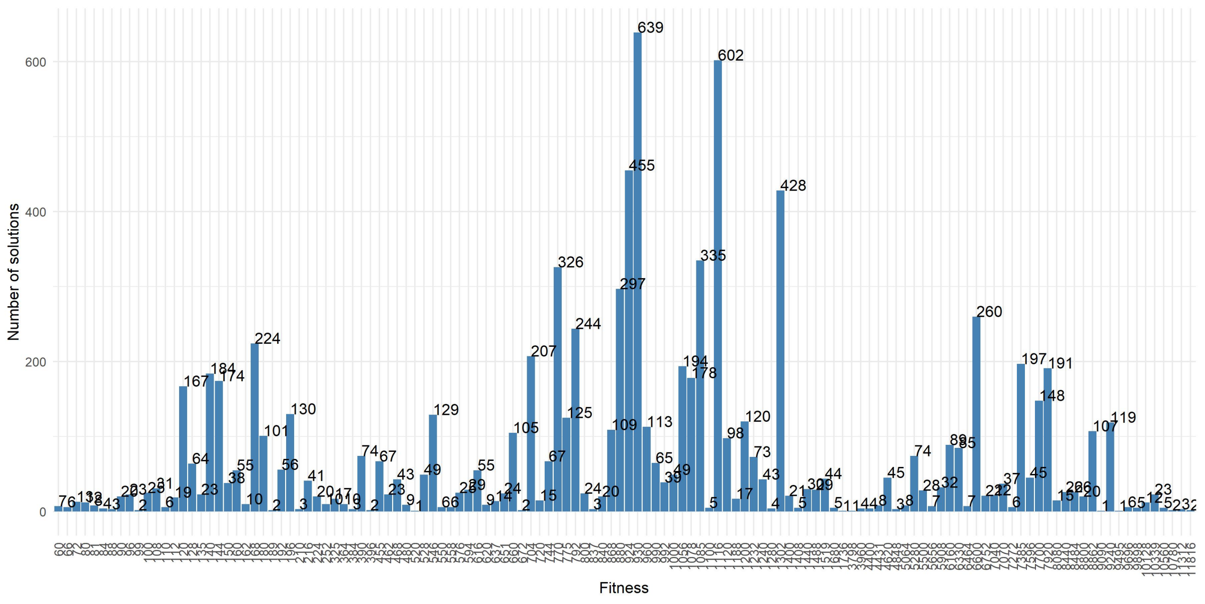


Figure 6.17: The figure represents the number of the alternative solutions (similar fitness but structurally different) for the fitness values observed for the Toy layout when simulated annealing (SA) algorithm was executed. The data consists of all unique fitness values observed for 20 runs of the SA each with 1500 iterations

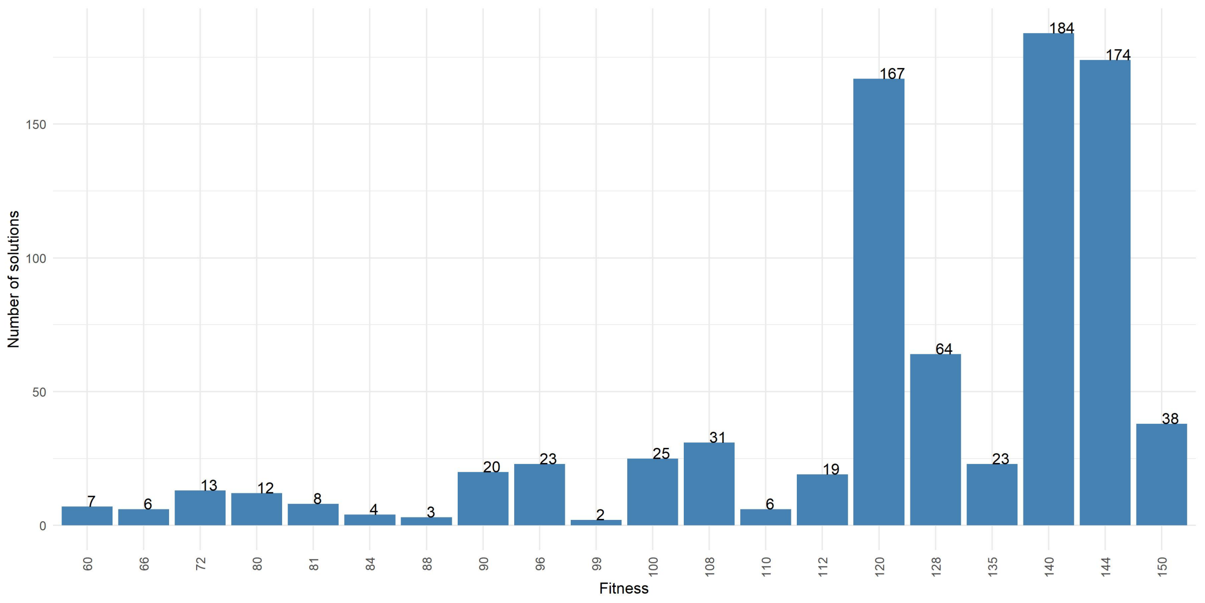
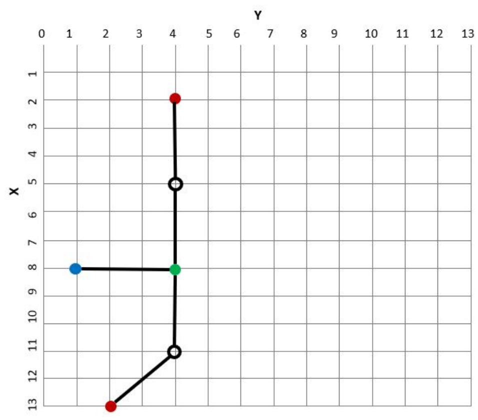
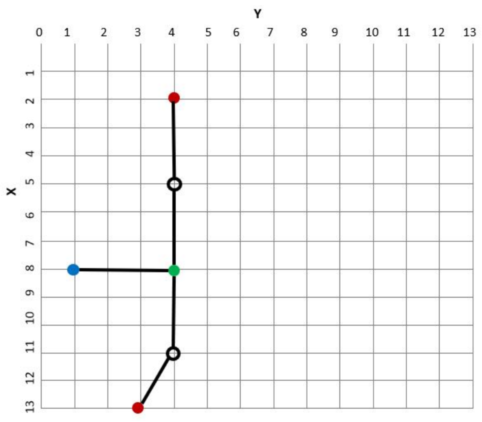
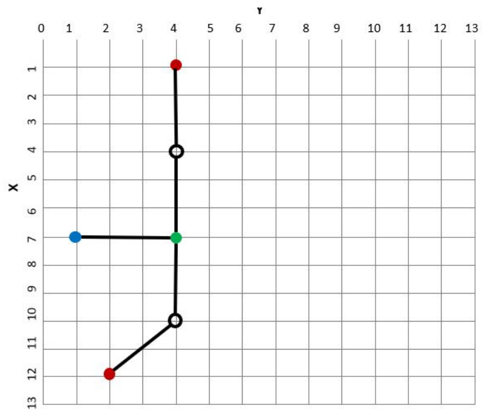


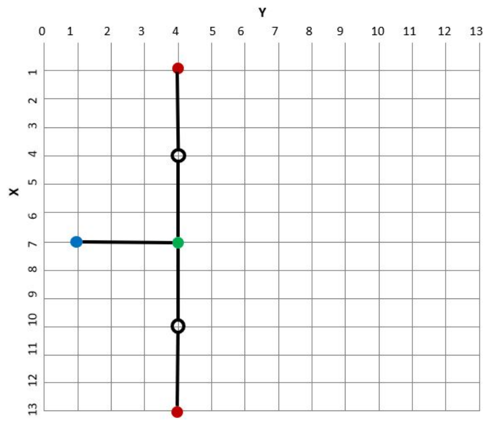
Figure 6.18: The figure represents the number of the alternative solutions (similar fitness but structurally different) for the fitness values observed for the Toy layout when simulated annealing (SA) algorithm was executed. The data consists of all unique fitness values observed for 20 runs of the SA each with 1500 iterations

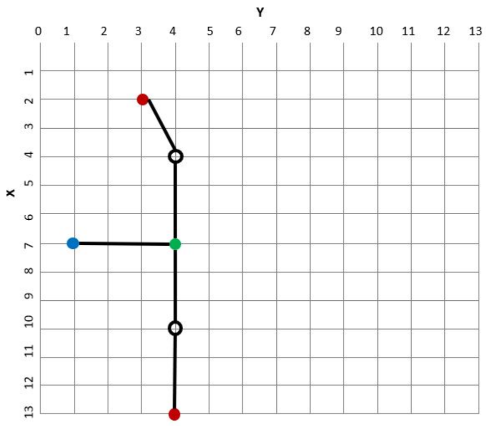
$$Maximumwidth = (maximumnoofnodesinabranch - 1) * 3 + 2 \tag{6.8}$$


(a) Layout: 8184b454d224


(b) Layout: 8184b454d324


(c) Layout: 717444a414c2


(d) Layout: 717444a422d4


(e) Layout: 717444a423d4

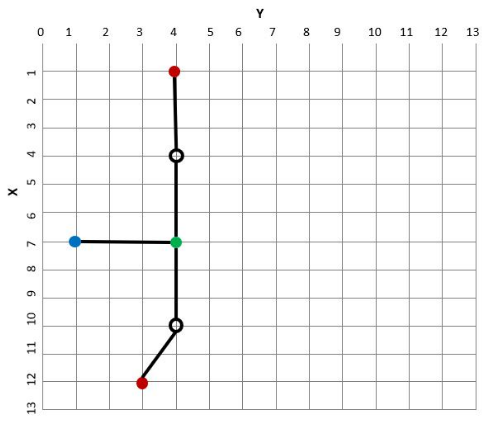
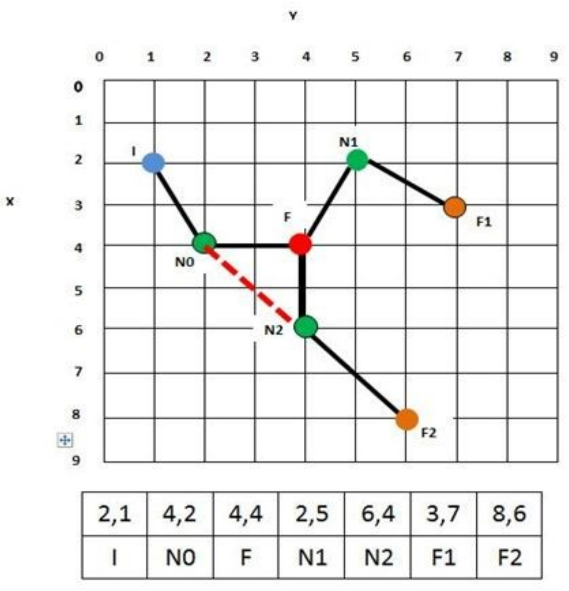
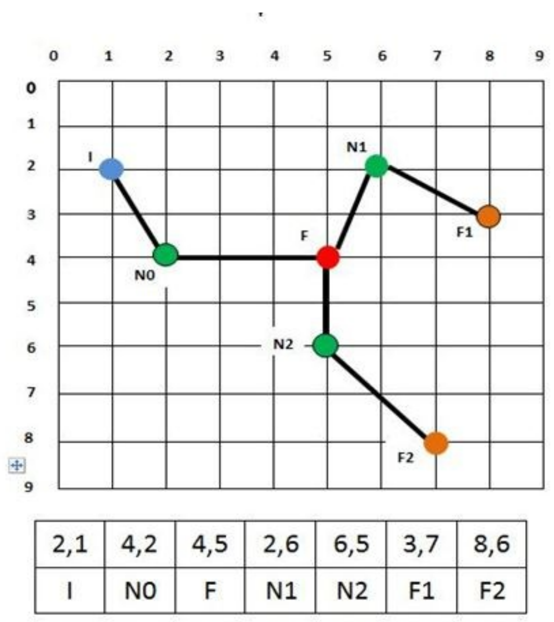

(f) Layout: 717444a414c3

Figure 6.19: Alternative solutions by area (both structurally and behaviourally different alternative solutions). These layouts have the same area of 18, however the fitness and leaks values are different to each other.

Maximum length: fork node and the two branching out sections should be laid out on a straight line



2,1	4,2	4,4	2,5	6,4	3,7	8,6
I	N0	F	N1	N2	F1	F2



2,1	4,2	4,5	2,6	6,5	3,7	8,6
I	N0	F	N1	N2	F1	F2

Figure 6.20: The first layout (left) is a Tpy0 layout that was evaluated to be incorrect by Marcie. The second layout (right) is a similar layout that Marcie evaluated to be correct. As you can see node 'N0' has 3 short distances and 'No' in the second layout has only 2 short distances.

$$Maximumlength = (noofnodesaftertheFORKnodeincludingtheFORKnode - 1) * 3 + 2 \quad (6.9)$$

Figure 6.24 shows an analysis of the maximum and minimum grid sizes for the four DNA circuit layouts Toy, Toy0, Toy1 and Toy2. It is evident that changing the grid size has significantly increased the solution space.

