A GENERATION PERTURBATIVE HYPER-HEURISTIC FOR COMBINATORIAL OPTIMIZATION PROBLEMS

by

George Mweshi

Submitted in fulfillment of the requirements for the degree

Master of Science (Computer Science)

in the

Department of Computer Science

Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

July 2020

ABSTRACT

A GENERATION PERTURBATIVE HYPER-HEURISTIC FOR COMBINATORIAL OPTIMIZATION PROBLEMS

by

George Mweshi

Supervisor:	Prof. Nelishia Pillay
Department:	Computer Science
University:	University of Pretoria
Degree:	Master of Science (Computer Science)
Keywords:	Grammatical Evolution, Hyper-heuristics, Examination Timetabling, Vehicle Routing, Boolean Satisfiability.

Perturbative heuristics or move operators are problem dependent operators commonly used by search techniques to solve computationally hard problems such as combinatorial optimization problems. These operators are generally derived manually by problem domain experts but this process is extremely challenging and time consuming. Hence, some initiatives aimed at automating the derivation process using search methodologies such as hyper-heuristics have been proposed in recent years. However, most of the proposed hyper-heuristic approaches generate new perturbative heuristics by recombining already existing and human-derived perturbative heuristics or components with various move acceptance criteria instead of generating the heuristics from scratch. As a result, these approaches cannot be easily applied to other problem domains where the human-derived heuristics are not available. In addition, the few hyper-heuristic approaches that have been proposed to generate perturbative heuristics from scratch are either designed for a single problem domain or applicable only to specific types of problems such as those that can be represented as graphs. The research presented in this dissertation addresses these issues by proposing a novel approach that can be used to automatically generate perturbative heuristics for any combinatorial optimization problem. In the proposed approach, perturbative heuristics are defined in terms of a set of basic operations (e.g. move and swap) and components of the solution (e.g. exam, period and room for the examination timetabling problem). Grammatical evolution, a

well-known Evolutionary Algorithm, is used to combine the basic operations and components of the solution into perturbative heuristics. The generality of the proposed approach is tested by applying it to benchmark sets from three different problem domains, namely examination timetabling, vehicle routing and Boolean satisfiability. In addition, the performance of the perturbative heuristics generated by the proposed approach on the benchmark sets is compared to that of the commonly-used human-derived perturbative heuristics as well as the perturbative heuristics generated by other hyper-heuristic approaches in the literature. The experimental results show that the perturbative heuristics evolved by the proposed approach, specifically the grammatical evolution extended approach, outperformed the human-derived perturbative heuristics on all benchmark sets from the three problem domains. When compared to existing hyper-heuristic approaches, the proposed approach obtained solutions that were superior to those obtained by most hyper-heuristic approaches on the examination timetabling problem and only slightly inferior to those obtained by the best performing hyper-heuristic approaches on the vehicle routing and Boolean satisfiability problems. This performance of the proposed approach can be attributed to the fact that the generated perturbative heuristics were applied as is with no optimization as is commonly done with most hyper-heuristic approaches. Overall, the experimental results demonstrated success in developing an approach that can be used to automatically generate perturbative heuristics from scratch. Future work will consider incorporating optimization techniques during problem solving as well as performing a fitness landscape analysis in order to further improve the quality of solutions and have a better understanding of the proposed approach.

PLAGIARISM DECLARATION

I, George Mweshi (Student Number : 16394713), declare that

1. The research reported in this thesis, except where otherwise indicated, is my original research.
2. This thesis has not been submitted for any degree or examination at any other university.
3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This thesis does not contain any other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   (a) their words have been rewritten but the general information attributed to them has been referenced;

   (b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.
5. Where I have reproduced a publication of which I am author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.
6. This thesis does not contain text, graphics or tables copied and pasted from the internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

PUBLICATIONS

Details of the contributions to publications that form part and/or include research presented in this thesis:

1. G. Mweshi and N. Pillay, "A Grammatical Evolution Approach for the Automated Generation of Perturbative Heuristics," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2019)*, Wellington, New Zealand, pp. 2643–2649, 2019.
2. G. Mweshi and N. Pillay, "An Improved Grammatical Evolution Approach for Generating Perturbative Heuristics to solve Combinatorial Optimization Problems," *August 2020 Expert Systems with Applications*, DOI: 10.1016/j.eswa.2020.113853.