6.6.3 Improving future design by identifying sub-design patterns using temporal analysis

The third insight which could be obtained from analysing the optimisation archive is the evolution of a solution over time. The time aspect is emulated with the use of iteration number. Hence, the optimisation results from different runs are analysed separately.

The approach used was to select the solutions which were selected as the current best solution (elite solution) and visually analyse the layouts from the worst best fitness to the least best fitness (because this is a minimization problem). Two main sub patterns were identified for the I, F, N1 and N2 nodes when the fitness value reached a value below 100.

Figure 6.25 shows an example of how the simulated annealing algorithm optimised the struc-

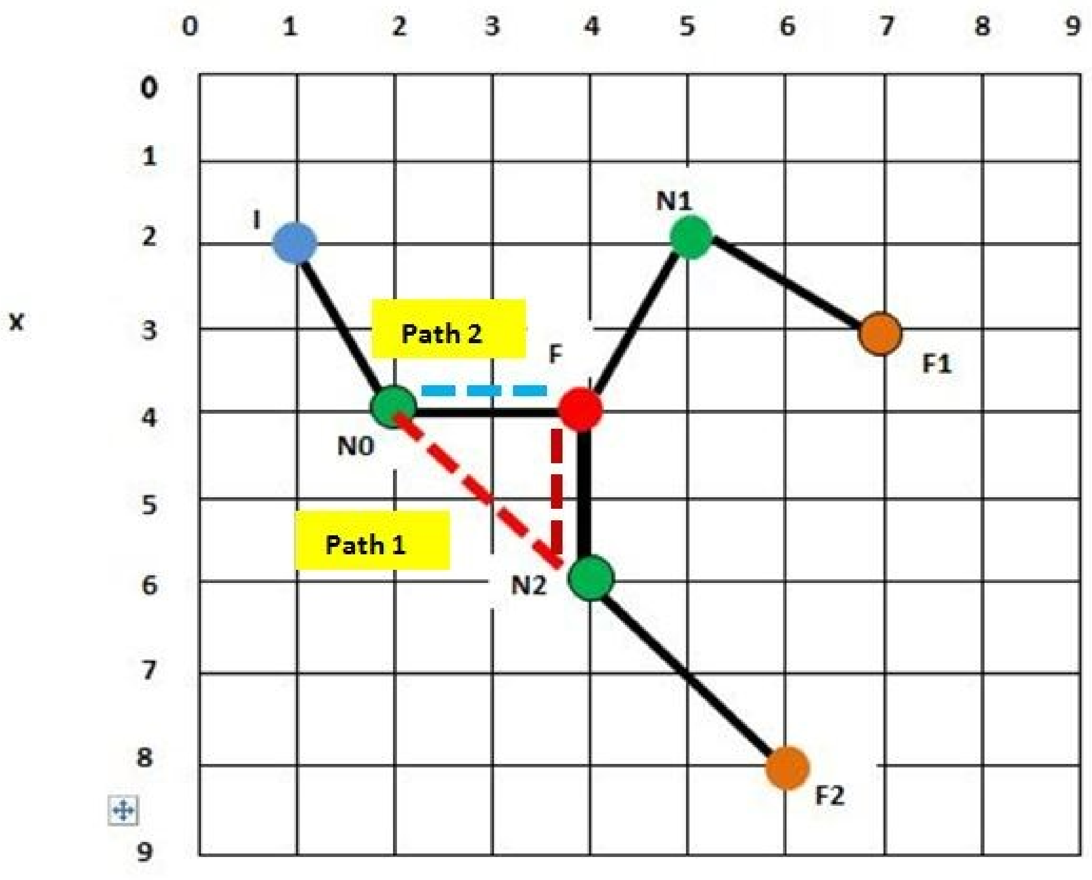


Figure 6.21: Two possible paths that can reach the FORK node in the incorrect layout

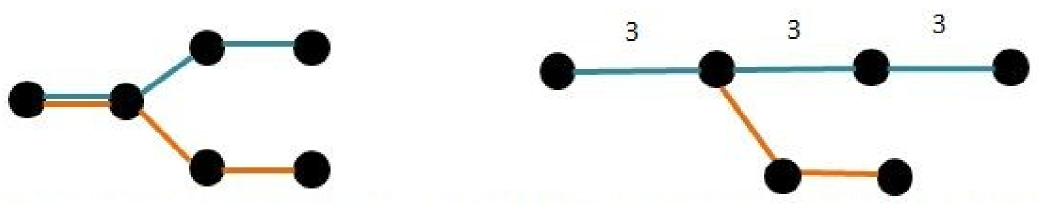


Figure 6.22: Representation of the Toy layout: general layout (left) and laying one branch on a straight line (right)

ture of a Toy layout with an initial fitness of 1488 to fitness of 72 in 5 steps.

The I, F, N1 and N2 nodes form in a T shapes as shown in figure 6.26a. 7 out of 20 runs reached an optimal fitness values less than 90. Out of the 7 runs 3 demonstrated the T pattern where the structure was stable.

Figure 6.27 shows the second sub design observed when the fitness value reaches a value below 90. In this sub shape the nodes I, F, N1 and N2 makes a y formation. 4 out of the 7 runs which reached a fitness value less than 90 had this pattern. Simulated annealing algorithm

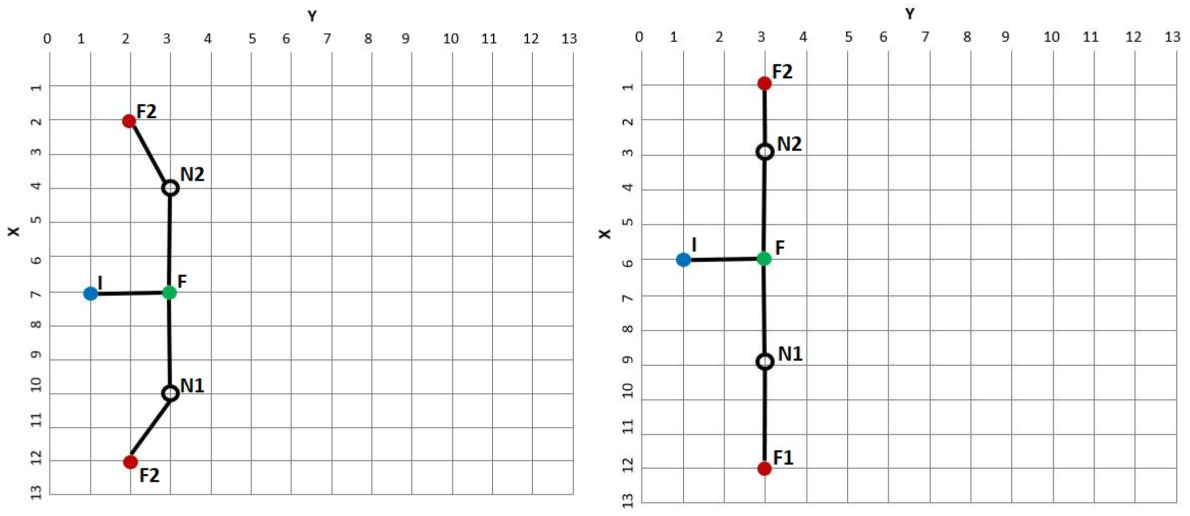


Figure 6.23: Finalizing the gird size for a layout: With the updated grid size I was able to obtain a new layout with a new fitness of 84 (left). However, I noticed that the FINAL nodes in the left hand side layout could be further extended to place the nodes in a vertical line as in the right hand side layout. Therefore maximum grid size can be defined as in algorithm 6.7

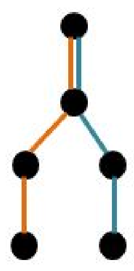
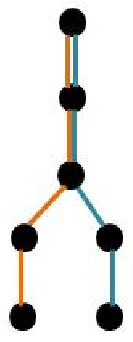
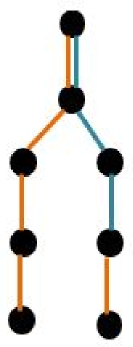
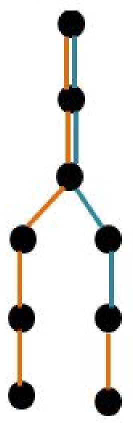
	Toy	Toy0	Toy1	Toy2
Circuit layout				
No of nodes	6	7	8	9
Initial grid size	7 * 8	9 * 9	10 * 10	15 * 15
Initial solution space	5944	208,492	77,338	Could not generate
New Max width	(4 – 1) * 3 + 2 = 11	(5 – 1) * 3 + 2 = 14	(5 – 1) * 3 + 2 = 14	(6 – 1) * 3 + 2 = 17
Max length	(5 – 1) * 3 + 2 = 14	(5 – 1) * 3 + 2 = 14	(7 – 1) * 3 + 2 = 20	(7 – 1) * 3 + 2 = 20
New solution space	263,920	786,942	394,001	640,608

Figure 6.24: Table shows an analysis of the maximum and minimum grid sizes for the four DNA circuit layouts Toy, Toy0, Toy1 and Toy2 with the newly defined constraints. Changing the grid size has significantly increased the solution space.

starts with a solution of fitness 594 and optimises the structure into fitness value 90.

Figure 6.26b represents the second static formation of I, F, N1 and N2 nodes in an optimal solution.

Identification of optimal formations of sub structures is important when working with larger circuits. As the number of nodes in the circuit increases, the number of variable locations the optimisation program needs to change during an iteration increases as well. Therefore the global solution spaces becomes larger and more iterations and runs would be required to identify the global optimal. This would inevitably increase computational time. In order to mitigate this issue, the optimal sub structures could be helpful. These optimal sub structures provide the relationships between a set of nodes. For example the T shape (figure 6.26a) indicates that the most optimal placement of I, F, N1 and N2 nodes when they are placed in a T shape and [I and F], [F and N1] and [F and N2] should be separated by 3 squares each. In the Y formation (figure 6.26b) the nodes I, F, N1 and N2 shoudl eb placed in a Y shape where I and N2 are 2 squares apart from F, and N1 is 3 squares apart from F. These relationships can be used to accurately predict the optimal substructures instead of randomly placing the nodes on the Cartesian plane. Subsequently it will improve the search and reduce overall time taken to reach the optimal solution. Currently the temporal analysis is done manually. In the further work section I have explained how this could be automated in order to identify the sub patterns automatically.

6.7 An Algorithm to Structurally Compare Circuit Layouts

The next challenge was to find a suitable distance metric to compare the circuit layouts. Circuit layouts have 3 characteristics namely leaks, area and fitness. However these values do not give a clear distinction of the layouts in a comparison computation as there are different layouts with the same area and number of leaks (see Figure 6.28).

Further there are some layouts that have a rotational symmetry. For example the first two layouts in 6.28 may look different but once the second layout is flipped 180 degrees about the Fork node they are the same layout. Figure 6.29 is a visualization of 2 layouts with a rotational symmetry.

DNA layouts can be considered as unidirectional isomorphic graphs (same number and type of nodes and the topology [connection between nodes] is same). I tried searching for an existing graph comparison method and was unsuccessful. Afterwards Prof Gilbert and I came up with two different algorithms to compare the layouts inspired by superimposition and can be used to compute the pair-wise distance between two layouts. If the two layouts are L1 and L2;

Prof. Gilbert's method:

- Superimpose the layouts by aligning two corresponding nodes in both layouts at a time (i.e. $L1_I$ and $L2_I$) and compute the Euclidean distance between rest of the corresponding pairs of nodes and add the distances e.g. In a Toy model if the INIT node from both layouts were superimposed, corresponding node pairs will be $(L1_F, L2_F)+(L1_N1, L2_N1)+(L1_N2, L2_N2)+(L1_F1, L2_F1)+(L1_F2, L2_F2)$. This step was repeated by superimposing of all the corresponding node pairs.
- Rotate one layout by 180 degree and repeat the process
- Select the minimum distance as pair-wise distance between two layouts

My method:

- Superimpose the layouts by aligning the layouts on the x and y axis
- Compute the Euclidean distance between corresponding nodes and add the distance
- Rotate one layout by 180 degrees and repeat the process
- Select the minimum distance as the pair-wise distance between two layouts

After defining the procedure these methods were tested by hands on the three sample layouts (see Figure 6.28). Both distance measures were proven to be a metric. All nodes were compared against all nodes in Prof Gilbert's method where as in my method only corresponding nodes were compared once. When computed the distance both methods gave similar answers and distance measure was able to cluster the three examples accurately. In Prof. Gilbert's methods points were moved off the grid. However there was one drawback in my method. For an instance in a larger grid the same layout is placed in the top half and the second layout is placed in the bottom half and as I am superimposing the layouts by the axis it could result in the larger dissimilarity due to positioning rather than actual shape.