DEDICATION

To my beloved wife Doris for the remarkable patience, unwavering love and support during the course of this research. And to my son Thando, who was born during the study period.

ACKNOWLEDGEMENTS

I am greatly indebted to the following people and institutions for their contributions to this research:

- Prof. Nelishia Pillay, who supervised this research, for her exceptional, insightful and professional guidance through out the journey. Her approachable demeanor made the consultative meetings so much fun and less stressful even when my confidence and morale were low —a great supervisor indeed;
- Mulungushi University for the financial support and for always making sure that the monies were in on time despite the financial difficulties faced by the university;
- The Centre for High Performance Computing (CHPC) for providing the computational resources on which to run the simulations for this research;
- Prof. Douglas Kunda, Dean, School of Science and Engineering Technology (SSET), Mulungushi University, for pushing me to do my very best and constantly reminding me of the need to finish the research in time;
- NICOG research group members for the wonderful discussions and valuable suggestions during our group meetings.

LIST OF ABBREVIATIONS

CO	Combinatorial Optimization
EAs	Evolutionary Algorithms
GP	Genetic Programming
GE	Grammatical Evolution
RVD	Random initialisation with valids and no duplicates
HH	Hyper-Heuristics
llhs	Low-level heuristics
SR	Success rate
AF	Average flips
IO	Improving or equal only
AM	All moves
SA	Simulated annealing

TABLE OF CONTENTS

CHAPTER 1 Introduction

1.1 PURPOSE OF STUDY

Many real-world problems are inherently difficult to solve computationally due to their usually large and heavily constrained search spaces. As a result, approximation and heuristic approaches, although providing no guarantees on the quality of solutions they obtain, are used in practice since exact techniques often fail to find good solutions in a reasonable amount of time. Perturbative heuristics or move operators play a crucial role in improving the quality of solutions obtained by search techniques. These operators are traditionally designed manually by domain experts through trial and error but some initiatives aimed at automating the design process have since been proposed. Although these initiatives have produced good results for the problem domains they have been applied to, the area of automated generation of perturbative heuristics has generally not been well-researched and very few works have actually been conducted in the field. The study presented in this dissertation adds to this body of knowledge by proposing a novel approach for automating the design of perturbative heuristics to solve combinatorial optimization problems. The aim of the research work is to develop a simple and general approach that can be used to automatically generate good quality perturbative heuristics for any combinatorial optimization problem domain without relying on pre-existing human-derived heuristics or requiring significant domain expertise. The research can therefore be considered as forming part of a larger initiative which aims to automate the design of machine learning and search techniques.

1.2 OBJECTIVES OF STUDY

To achieve the aim of this research work, an in-depth survey of the literature related to the automated generation of perturbative heuristics using search techniques such as hyper-heuristics will be carried

out. The focus on hyper-heuristics is motivated by the fact that these search techniques provide a more general framework for developing cross domain algorithms which is one of the goals of this research work. And since most hyper-heuristics, especially those concerned with generating heuristics, employ either genetic programming (GP) or grammatical evolution (GE), the literature survey will also include a detailed discussion on both GP and GE.

Based on the analysis of the literature, a new GE-based approach for generating perturbative heuristics to solve combinatorial optimization problems in more than one problem domain is proposed in this study. GE is selected over GP due to the simplicity with which it can represent heuristic components. To test and evaluate the feasibility of the proposed approach, it is applied to benchmark sets from three well-known problem domains, namely examination timetabling, vehicle routing and boolean satisfiability. In addition, the performance of the perturbative heuristics generated by the proposed approach on the benchmark sets is compared to that of the commonly used human-derived perturbative heuristics and existing generation perturbative hyper-heuristics in the literature. The objectives of the study can therefore be summarized as follows:

1. To develop an approach that automatically generates perturbative heuristics for more than one problem domain using grammatical evolution;
2. To test the generality of the proposed approach on three different problem domains, namely examination timetabling, vehicle routing and boolean satisfiability;
3. To compare the performance of the perturbative heuristics generated by the proposed approach to that of the human-designed perturbative heuristics for the three problem domains;
4. To compare the performance of the perturbative heuristics generated by the proposed approach to that of the perturbative heuristics generated by existing generation perturbative hyper-heuristics in the literature.

1.3 CONTRIBUTIONS

The main contribution of the study presented in this dissertation is the new approach for automating the generation of perturbative heuristics to solve combinatorial optimization problems. As mentioned earlier, this domain has not been well-researched and therefore this research makes a significant contribution to the field in this regard. In addition, this research:

1. Provides a thorough survey and analysis of the research works that have been conducted in the domain of automated generation of perturbative heuristics using hyper-heuristics.
2. Is the first study to propose a general approach for generating perturbative heuristics to solve any type of combinatorial optimization problem based on a thorough investigation of the current literature.

1.4 LAYOUT OF DISSERTATION

The layout of this dissertation is as follows:

1.4.1 Chapter 2 - Combinatorial Optimization Problems

This chapter provides a brief introduction to optimization problems with the main focus being on combinatorial optimization problems. A discussion on some of the approaches commonly used to solve combinatorial optimization problems is also presented.

1.4.2 Chapter 3 - Genetic Programming

This chapter discusses genetic programming, an evolutionary algorithm that is widely used to get computers to solve problems on their own. Genetic programming has been widely employed in many heuristic generation approaches. Although many variants of genetic programming exist today, the chapter nevertheless focuses on the original and easier to understand tree-based variant proposed by Koza [1]. The chapter also discusses other aspects of genetic programming such as methods used to generate the initial population of individuals, how to evaluate the goodness (i.e. fitness) of an individual, methods used to select individuals to participate in reproduction and the genetic operators that can be applied to the selected individuals. In addition, a discussion on the genetic programming algorithm including the preparatory steps and control parameters required for a successful genetic programming run is presented.

1.4.3 Chapter 4 - Grammatical Evolution

This chapter provides an overview of another popular evolutionary algorithm employed in many heuristic generation approaches called grammatical evolution. Grammatical evolution is a grammar-based variant of genetic programming. The chapter includes a discussion on the main components of grammatical evolution, namely the grammar, search engine and a mapper. In addition, a discussion on the overall grammatical evolution algorithm including the methods used to generate the initial population of individuals, the criteria used to evaluate the goodness (i.e. fitness) of an individual, the methods used to select individuals to participate in reproduction and the genetic operators that can be applied to the selected individuals is presented.

1.4.4 Chapter 5 - Hyper-heuristics

This chapter discusses search techniques called hyper-heuristics. Hyper-heuristics are widely accepted as more general search techniques than meta-heuristics such as evolutionary algorithms since they usually obtain good results over a wider range of problems unlike meta-heuristics which obtain good results only for one or a few problems, and poor results for others. The chapter firstly introduces the definition of hyper-heuristics adopted in this research and thereafter discusses the four main classes of hyper-heuristics. Special attention is however paid to generation perturbative hyper-heuristics as this is the class of hyper-heuristics that is more relevant to this research. A survey and analysis of the research works that have been conducted in the domain of generation perturbative hyper-heuristics is also presented.

1.4.5 Chapter 6 - Methodology

The chapter describes the methodology adopted in order to achieve the objectives of this study. The chapter first presents a critical analysis of the related literature which is followed by a discussion on the research methodology adopted for the study. A brief discussion on the two proposed GE-based approaches, namely the grammatical evolution baseline approach and the grammatical evolution extended approach is also presented. In addition, a discussion on the problem domains the proposed approaches will be applied to, how the initial solutions will be constructed and the performance measures that will be considered when evaluating the approaches, is presented. The chapter ends with

a discussion on the technical specifications of both the software and hardware used to achieve the objectives of the study.

1.4.6 Chapter 7 - Grammatical Evolution Baseline Approach

This chapter discusses the grammatical evolution baseline approach in more detail. The baseline approach was developed to test the main idea behind the proposed GE approach. As a baseline approach, it features a basic BNF grammar which incorporates minimal domain knowledge. A discussion on all aspects of the approach including the basic BNF grammar is presented.

1.4.7 Chapter 8 - Grammatical Evolution Extended Approach

This chapter discusses the grammatical evolution extended approach which is considered as an improvement on the baseline approach. The extended approach was developed in order to improve the quality of solutions obtained by the baseline approach. In the extended approach, the BNF grammar has been redesigned to allow for the capturing of more domain specific knowledge from the search space as well as the generation of a wider range of perturbative heuristics including decision rules. The new grammar and all aspects of GE are discussed in detail.