Next Prof Gilbert consulted a colleague of his Prof Juris Viksna who specialises in the area and he suggested two widely used comparison methods; Geometric hashing (Wolfson and Rigoutsos 1997) and RMSD minimisation (Sadeghi et al. 2013). Both methods are often used to measure the physical distance between 3D molecular structures (i.e. proteins). Geometric hashing is computationally expensive procedure whereas RMSD (Root Mean Square Deviation) minimisation was simple to implement. Therefore I started off with the RMSD minimisation. As with the first two algorithms I computed the distances for the same layouts by hand and confirmed it is suitable. Next I implemented it in Java.

6.7.1 RMSD (Root Mean Square Deviation) minimisation

RMSD is a pair-wise distance comparison method. One structure can be compared to a different structure of the same type, or relative to a base model (synthetic/ real) or relative to a mean structure (synthetic/ real). When applying RMSD to compare DNA walker circuits, I will be performing a pair-wise comparison of two circuit layouts of the same circuit type.

Next when superimposing structures a reference point should be defined. There are two main widely accepted methods for this; least square fit and geometric hashing. Due to the complexity of the algorithm and time constraints I opted for the least square fit. In the least square fit model, the model is superimposed in manner which minimizes the RMSD value.

RMSD definition

Given two layouts (structures) $L1$ and $L2$ with identical number and types of nodes (n), and arc connections nodes in $L1$ and $L2$ can be defined as;

$L1 = L1_{ni} | i = 1, 2, \ldots, m$

$L2 = L2_{ni} | i = 1, 2, \ldots, m$

$$RMSD = \sqrt{,}\frac{1}{m}\sum_{i=1}^{m}, |L1_{ni} + L2_{ni}|^2$$

Application of RMSD minimisation to layouts

As mentioned above the algorithm will only compare layout structures which has an identical number of nodes and type of nodes. In order to illustrate using an example the 2 Toy circuit layouts in figure 6.30 will be used.

Algorithm is designed as follows:

- The first layout is placed on the Cartesian plane as it is
- The second layout is placed on the same plane, so the I node falls on the $(1, 1)$ position (see Figure 6.31a). The back layout is layout 1 and yellow colour is used to represent

layout 2. Once the second layout is placed it should be a valid placement according to the DNA circuit layout constraints.

According to the DNA circuit layout constraints, a valid placement has to conform to the following conditions.

– No nodes can be placed outside of the grid

– No nodes can be placed on the edges of grids $X = 0$, $X = maxGridX$, $Y = 0$ and $Y = maxGridY$, For a Toy layout maximum grid size is 13. Hence $maxGridX = maxGridY = 13$

Hence the initial placement of layout 2 is not a valid placement (see figure 6.31a). As it is not a valid placement pair wise distance is not computed.

- Next the 2nd layout is moved along the $Y = 1$ axis by one. Now the I node is at $[2, 1]$ position. Again the algorithm will check whether the placement is valid. According to the above mentioned constraints it is not (see figure 6.31b). Hence the distance is not computed.

- Again the 2nd layout is moved by 1 along the $Y = 1$ axis. This results in a valid positioning of the layout on the grid. Hence the pairwise RMSD distance is computed (see figure 6.31c).

- As explained in the previous sub section, the layouts demonstrate rotational symmetry. In order to accurately calculate the distance between layouts, a rotation will be performed. Mainly two types of rotations will be performed.

  – Rotation along the $I - F$ branch: The $I - F$ branch will be taken as a reference point when making the rotation. If the $I - F$ branches of both layouts are parallel (i.e. have the same X coordinate or Y coordinate) no rotation will be performed to align the axes. Else the second layout will be rotated to align so the $I - F$ branches of bot layouts are aligned (see figures 6.32a, 6.32b, 6.32c and 6.32d).

  – Rotation about the F node - in each pairwise comparison, the second layout will be rotated 180 degrees about the F node to identify rotational symmetries (see figures 6.32e and 6.32f).

- Similarly the I node of the second layout is moved by 1 until the value of $X < maxGridX$. Each time the pair wise RMSD is computed using the algorithm 4. Next the layout is rotated about the F node by 180 degrees and the pairwise distance is computed. Finally the minimum distance out of all pairwise distance is selected as the RMSD distance between the two layouts.

Algorithm 4 Compute pairwise RMSD distance

Input: Coordinate strings for two layouts
Output: RMSD distance between L1 and L2

```
1: procedure ComputePairwiseRMSDDistance
2:     $L1 \leftarrow$ Coordinate string for layout 1
3:     $L2 \leftarrow$ Coordinate string for layout 2
4:     $minRMSDDist \leftarrow 0$
5:     $currentRMDS \leftarrow 0$
6:     for $i \leftarrow 1$ to maxYGrid do
7:         $S' \leftarrow$ computeRMSD($L1$, $L2$)
8:         if $currentRMDS'$ better $minRMSD$ then
9:             $minRMSD \leftarrow currentRMDS$
10:        end if
11:    end for
12:    return $minRMSD$
13: end procedure
```

The pair wise RMSD computation was developed using Java. The comparison algorithm was applied to a sample set of layouts which accurately grouped the similar structures. 13 solutions from the 9*th* simulated annealing run of the Toy was selected. Once the pair wise distance computation was completed R was used to perform the clustering.

Hierarchical clustering was used to cluster the layouts. The main reasons to opt for hierarchical clustering were a) hierarchical clustering algorithm can easily accommodate a pre-computed distance matrix as the dissimilarity matrix. This isn't possible with certain clustering algorithm such as kmeans; b) the clustering of 13 solutions was a simple task and hierarchical clustering algorithm clustered the layouts accurately; c) hierarchical clustering has been used for previous analysis in other example applications; and, d) R has inbuilt methods to perform hierarchical clustering and cluster validation methods. Figure 6.33 represents the cluster dendrogram produced by hierarchical clustering. The goodness of the clusters was determined using Silhouette index in R. Silhouette index suggested 4 clusters was the best suitable for the given sample with a value of 0.38. Figure 6.34 represents results of the Silhouette test and the dendrogram divided into 4 clusters.

The layouts for each cluster are included below.

Summary

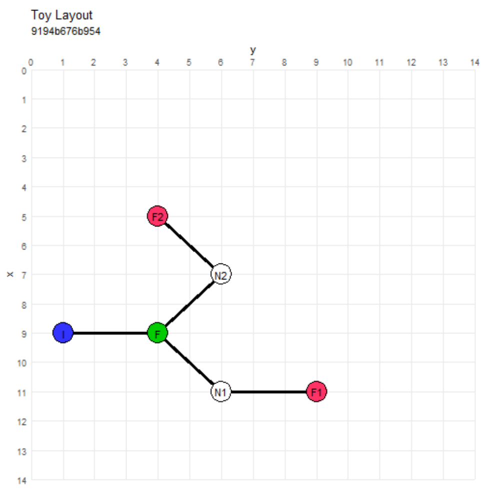
This chapter presented the third case study which was used to test the effective usage of the optimisation-history analysing methodology. The case study was focused at employing optimisation to enhance the design of DNA walker circuits. A DNA walker circuit is a biochemical circuit built using DNA strands. Four different layouts of DNA walker circuits namely Toy, Toy0, Toy1 and Toy2 were used in the example. The methodology presented in Chapter 3 proposed a framework consisting of 3 steps to enhance computational design by using optimisation in the design process and identify alternative solutions by analysing the optimisation history.

The first step is to define a mathematical model which mimics the behaviour of the real system. Professor Gilbert and his research team designed a model for to depict the behaviour of a DNA walker circuit. The second step focuses on applying computational optimisation to enhance the design based on the design requirements. In order to optimise the design of DNA walker circuits mainly two optimisation methods; hill climbing and simulated annealing, were applied. Out of the two optimisation algorithms, simulated annealing was the most suitable method for the task. At the end of the second step an optimisation archive was created which consisted of the history of the optimisation. In a usual optimisation problem, the focus is mainly on the final optimal solution. However, the speciality in the method proposed in this thesis is that is uses the history of the optimisation for further analysis and improve the design process.

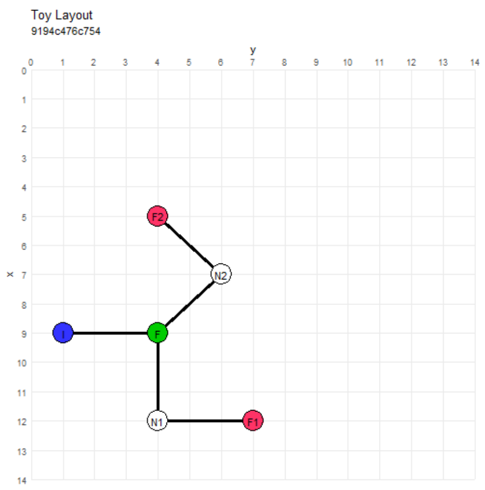
The third step focuses on analysing the optimisation archive. Three main types of insights were extracted from analysing the optimisation archive generated for DNA walker circuits via simulated annealing optimisation. Firstly alternative solutions (behaviourally similar yet structurally different) were extracted by analysing the optimisation archive. These solutions would be helpful when biologists have to make design decisions under constraints. Secondly the optimisation approach was able to successfully employ in confirming design constraints. In this example case study, optimisation history was used to successfully develop an equation to determine the maximum grid size for any given layout. Finally the optimisation archive was used to extra sub design patterns which remained constant in optimal solutions. By observing the evolution of a solution (how does an initial solution change into the optimal solution over time) optimal formations of nodes could be identified. The Toy layout resulted in two main sub designs where the $I, F, N1 and N2$ were formed in a T shape and Y shape.

Alternative solutions of DNA walker circuits identified in this example application are computational design solutions. A simulated annealing optimisation approach was used to identify the optimal way of laying out the nodes in a circuit. However, according to the usual synthetic biology design approach, these designs need to be constructed physically in order to verify the differences between the computationally predicted behaviour and observed physical behaviour. These differences behaviour are fed-back to the computational design process in order to improve the computational design. Therefore alternative computational solutions (design) can be used to drive the engineering of alternative physical designs. In this case the alternative solution of DNA walker circuits acts as an alternative design template (or blueprint) as well.

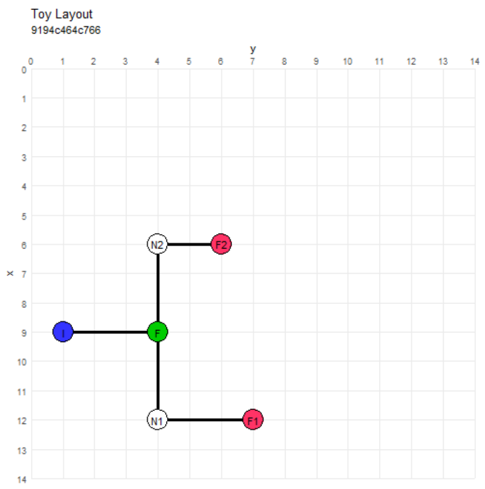
Lastly the chapter discusses about a dissimilarity measure inspired by RMSD minimisation which was developed to compare two DNA circuit layouts. This dissimilarity measure was implemented to identify structural dissimilarities in DNA walker circuit layout and group them. The last section demonstrate the use of the dissimilarity measure to cluster a set of DNA walker circuits for the Toy layout.



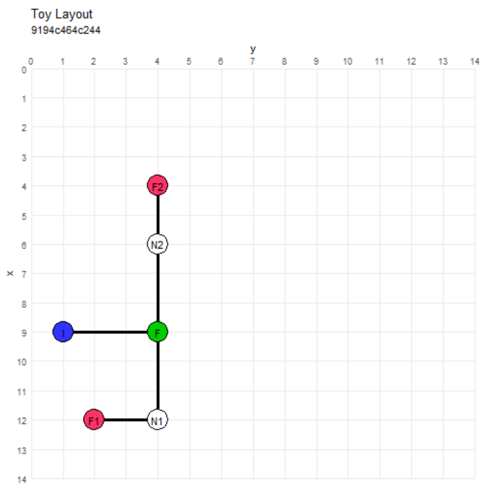
(a) Layout 1: 9194b676b954, Fitness: 1488



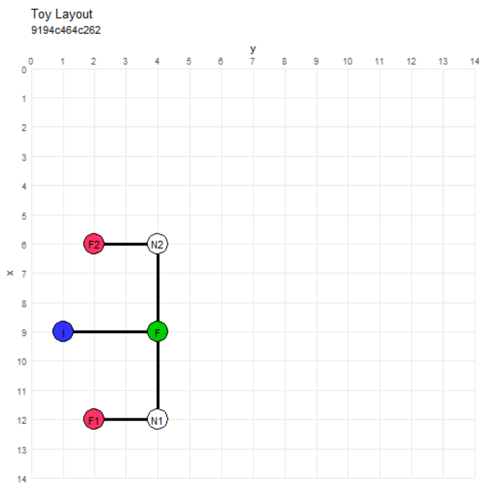
(b) Layout 2: 9194c476c754, Fitness: 168



(c) Layout 3: 9194c464c766, Fitness: 144

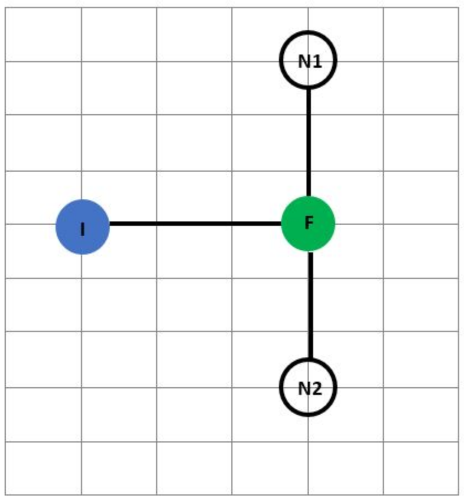


(d) Layout 4: 9194c464c244, Fitness: 96

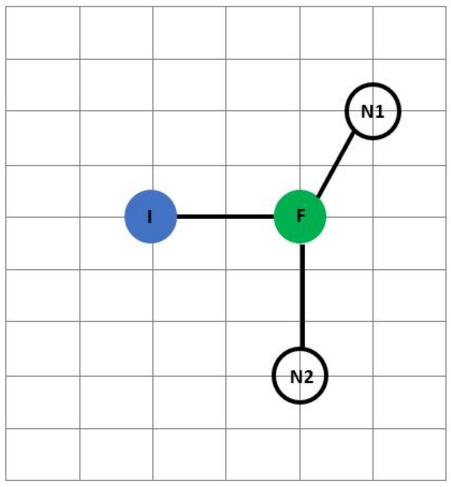


(e) Layout 5: 9194c464c262, Fitness: 72

Figure 6.25: This series of figures represents how simulated annealing algorithm optimised the structure of a Toy layout with an initial fitness of 1488 into a fitness with 72. The I, F, N1 and N2 nodes at the end forms a T shape when the structure becomes optimal relative to number of leaks and circuit area

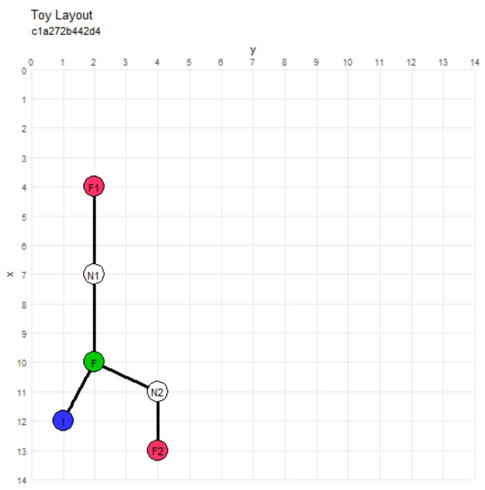


(a) T shape formation of the I, F, N1 and N2 nodes in a Toy layout which provides the optimal sub layout

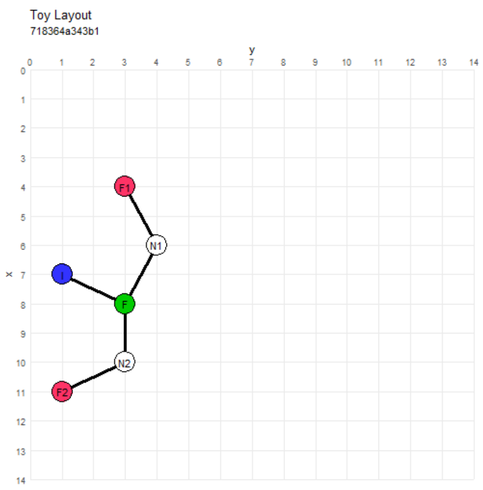


(b) Y shape formation of the I, F, N1 and N2 nodes in a Toy layout which provides the optimal sub layout

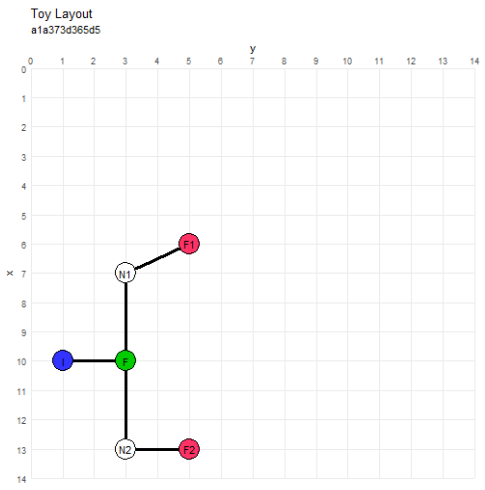
Figure 6.26: The two main sub optimal structures observed in the Toy layout during temporal analysis



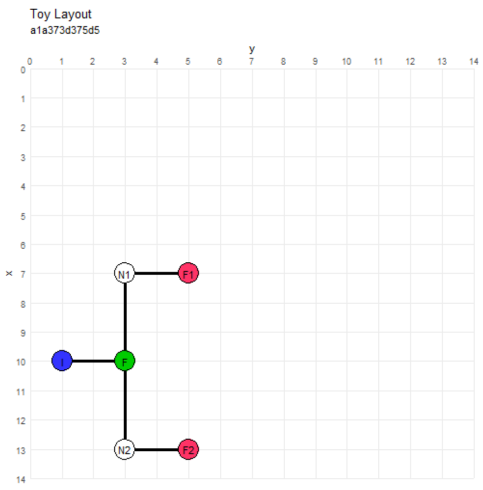
(a) Layout 1: c1a272b442d4, Fitness: 594



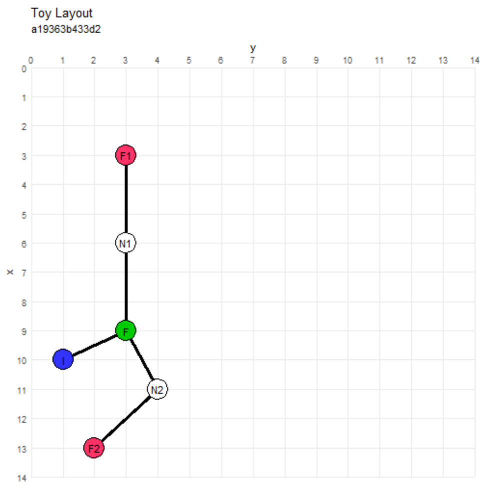
(b) Layout 2: 718364a343b1, Fitness: 462



(c) Layout 3: a1a373d365d5, Fitness: 112



(d) Layout 4: a1a373d375d5, Fitness: 96



(e) Layout 5: a19363b433d2, Fitness: 90

Figure 6.27: This series of figures represents how simulated annealing algorithm optimised the structure of a Toy layout with an initial fitness of 594 into a fitness with 90. The I, F, N1 and N2 nodes at the end forms a Y shape when the structure becomes optimal relative to number of leaks and circuit area

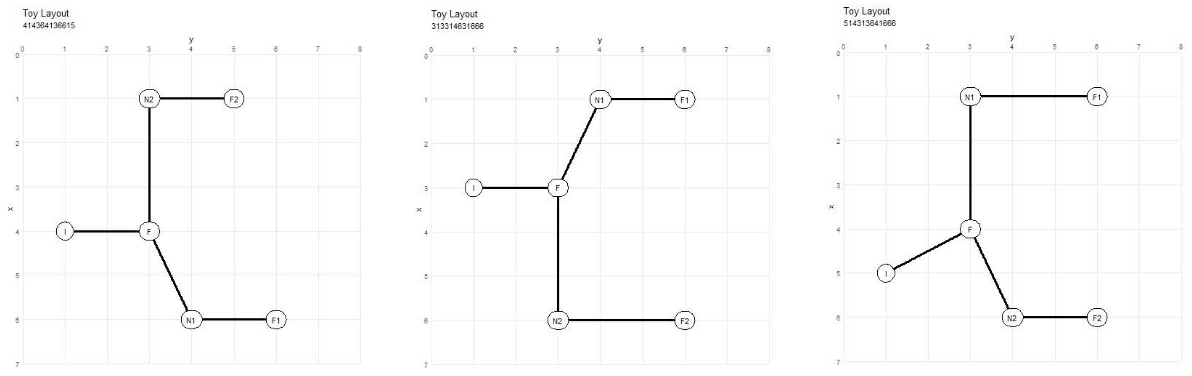


Figure 6.28: Three different layouts with the same number of leaks, area and fitness – from left to right layouts are named as L1, L2, and L3

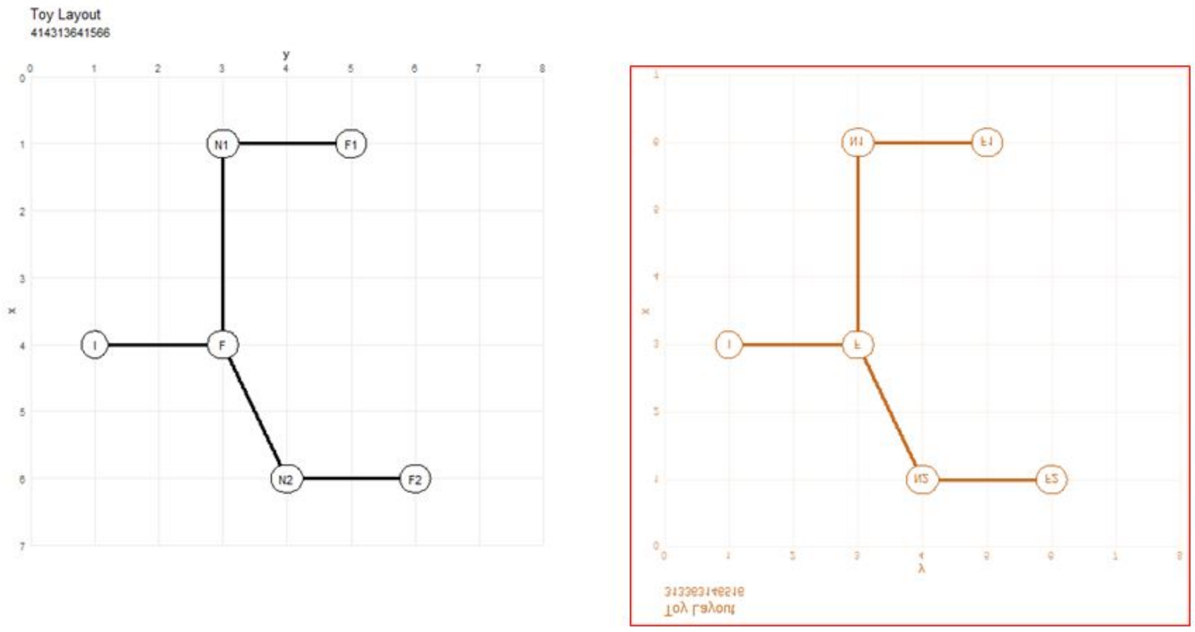
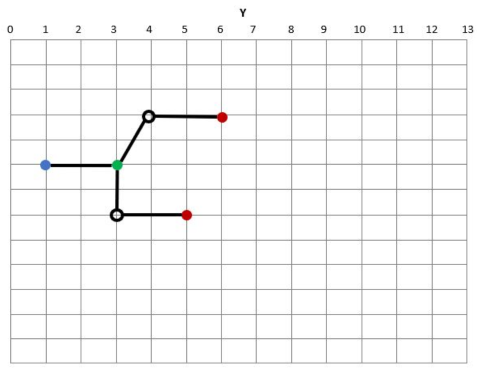
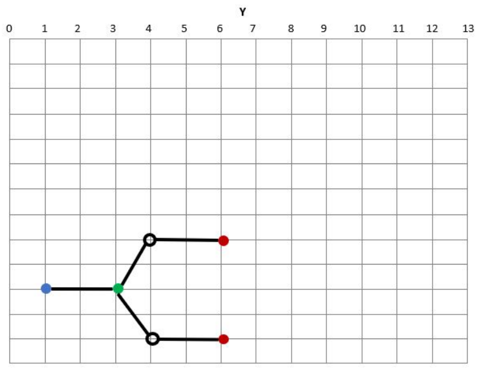


Figure 6.29: Layout 2 is flipped about the FORK node. This makes both Layout 1 and 2 similar

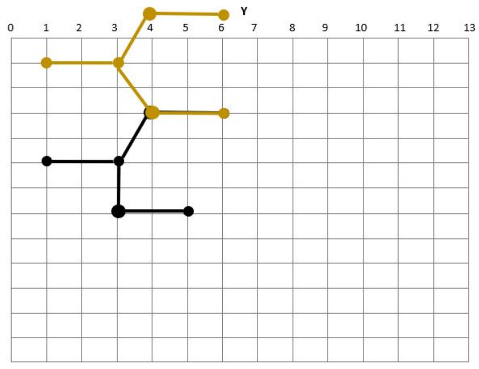


(a) Toy layout 1: I[5,1], F[5,3], N1[3,4], N2[7,3], F1[3,6], F2[7,5]

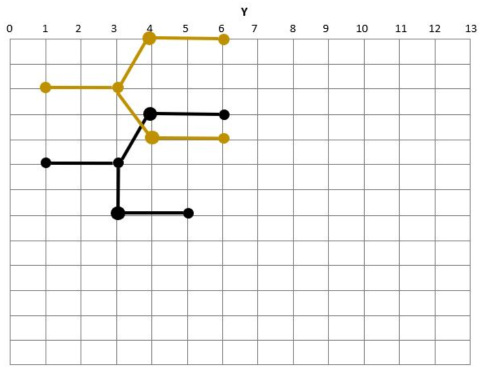


(b) Toy layout 2: I[10,1], F[10,3], N1[8,4], N2[12,4], F1[8,6], F2[12,6]

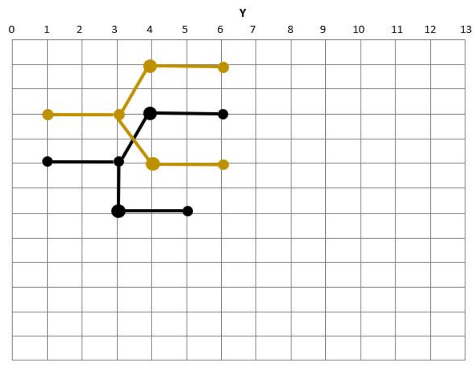
Figure 6.30: Example Toy layouts



(a) Initial placement of the second layout on the grid. I is placed on $[1,1]$ position. This is not a valid positioning according to the DNA circuit layout constraints.

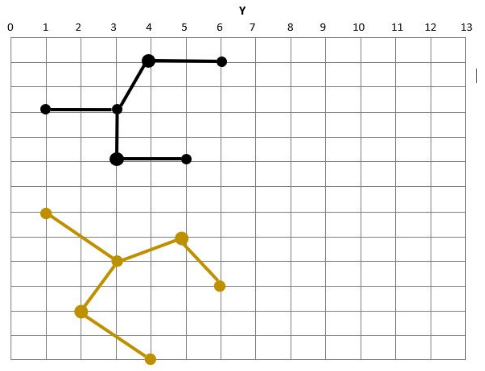


(b) I of the second layout is moved down along the $Y = 1$ axis by 1. I is placed on $[2,1]$ position. This is not a valid positioning according to the DNA circuit layout constraints.

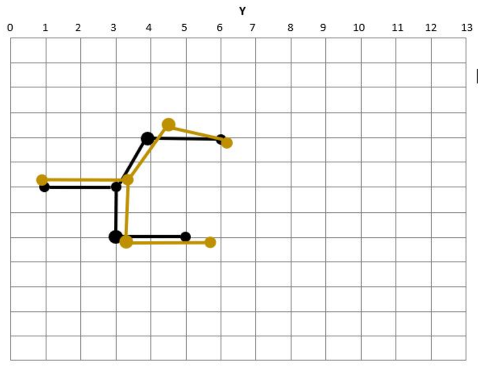


(c) I of the second layout is moved down along the $Y = 1$ axis by 1. I is placed on $[3,1]$ position. This forms a valid positioning according to the DNA circuit layout constraints.

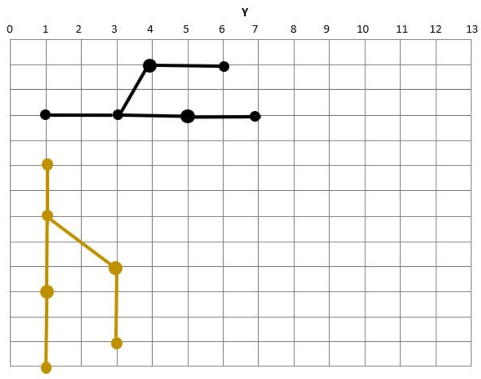
Figure 6.31: Placement of the two layouts on the same grid. Layout 1 is represented in back and layout 2 is represented in yellow colour. The second layout is moved along the $Y = 1$ axis.



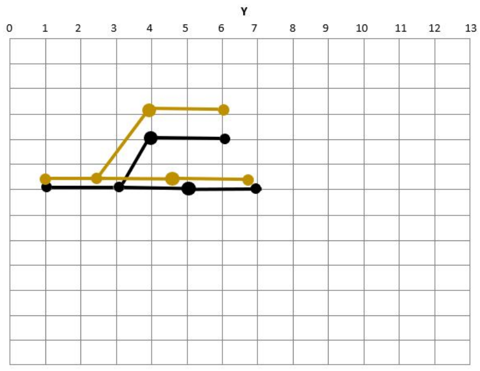
(a) Two layouts with rotational symmetry of 45 degrees



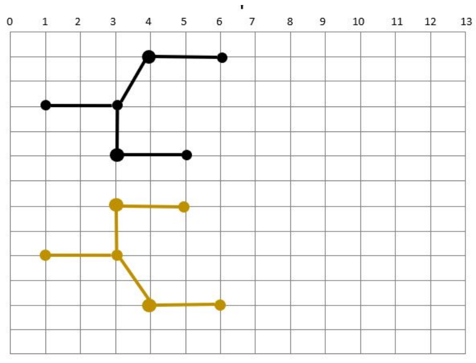
(b) Layout 2 is rotated 45 in order for the $I - F$ to align with each other



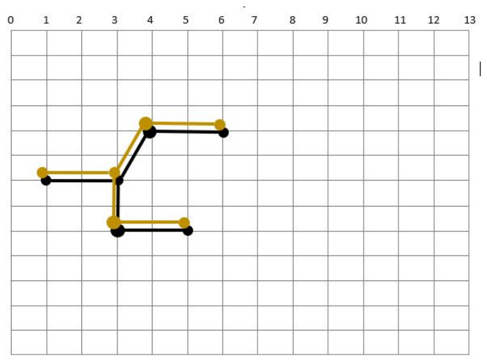
(c) Two layouts with rotational symmetry of 90 degrees



(d) Layout 2 is rotated 90 in order for the $I - F$ to align with each other



(e) Two layouts with rotational symmetry of 180 degrees



(f) Layout 2 is rotated 180 degrees about the F node to align with each other

Figure 6.32: Examples of rotational symmetry in the comparing Toy layouts

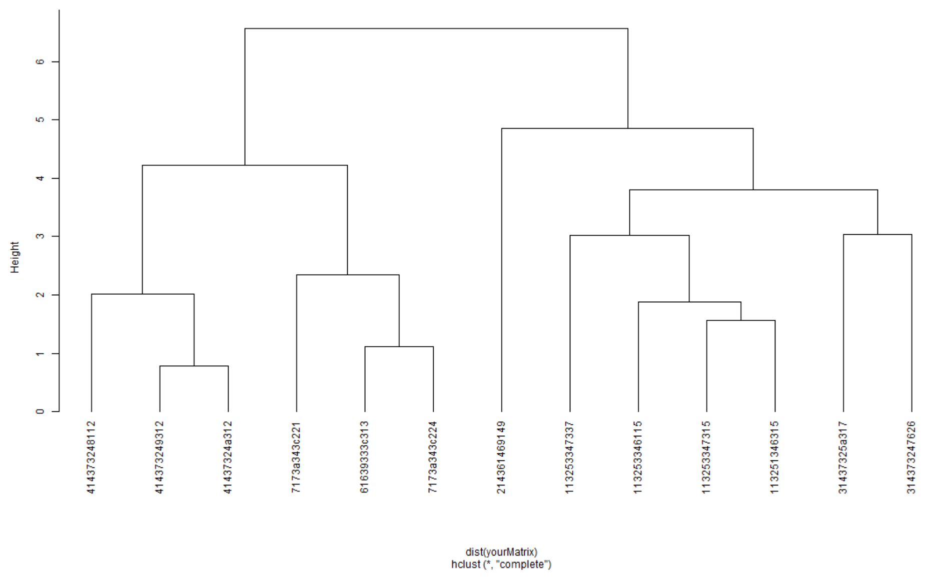
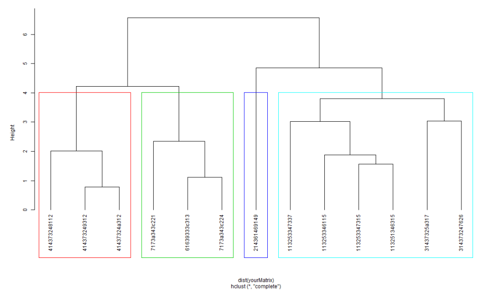
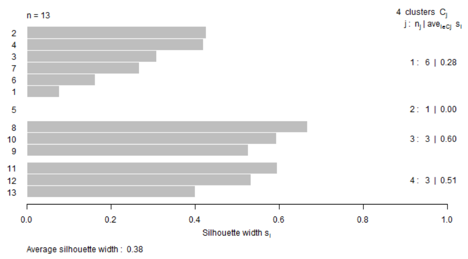


Figure 6.33: Clustering of the Toy layout using RMSD pair wise distance

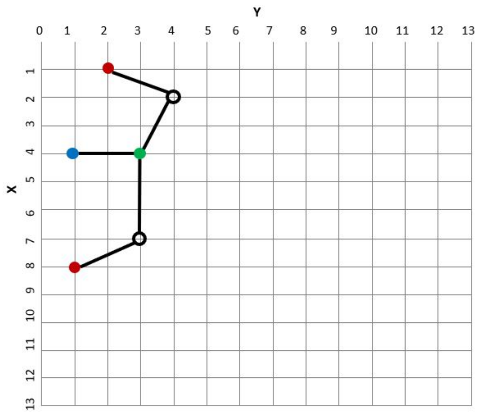




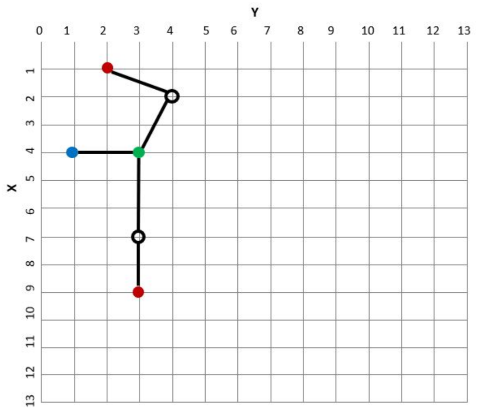
(a) The dendraogram has been divided into 4 clusters based on the Silhouette value

(b) The best Silhouette value achieved was 0.38 for 4 clusters. Hence the optimal number of clusters was taken as 4

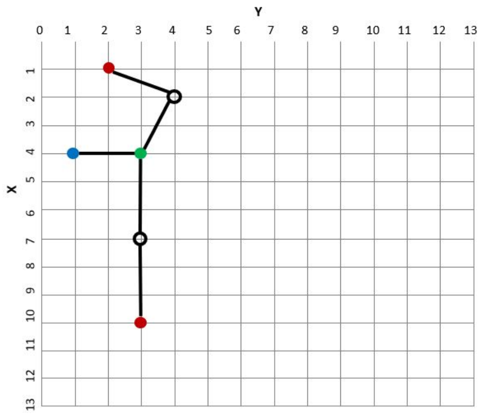
Figure 6.34: Evaluation of the goodness of clusters using Silhouette measure



(a) Layout 1: 414373248112

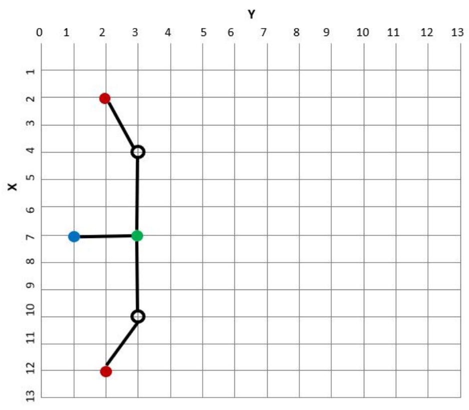


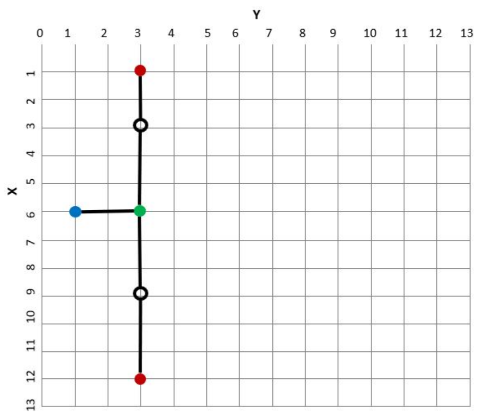
(b) Layout 2: 414373249312



(c) Layout 3: 41437324a312

Figure 6.35: Elements in Cluster No 1 is a result of clustering of the 13 Toy solutions based on their structural similarity using RMSD pairwise distance resulted in 4 clusters (see figure 6.34). As you can see the solutions in the cluster a similar in structure.


(a) Layout 1: 7173a343c221


(b) Layout 2: 61639333c313

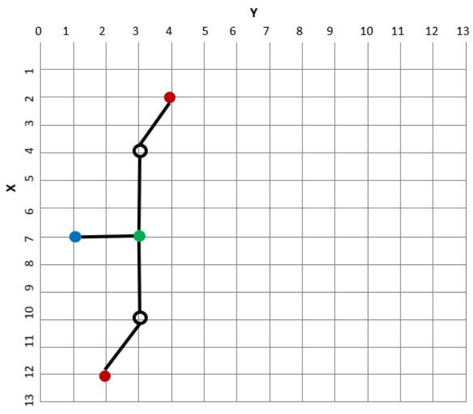

(c) Layout 3: 7173a343c224

Figure 6.36: Elements in Cluster No 2 is a result of clustering of the 13 Toy solutions based on their structural similarity using RMSD pairwise distance resulted in 4 clusters (see figure 6.34). As you can see the solutions in the cluster a similar in structure.

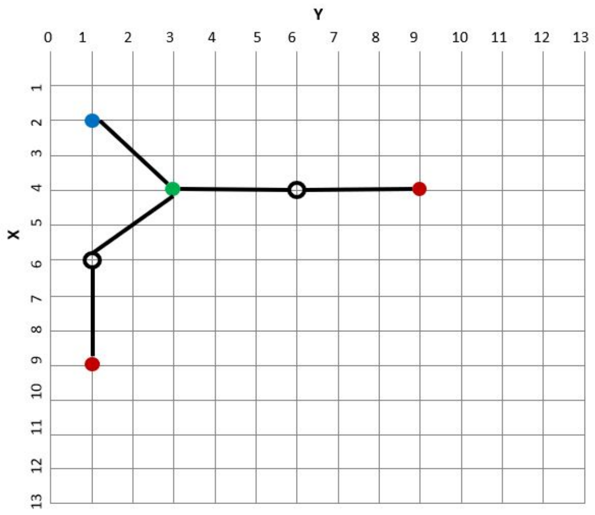
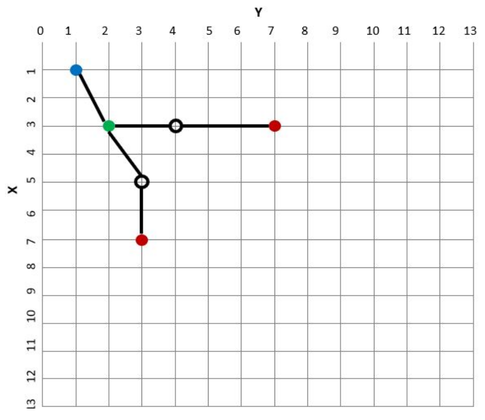
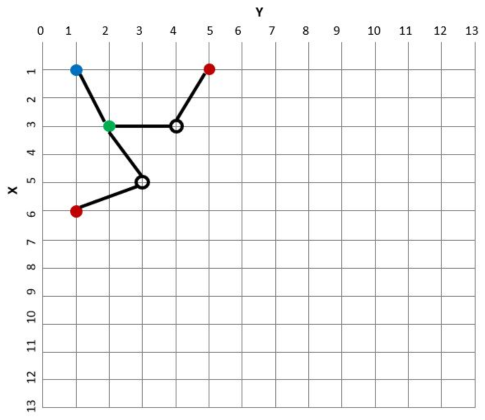


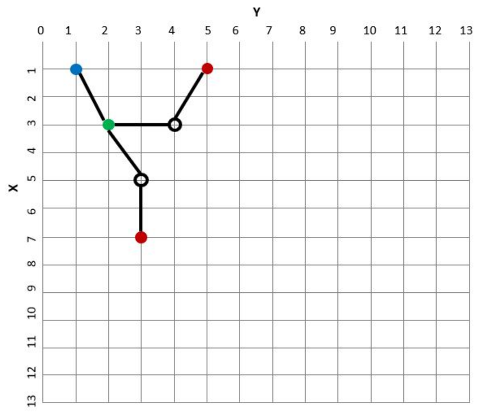
Figure 6.37: Layout 1: 214361469149: Elements in Cluster No 3 is a result of clustering of the 13 Toy solutions based on their structural similarity using RMSD pairwise distance resulted in 4 clusters (see figure 6.34). As you can see the solutions in the cluster a similar in structure.



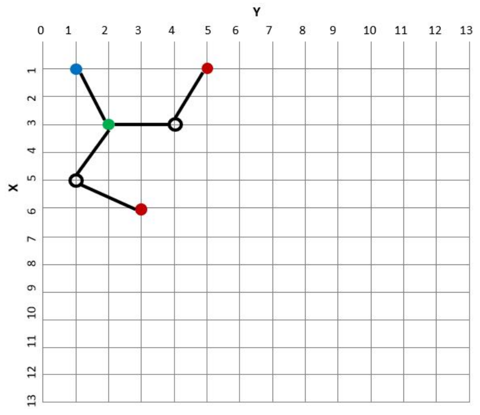
(a) Layout 1: 113253347337



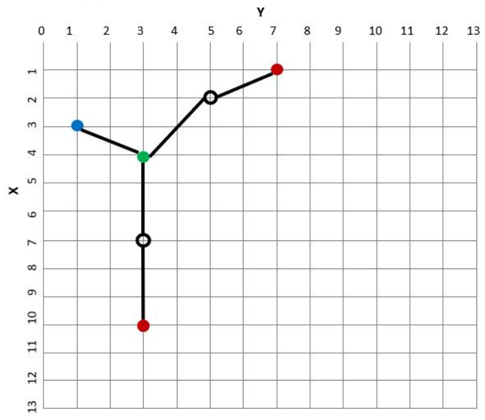
(b) Layout 2: 113253346115



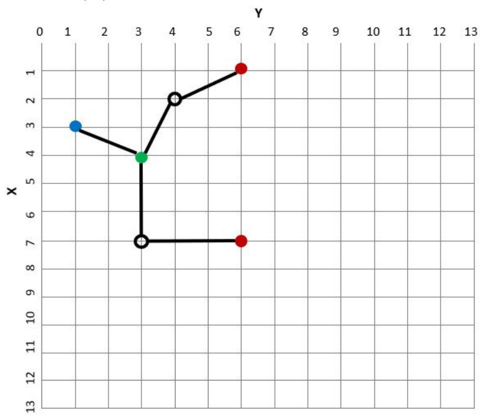
(c) Layout 3: 113253347315



(d) Layout 4: 113251346315



(e) Layout 5: 31437325a317



(f) Layout 6: 314373247626

Figure 6.38: Elements in Cluster No 4 is a result of clustering of the 13 Toy solutions based on their structural similarity using RMSD pairwise distance resulted in 4 clusters (see figure 6.34). As you can see the solutions in the cluster a similar in structure.

Chapter 7

Summary, Conclusions and Further Work

Introduction

This chapter revisits the main contributions of the thesis. Corresponding open problems that could be addressed in the future are described in the end.

7.1 Summary

Computational modelling is an effective method in tuning the performance of complex systems. Further, computational optimisation plays a significant role in creating 'better' designs by experimenting on endless permutations and combinations of initial conditions.

However, in most cases there is a gap between approximations delivered by optimization and their practical realization. The main reason for these discrepancies is the degree of complexity in the real-world complex systems. As the complexity of a system increases, the number of variables which determine the behaviour of a complex system increases as well. Subsequently, the non-linear relationships between these variables and partial understanding of these relationships attribute to inconsistencies in computational model and actual system. Further, limitations in computational resources used for simulating computational models restrict modellers to modelling sub-systems (larger the model more computational resources are required for simulations). Therefore, when pugging in sub systems to form a complete system might result in certain inconsistencies. Hence, there are limitations in capturing the exact behaviour of a complex system through optimization tools.

As a result, the optimal design produced by the computational optimisation approach in certain instances will have a higher in-vitro implementation cost (i.e. too many modifications, impossible alterations, highly time and resource consuming alterations). These implementation impracticalities result in a less robust optimal design. In such cases, a more practical approach would be to settle for solutions with sub-optimal behaviour with a reasonable accuracy and resource expenditure. Hence the accessibility to alternative solutions with optimal (similar behaviour and different structures) or suboptimal behaviour play a vital role in the design process. The importance of having alternative solutions can be better explained through a common example such as timetabling. When preparing a timetable various factors are taken into consideration such as the subject being taught, type of class room, number of students, frequency of the activity, class of students and etc. Due to the contradicting nature of constraints, often multiple timetables are generated for the same cohort. Availability of multiple (or alternative) solutions gives the freedom to the persons who make decisions to select the most viable solution. The most viable solution might not always be the most optimal solution. Similarly, in synthetic biology, availability of alternative design solutions gives the designers the opportunity to select the most cost-effective, easy and fast to implement design instead of an optimal solution which is impossible to implement.

Therefore, the work presented in this thesis was focused on proposing a methodology (Chapter 3) to analyse a population of solutions generated by an optimisation program and extract alternative solutions. The generalizability of the methodology has been demonstrated by applying the methodology to three different problem areas based on three different data types (Chapter 4, 5 and 6).

7.1.1 Methodology

In this thesis a general methodology was proposed which can be applied to extract alternative designs of computational models by analysing the optimisation history obtained from a population based heuristic search.

The methodology consists of three main phases; mathematical model definition, target driven optimisation and analysis of optimisation archive. Defining of a mathematical model which depicts the whole/ partial behaviour of a complex system is carried out in the first phase. This phase can be initiated with an existing definition of an model as well. In such instances, the behaviour to be optimised is defined in the first phase.

In the second phase computational optimisation is applied to generate the most optimal model for the pre-defined targets. The methodology does not define a specific optimisation algorithm as the applicability of an algorithm depends on the characteristics of the model. The optimisation archive is created in the second phase. Optimisation archive stores the solution space explored by an optimisation algorithm in the process of reaching the optimal solution. These solutions are stored in a sequential order as they appear in the search.

The third phase is focused on analysing the optimisation archive to extract alternative solutions. In addition to the main objective insights from analysing archive data can be used to improve model design through evolutionary analysis of structures and standardising model constraints. A range of data analytic techniques are used to extract alternative solutions by grouping the models by behaviour and structure. These techniques are depended on the characteristics of models being analysed.

7.1.2 Applications of the methodology

The methodology was applied to three examples scenarios in order to demonstrate its generality in application. The three examples focus on three different data categories and optimisation techniques.

The first example was focused on identifying alternative mathematical expressions which evaluates to a specified target and could only be compiled with a pre-defined set of operands and operators. Gene Expression Programming (GEP) optimisation algorithm was used to obtain the optimal solution. Mathematical expressions were interpreted as binary trees for the purpose of comparison. A tree edit distance was used to compare between structures and similar structures were grouped using hierarchical clustering (see Chapter 4).

The second application area was identifying alternative solutions in a population of solutions of Genomic Metabolic Models (GEM) models of bacteria. The optimisation archive was consisted of solutions generated from genetic algorithm optimisation technique. The data category analysed in this example was static data. Characteristics of models such as gene and reaction composition were used to compute similarity between solutions. Hierarchical clustering and visual analytic techniques were used to group solutions and identify alternative designs (see Chapter 5).

Lastly, the methodology was applied to a population of solutions consisting of DNA Walker circuits. Simulated annealing optimisation technique was used to explore the solution space to find the optimal circuit layout with the least number of leaks and area. The optimisation archive was composed of the solutions explored during the search. DNA circuits were represented as graphs. Therefore, the analysis was performed on graph data category. A new comparison algorithm which was developed to structurally compare circuit layouts and the similarity measure was used to hierarchically cluster the solutions in order to identify alternative solutions. Further, descriptive analytic techniques were also used. In addition to extracting alternative solutions, visual analytic techniques were applied to observe the evolution of an initial solution into an optimal solution. Identified optimal sub structures was used to improve model design. Further, this example demonstrated the application of the methodology to standardise model constraints (see Chapter 6).

The application areas covered in this thesis are mainly from synthetic biology. As explained in Chapter 1 (refer Figure 1.2), computational models that depict the desired behaviour need to be physically constructed in order to verify its behaviour and the computational approach which was used to generate the design. This information is then used to improve the computational design process. Therefore alternative computational solutions (design) can be used to drive the engineering of alternative physical designs. In this case the [alternative] solution acts as an [alternative] design template (or blueprint). Therefore, alternative solutions presented for GEM models and DNA circuits act as blueprints as well.

7.2 Conclusions

Computational optimisation is often used in the design process to select the best solution from a set of available solutions. However, in most cases there is a gap between approximations delivered by optimization and their practical realization. In an instance where the optimal solution selected by an optimisation algorithm has a significantly high implementation cost it would be ideal to opt for a sub optimal solution with a lower implementation cost.

An optimisation algorithm would always look for the next best solution by maximizing or minimizing an objective function until a pre-defined target value is reached or a fixed number search iterations have been completed. Therefore the search history of an optimisation algorithm contains solutions which are sub optimal. However, due to the nature of an optimisation algorithm of searching for the best solution, often the history of the search is discarded.

Hence, the methodology proposed in this thesis was aimed at analysing the search history of an optimisation search process. The main advantage of the methodology is that, it utilises intermediary results from an existing search process aiming at selecting the best solution. Therefore, it eliminates the need to generate new data for the sole purpose of extracting alternative solutions. For example, the optimiser generates around 500 - 600 unique solutions within a single run of simulated annealing with 30,000 iterations. As the aim of an optimisation approach is to select the best solution only one gets picked at the end. The remaining 599 solutions are usually discarded. Hence, the methodology is utilizing an untapped source of data to extract alternative solutions that have affordable implementation costs, and to improve model design by observing the evolution of an solution from the initial to optimal. Further, this method facilitates the concept of improving initial model design and improving time spent on heuristic search by limiting the degree of variability. This feature is achieved through extracting constant core sub patterns through analysing the evolution of initial solution to the optimal solution. For example when optimising the DNA walker circuit layouts with SA optimisation, the program was required to randomly pick positions for all nodes whenever a solution was created. If there were 6 nodes the program had to randomly guess coordinates for all 6 nodes. However, if we could place certain nodes in constant positions we could reduce the number of guesses when creating a new solution and cut down execution time. In order to identify which nodes can be placed in constant places so that the fitness would improved, is done through temporal analysis. By observing how a solution evolves over time we can identify which parts of the layout that remains same for a better fitness. These parts in a layout that doesn't change are referred to as constant core sub patterns. As the methodology is applied to intermediary results of an optimisation process, it is not required to wait till the search process is completed to analyse the solutions. This is an advantage when a heuristic search process takes a long time to select the optimal solution because implementation can begin on a partially optimal solution or design could be improved with the evolutionary insights obtained from the analysis and restart the search process with a better search position.

Another advantage is the generality of the methodology. The methodology can be ideally used for any population of solutions generated from a heuristic search process and it is independent of the data category of the solutions. This has been demonstrated by three different applications which is described in chapter 4, 5 and 6. As summarised above the methodology was applied to three different heuristic search approaches namely; gene expression programming, genetic algorithms and simulated annealing, which focused on three different data categories namely; binary trees, static data and graphs respectively.

7.3 Further Work

The methodology can be extended in the future to address the following open problems.

7.3.1 Apply particle swarm optimisation to optimise DNA walker circuit layouts

Two main optimisation algorithms; hill climbing and simulated annealing, were used to optimise the design of DNA walker circuits. Simulated annealing algorithm performed better than the hill climbing algorithm. However, one of the main short comings of the simulated annealing algorithm is that the solution space consists of more repeated solutions. The number of unique solutions is far less and this cause an inefficient search in the solution space. Therefore I intend to apply particle swarm optimisation (PSO) algorithm (Liu et al. 2018) to improve the design process of DNA walker circuits. This study will be done in collaboration with a another research group at Brunel.

7.3.2 Test the models in-vitro environment

The optimal and sub optimal designs obtained for GEM models and DNA walker circuits in the work presented in this thesis are based on computational accuracy. The behaviour simulations were performed on computational models. If the in-silico models could be tested in-vitro the output can be used to further improve the design process.

7.3.3 Automatically identify sub patterns

For DNA walker circuits the evolution of an initial solution into the optimal solution is observed through visually. This task was performed manually by observing the pattern change through image transition. It could be ideal to implement a software which could analyse the evolution of structures and extract the static core sub designs.

7.3.4 Validation of the methodology from other domains of science

The methodological approach to identify alternative solutions from a computational optimisation program was mainly applied to example applications from synthetic biology domain. However, there are other domains where computational optimisation is used for designing purposes. For example, truss optimisation widely used in architecture (Gomes 2011), optimisation of semi-conductor circuit layouts (Gaston and Walton 1994), mechanical design optimisation (Rao and

Savsani 2012) and warehouse layout optimisation (Karásek 2013) is a few to name from the engineering domain. The methodology can be potentially applied to optimisation approaches with external constraints to test the robustness of the methodology.

References

Allhoff, Fritz, Patrick Lin, and Daniel Moore (2010). "What is nanotechnology and why does it matter". In: *From Science to Ethics* 30, p. 00.

Allwright, James RA and DB Carpenter (1989). "A distributed implementation of simulated annealing for the travelling salesman problem". In: *Parallel Computing* 10.3, pp. 335–338.

Andrianantoandro, Ernesto et al. (2006). "Synthetic biology: new engineering rules for an emerging discipline". In: *Molecular systems biology* 2.1, pp. 2006–0028.

Antoniou, Andreas and Wu-Sheng Lu (2007). *Practical optimization: algorithms and engineering applications*. Springer Science & Business Media.

Arriaga, Jonathan and Manuel Valenzuela-Rendón (2012). "Steepest ascent hill climbing for portfolio selection". In: *European Conference on the Applications of Evolutionary Computation*. Springer, pp. 145–154.

Bandaru, Sunith and Kalyanmoy Deb (2011). "Towards automating the discovery of certain innovative design principles through a clustering-based optimization technique". In: *Engineering optimization* 43.9, pp. 911–941.

— (2015). "Temporal innovization: Evolution of design principles using multi-objective optimization". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, pp. 79–93.

Benner, Steven A and A Michael Sismour (2005). "Synthetic biology". In: *Nature Reviews Genetics* 6.7, pp. 533–543.

Benner, Steven A, Zunyi Yang, and Fei Chen (2011). "Synthetic biology, tinkering biology, and artificial biology. What are we learning?" In: *Comptes Rendus Chimie* 14.4, pp. 372–387.

Berkens, Martinus Maria et al. (2012). *Method for optimizing an integrated circuit physical layout*. US Patent 8,151,234.

Bradner, Erin, Francesco Iorio, Mark Davis, et al. (2014). “Parameters tell the design story: ideation and abstraction in design optimization”. In: *Proceedings of the symposium on simulation for architecture & urban design*. Vol. 26.

Brooks Jr, Frederick P (2010). *The design of design: Essays from a computer scientist*. Pearson Education.

Cameotra, SS and RS Makkar (1998). “Synthesis of biosurfactants in extreme conditions”. In: *Applied Microbiology and Biotechnology* 50.5, pp. 520–529.

Collins, Francis S et al. (1998). “New goals for the US human genome project: 1998-2003”. In: *Science* 282.5389, pp. 682–689.

Dannenberg, Frits et al. (2013). “DNA walker circuits: Computational potential, design, and verification”. In: *International Workshop on DNA-Based Computers*. Springer, pp. 31–45.

Dannenberg, Frits Gerrit Willem (2016). “Modelling and verification for DNA nanotechnology”. PhD thesis. University of Oxford.

Deb, Kalyanmoy, Sunith Bandaru, and Cem Celal Tutum (2012). “Temporal evolution of design principles in engineering systems: Analogies with human evolution”. In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 1–10.

Deb, Kalyanmoy and Aravind Srinivasan (2006). “Innovization: Innovating design principles through optimization”. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 1629–1636.

Díaz, Josep, Jordi Petit, and Maria Serna (2002). “A survey of graph layout problems”. In: *ACM Computing Surveys (CSUR)* 34.3, pp. 313–356.

Dorst, Kees and Judith Dijkhuis (1995). “Comparing paradigms for describing design activity”. In: *Design Studies* 16.2, pp. 261–274.

Edwards, Jeremy S and Bernhard O Palsson (2000). “The Escherichia coli MG1655 in silico metabolic genotype: its definition, characteristics, and capabilities”. In: *Proceedings of the National Academy of Sciences* 97.10, pp. 5528–5533.

Ester, Martin et al. (1996). “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Vol. 96. 34, pp. 226–231.

Feist, Adam M et al. (2007). “A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information”. In: *Molecular systems biology* 3.1, p. 121.

Ferreira, C (2010). *What is gep? from genexprotools tutorials-a gepsoft web resource*.

Ferreira, Candida (2001). “Gene expression programming: a new adaptive algorithm for solving problems”. In: *arXiv preprint cs/0102027*.

Ferreira, Cândida (2006). “Automatically defined functions in gene expression programming”. In: *Genetic systems programming*. Springer, pp. 21–56.

Fuentes, Paulina et al. (2016). “A new synthetic biology approach allows transfer of an entire metabolic pathway from a medicinal plant to a biomass crop”. In: *Elife* 5, e13664.

Gaston, Gdfrey J and Anthony J Walton (1994). “The integration of simulation and response surface methodology for the optimization of IC processes”. In: *IEEE Transactions on Semiconductor Manufacturing* 7.1, pp. 22–33.

Gilbert, David, Monika Heiner, and Christian Rohr (2018). “Petri-net-based 2D design of DNA walker circuits”. In: *Natural Computing* 17.1, pp. 161–182.

Gilbert, David et al. (2019). “Towards dynamic genome-scale models”. In: *Briefings in bioinformatics* 20.4, pp. 1167–1180.

Gomes, Herbert Martins (2011). “Truss optimization with dynamic constraints using a particle swarm algorithm”. In: *Expert Systems with Applications* 38.1, pp. 957–968.

Goodsell, David S and Arthur J Olson (1990). “Automated docking of substrates to proteins by simulated annealing”. In: *Proteins: Structure, Function, and Bioinformatics* 8.3, pp. 195–202.

Guess, Michael J and Scott B Wilson (2002). “Introduction to hierarchical clustering”. In: *Journal of clinical neurophysiology* 19.2, pp. 144–151.

Harper, Douglas (2015). “Online etymology dictionary. 2001”. In: *Availabe from: www. etymonline. com/index. php*.

Hasançebi, Oğuzhan and Fuat Erbatur (2002). “Layout optimisation of trusses using simulated annealing”. In: *Advances in Engineering Software* 33.7-10, pp. 681–696.

Heiner, Monika, David Gilbert, and Robin Donaldson (2008). “Petri nets for systems and synthetic biology”. In: *International school on formal methods for the design of computer, communication and software systems*. Springer, pp. 215–264.

Heiner, Monika, Christian Rohr, and Martin Schwarick (2013). “"MARCIE–model checking and reachability analysis done efficiently"”. In: *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, pp. 389–399.

Heiner, Monika et al. (2012). “Snoopy–a unifying Petri net tool”. In: *International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, pp. 398–407.

Hosseini-Nasab, Hasan et al. (2018). “Classification of facility layout problems: a review study”. In: *The International Journal of Advanced Manufacturing Technology* 94.1-4, pp. 957–977.

Huang, Wenlan, Yu Zhang, and Lan Li (2019). “Survey on multi-objective evolutionary algorithms”. In: *Journal of Physics: Conference Series*. Vol. 1288. 1. IOP Publishing, p. 012057.

Hucka, Michael et al. (2003). “The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models”. In: *Bioinformatics* 19.4, pp. 524–531.

Hughes, Peter William, Shannon Vance Morton, and Trevor Kenneth Monk (2007). *Method for optimising transistor performance in integrated circuits*. US Patent 7,266,787.

Jiang, Wenzhi et al. (2013). “Demonstration of CRISPR/Cas9/sgRNA-mediated targeted gene modification in Arabidopsis, tobacco, sorghum and rice”. In: *Nucleic acids research* 41.20, e188–e188.

Karásek, Jan (2013). “An overview of warehouse optimization”. In: *International journal of advances in telecommunications, electrotechnics, signals and systems* 2.3, pp. 111–117.

Kawamata, Ibuki, Fumiaki Tanaka, and Masami Hagiya (2009). “Automatic design of DNA logic gates based on kinetic simulation”. In: *International Workshop on DNA-Based Computers*. Springer, pp. 88–96.

Kirkpatrick, Scott, C Daniel Gelatt, and Mario P Vecchi (1983). “Optimization by simulated annealing”. In: *science* 220.4598, pp. 671–680.

Kitano, Hiroaki (2002). “Systems biology: a brief overview”. In: *science* 295.5560, pp. 1662–1664.

Koza, John R (1995). “Survey of genetic algorithms and genetic programming”. In: *Wescon conference record*. WESTERN PERIODICALS COMPANY, pp. 589–594.

Koza, John R et al. (1996). “Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming”. In: *Proceedings of the First Annual Conference on Genetic Programming*. Stanford University MIT Press, Cambridge, MA, pp. 132–140.

Lewis, Kemper E, Wei Chen, and Linda C Schmidt (2006). *Decision making in engineering design*. ASME Press New York.

Li, Ting et al. (2012). “High-efficiency TALEN-based gene editing produces disease-resistant rice”. In: *Nature biotechnology* 30.5, p. 390.

Liao, T Warren (2005). “Clustering of time series data—a survey”. In: *Pattern recognition* 38.11, pp. 1857–1874.

Liu, Weibo et al. (2018). “A novel particle swarm optimization approach for patient clustering from emergency departments”. In: *IEEE Transactions on Evolutionary Computation* 23.4, pp. 632–644.

Lu, Jason Jason and Minlu Zhang (2013). “Heuristic Search”. In: *Encyclopedia of Systems Biology*. Ed. by Werner Dubitzky et al. New York, NY: Springer New York, pp. 885–886. ISBN: 978-1-4419-9863-7. DOI: `10.1007/978-1-4419-9863-7\_875`. URL: `https://doi.org/10.1007/978-1-4419-9863-7\_875`.

Mackenzie, Adrian (June 2010). “Design in synthetic biology”. In: *BioSocieties* 5.2, pp. 180–198. ISSN: 1745-8552. DOI: `10.1057/biosoc.2010.4`.

Messac, Achille and Christopher A Mattson (2004). “Normal constraint method with guarantee of even representation of complete Pareto frontier”. In: *AIAA journal* 42.10, pp. 2101–2111.

Miao, Hui and Yu-Chu Tian (2008). “Robot path planning in dynamic environments using a simulated annealing based approach”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE, pp. 1253–1258.

Monk, Jonathan M et al. (2013). “Genome-scale metabolic reconstructions of multiple Escherichia coli strains highlight strain-specific adaptations to nutritional environments”. In: *Proceedings of the National Academy of Sciences* 110.50, pp. 20338–20343.

Nielsen, Jens (2001). “Metabolic engineering”. In: *Applied microbiology and biotechnology* 55.3, pp. 263–283.

Niidome, T and L Huang (2002). “Gene therapy progress and prospects: nonviral vectors”. In: *Gene therapy* 9.24, pp. 1647–1652.

Orth, Jeffrey D, Ines Thiele, and Bernhard Ø Palsson (2010). “What is flux balance analysis?” In: *Nature biotechnology* 28.3, pp. 245–248.

Oxford Dictionary, Oxford English (2004). “Oxford English dictionary online”. In: *Mount Royal College Lib., Calgary* 14.

Oxman, Rikva (2006). “Theory and design in the first digital age Design Studies”. In: *Faculty of Architecture and Town Planning Technion, Haifa* 32000.

Paddon, Chris J and Jay D Keasling (2014). “Semi-synthetic artemisinin: a model for the use of synthetic biology in pharmaceutical development”. In: *Nature Reviews Microbiology* 12.5, pp. 355–367.

Parejo, José Antonio et al. (2012). “Metaheuristic optimization frameworks: a survey and benchmarking”. In: *Soft Computing* 16.3, pp. 527–561.

Park, Hae-Sang, Jong-Seok Lee, and Chi-Hyuck Jun (2006). “A K-means-like Algorithm for K-medoids Clustering and Its Performance”. In: *Proceedings of ICCIE*, pp. 102–117.

Paynter, Henry M (1961). *Analysis and design of engineering systems*. MIT press.

Porter, Alan L and Jan Youtie (2009). “How interdisciplinary is nanotechnology?” In: *Journal of nanoparticle research* 11.5, pp. 1023–1041.

Pressman, Roger S (2005). *Software engineering: a practitioner’s approach.* Palgrave Macmillan.

Primrose, Sandy B and Richard Twyman (2013). *Principles of gene manipulation and genomics.* John Wiley & Sons.

Qian, Lulu and Erik Winfree (2011). “Scaling up digital circuit computation with DNA strand displacement cascades”. In: *science* 332.6034, pp. 1196–1201.

Qing, Anyong (2006). *Differential evolution: Fundamentals and Applications in Electrical Engineering.* Wiley-Blackwell.

Ralph, Paul and Yair Wand (2009). “A proposal for a formal definition of the design concept”. In: *Design requirements engineering: A ten-year perspective.* Springer, pp. 103–136.

Raman, Dhamodharan, Sev V Nagalingam, and Bruce W Gurd (2009). “A genetic algorithm and queuing theory based methodology for facilities layout problem”. In: *International Journal of Production Research* 47.20, pp. 5611–5635.

Rao, R Venkata and Vimal J Savsani (2012). “Mechanical design optimization using advanced optimization techniques”. In.

Robl, James M (2002). *Application of Transgenic Technology in Animal Agriculture.*

Ross, Peter and Dave Corne (1995). “Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems”. In: *AISB workshop on evolutionary computing.* Springer, pp. 94–102.

Rossetti, Antonio, Alarico Macor, and Martina Scamperle (2017). “Optimization of components and layouts of hydromechanical transmissions”. In: *International Journal of Fluid Power* 18.2, pp. 123–134.

Sadeghi, Ali et al. (2013). “Metrics for measuring distances in configuration spaces”. In: *The Journal of chemical physics* 139.18, p. 184118.

Selman, Bart and Carla P Gomes (2006). “Hill-climbing Search”. In: *Encyclopedia of Cognitive Science.*

Snyder, Larry et al. (2003). *Molecular genetics of bacteria.* American Society of Microbiology.

Tayal, Akash and Surya Prakash Singh (2018). “Integrating big data analytic and hybrid firefly-chaotic simulated annealing approach for facility layout problem”. In: *Annals of Operations Research* 270.1-2, pp. 489–514.

Tort, Cenk, Serkan Şahin, and Oğuzhan Hasançebi (2017). “Optimum design of steel lattice transmission line towers using simulated annealing and PLS-TOWER”. In: *Computers & Structures* 179, pp. 75–94.

Turner, Alasdair et al. (1996). “Obtaining multiple distinct solutions with genetic algorithm niching methods”. In: *International Conference on Parallel Problem Solving from Nature.* Springer, pp. 451–460.

Turner, Patrick Alasdair (1994). “Genetic algorithms and multiple distinct solutions”. PhD thesis. Citeseer.

Van Den, Augustinus Franciscus Johannes Maria, Mireille Maria Augusta Van Damme, et al. (2013). *Disease resistant plants.* US Patent 8,354,570.

Wall, Robert J et al. (2005). “Genetically enhanced cows resist intramammary Staphylococcus aureus infection”. In: *Nature biotechnology* 23.4, pp. 445–451.

Wolfson, Haim J and Isidore Rigoutsos (1997). “Geometric hashing: An overview”. In: *IEEE computational science and engineering* 4.4, pp. 10–21.

Wong, DF, Hon Wai Leong, and HW Liu (2012). *Simulated annealing for VLSI design.* Vol. 42. Springer Science & Business Media.

Wortmann, Thomas and Giacomo Nannicini (2016). “Black-box optimisation methods for architectural design”. In.

Yang, Xin-She (2011). “Metaheuristic optimization”. In: *Scholarpedia* 6.8, p. 11472.

Yau, Yuan-Yeu and C Neal Stewart (2013). “Less is more: strategies to remove marker genes from transgenic plants”. In: *BMC biotechnology* 13.1, pp. 1–23.

Zhang, David Yu and Erik Winfree (2009). “Control of DNA strand displacement kinetics using toehold exchange”. In: *Journal of the American Chemical Society* 131.47, pp. 17303–17314.

Appendix A - CANDL file definition

```
colspn [toy]
{
constants:
all:
  int D1 = 7;
  int D2 = 8;
  int dS = 3; // short distance
  int dM = 5; // medium distance
  int dL = 8; // long distance
parameters:
  double rateShort = 0.009;
  double rateShortInit = 0.003;
  double rateShortFinal = 0.0009;
  double rateMedium = rateShort/50;
  double rateMediumInit = rateShortInit/50;
  double rateMediumFinal = rateShortFinal/50;
  double rateLong = rateShort/100;
  double rateLongInit = rateShortInit/100;
  double rateLongFinal = rateShortFinal/100;
  double rateLoop = 1e-09;
  double weightBlock = 0.7;
block:
  int m_X = 0;
  int m_NX = 0;

colorsets:
  Dot = {dot};
  CD1 = {1..D1};
  CD2 = {1..D2};
  enum Type = {INIT,FINAL,NORM,FORK,JOIN};
  enum Label= {E,T,F,X,NX};
  Circuit = PROD(CD1,CD2,Type,Label);

variables:
  CD1 : x1;
  CD2 : y1;
  CD1 : x2;
  CD2 : y2;
  Type : z1;
  Type : z2;
  Label : w1;
  Label : w2;

colorfunctions:

bool Positions(CD1 x, CD2 y, Type z, Label w) {
   (x=2 & y=1 & z=INIT & w=E) | (x=3 & y=3 & z=FORK & w=E)|
   (x=3 & y=5 & z=NORM & w=X) | (x=5 & y=3 & z=NORM & w=NX)|
   (x=3 & y=7 & z=FINAL & w=T)| (x=6 & y=1 & z=FINAL & w=F)
};

bool Blockage(Label w) {
    w!=E & w!=T & w!=F
};

// Rectilinear distance, Manhattan distance, L1 norm
CD1 RectilinearDistance(CD1 x1,CD2 y1,CD1 x2,CD2 y2) {abs(x1-x2) + abs(y1-y2)};

// Chessboard distance, Chebyshev distance, Loo norm
CD1 ChessboardDistance(CD1 x1,CD2 y1,CD1 x2,CD2 y2) {max(abs(x1-x2), abs(y1-y2))};

bool NoSelfLoop (CD1 x1,CD2 y1,CD1 x2,CD2 y2)
     {(x1 != x2 | y1 != y2)};

bool ShortDistance (CD1 x1,CD2 y1,CD1 x2,CD2 y2)
     {RectilinearDistance(x1,y1,x2,y2) <= dS || ChessboardDistance(x1,y1,x2,y2) < dS};

bool MediumDistance(CD1 x1,CD2 y1,CD1 x2,CD2 y2)
     {(RectilinearDistance(x1,y1,x2,y2) > dS && ChessboardDistance(x1,y1,x2,y2) >=
         dS) &&
      (RectilinearDistance(x1,y1,x2,y2) <= dM || ChessboardDistance(x1,y1,x2,y2) <
          dM)};

bool LongDistance (CD1 x1,CD2 y1,CD1 x2,CD2 y2)
     {(RectilinearDistance(x1,y1,x2,y2) > dM && ChessboardDistance(x1,y1,x2,y2) >=
         dM) &&
      (RectilinearDistance(x1,y1,x2,y2) <= dL || ChessboardDistance(x1,y1,x2,y2) <
          dL)};

bool IsNeighbourShortD (CD1 x1,CD2 y1,Type z1,Label w1,CD1 x2,CD2 y2,Type z2,Label w2)
     {ShortDistance(x1,y1,x2,y2) && NoSelfLoop(x1,y1,x2,y2) && Positions(x1,y1,z1,w1)
      && Positions(x2,y2,z2,w2)};

bool IsNeighbourMediumD (CD1 x1,CD2 y1,Type z1,Label w1,CD1 x2,CD2 y2,Type z2,Label
    w2)
     {MediumDistance(x1,y1,x2,y2) && NoSelfLoop(x1,y1,x2,y2) && Positions(x1,y1,z1,w1)
      && Positions(x2,y2,z2,w2)};

bool IsNeighbourLongD (CD1 x1,CD2 y1,Type z1,Label w1,CD1 x2,CD2 y2,Type z2,Label w2)
     {LongDistance (x1,y1,x2,y2) && NoSelfLoop(x1,y1,x2,y2) && Positions(x1,y1,z1,w1)
      && Positions(x2,y2,z2,w2)};

places:
discrete:
  Circuit A = [z1 = INIT]2`(x1,y1,z1,w1) ++ [z1 != INIT]1`(x1,y1,z1,w1);
  Circuit B = m[w1]`(x1,y1,z1,w1);

transitions:
  stepShort
 {[IsNeighbourShortD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 != FINAL]}
    :
    : [A - {2`(x1,y1,z1,w1)++1`(x2,y2,z2,w2)}] & [A + {2`(x2,y2,z2,w2)}]
    : [z1 = INIT]rateShortInit ++ [z1 != INIT]rateShort
    ;
  stepShortFinal
 {[IsNeighbourShortD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 = FINAL]}
    :
```

```
    : [A - {2‘(x1,y1,z1,w1)++1‘(x2,y2,z2,w2)}] & [A + {2‘(x2,y2,z2,w2)}]
    : rateShortFinal
    ;
  stepMedium
 {[IsNeighbourMediumD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 != FINAL]}
    :
    : [A - {2‘(x1,y1,z1,w1)++1‘(x2,y2,z2,w2)}] & [A + {2‘(x2,y2,z2,w2)}]
    : [z1 = INIT]rateMediumInit ++ [z1 != INIT]rateMedium
    ;
  stepMediumFinal
 {[IsNeighbourMediumD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 = FINAL]}
    :
    : [A - {2‘(x1,y1,z1,w1)++1‘(x2,y2,z2,w2)}] & [A + {2‘(x2,y2,z2,w2)}]
    : rateMediumFinal
    ;
  stepLong
 {[IsNeighbourLongD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 != FINAL]}
    :
    : [A - {2‘(x1,y1,z1,w1)++1‘(x2,y2,z2,w2)}] & [A + {2‘(x2,y2,z2,w2)}]
    : [z1 = INIT]rateLongInit ++ [z1 != INIT]rateLong
    ;
  stepLongFinal
 {[IsNeighbourLongD(x1,y1,z1,w1,x2,y2,z2,w2) && z1 != FINAL && z2 = FINAL]}
    :
    : [A - {2‘(x1,y1,z1,w1)++1‘(x2,y2,z2,w2)}] & [A + {2‘(x2,y2,z2,w2)}]
    : rateLongFinal
    ;
  loop
 {[Positions(x1,y1,z1,w1) && z1 = FINAL]}
    :
    : [A - {2‘(x1,y1,z1,w1)}] & [A + {2‘(x1,y1,z1,w1)}]
    : rateLoop
    ;

immediate:
  block
  {[Blockage(w1) && Positions(x1,y1,z1,w1)]}
```

```
    :
    : [A - {(x1,y1,z1,w1)}] & [B - {(x1,y1,z1,w1)}]
    : weightBlock
    ;
  fail
  {[Blockage(w1) && Positions(x1,y1,z1,w1)]}
    :
    : [B - {(x1,y1,z1,w1)}]
    : 1-weightBlock
    ;
}
// end colspn [toy]
```