1.4.8 Chapter 9 - Results and Discussion

This chapter presents and discusses the results obtained by the two approaches, namely the baseline and extended approaches on benchmark sets from the examination timetabling, vehicle routing and boolean satisfiability problem domains. The obatined results are further compared to those obtained by the commonly used human-derived perturbative heuristics as well as existing generation perturbative hyper-heuristics. Statistical test results to evaluate the significance of the obtained results are also presented.

1.4.9 Chapter 10 - Conclusion and Future Works

Finally, this chapter provides a summary of the research findings, outcomes of each research objective and presents some ideas on future works based on the research findings.

CHAPTER 2 Combinatorial Optimization Problems

2.1 INTRODUCTION

To optimize generally means to make the best decision so that the available resources are used effectively [2]. Human beings encounter these decision problems every day. For example, one may ask: what time should I wake up to make it on time for the lecture? This question may seem simple at first but after some thought, one soon realises that there are certain interesting intricacies associated with the question. It is also obvious at this point that the decision on the best time to wake up will depend on what resources this person has and how much he knows. If the person knows what time the lecture will start and also knows the quickest route to get there, then the best time to wake up can be estimated very easily. Optimization problems conceptualize the notion of effectively using the resources and knowledge one has available. As such, these problems are embroidered in almost all the decisions that people make.

In applied computer science and more specifically in the area of problem solving, optimization plays an imperative role in the design of efficient, robust machine learning and search techniques to solve real-world problems. And since thousands of real-world problems can be formulated as abstract combinatorial optimization problems, it is therefore important to understand what combinatorial optimization problems are and why such a formulation makes it easier to solve these otherwise very complex problems.

This chapter presents a brief overview of optimization problems within the context of problem solving using search. Section 2.2 introduces optimization problems while section 2.3 discusses another common type of optimization problems called combinatorial optimization problems. Sections 2.4-2.6

briefly discusses some of the techniques commonly used to solve combinatorial optimization problems. The chapter summary is provided in section 2.7.

2.2 OPTIMIZATION

Optimization can be broadly defined as the art of selecting the best element from a given set of alternatives [3]. The goodness of an element in the set is usually specified by an objective function. As such, an optimization problem can be described as the problem of finding the smallest or largest value of an objective function subject to a given set of constraints and the relationship between one or more decision variables. The general mathematical form of an optimization problem as proposed by Boyd et al. [4] is shown in Equation (2.1).

$$\begin{aligned} &\underset{x}{\text{minimize / maximize}} && F(X) \\ &\text{subject to} && f_i(x) \leq b_i,\ i = 1, \ldots, m. \end{aligned} \tag{2.1}$$

where X = $(x_1, ...x_n)$ is a vector of problem variables, F(X) : $\mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function and $f_i(x) \leq b_i$ is the set of constraints

Optimization problems can be divided into two categories depending on the type of values the decision variables encoded with. If the variables are encoded with real numbers, then the problem is called *continous*. If they are encoded with discrete values, then the problem is called *discrete*. This research is concerned with discrete optimization problems and within these types of problems, the focus is on Combinatorial Optimization (CO) problems where mathematical techniques are applied in order to find optimal solutions from a finite set of possible solutions. Usually, for such problems, the set of possible solutions is not only defined by a set of restrictions, but is also too large for an exhaustive search. As a result, it is not guaranteed to find an algorithm that can solve these problems in polynomial time. Combinatorial optimization is discussed in more detail in the next section.

2.3 COMBINATORIAL OPTIMIZATION

The word "combinatorial" refers to an ordering or arrangement of items. Combinatorial optimization therefore aims to find the best ordering or arrangement of the items from a finite or countably infinite set of possible alternatives [5]. The ordering or arrangement maybe in the form of mathematical

structures such as sets, permutations, integer numbers, graphs, matroids, or polytopes [6]. A formal definition of a combinatorial optimization problem, as used in this research, is given in Definition 2.1 [6].

Definition 2.1 *Given a set of instances* $\boldsymbol{I}$*, where each instance* $x \in \boldsymbol{I}$ *is specified by a set* $\boldsymbol{S}$ *of all feasible solutions for* x *(also called search space), constraints for* x *and an objective function* $f : \boldsymbol{S} \rightarrow \mathbb{R}$*, a* combinatorial optmimization problem *is the problem of finding a feasible solution* $s \in \boldsymbol{S}$ *such that* $f(s)$ *is minimum or maximum. This solution* s *is called the optimal solution of* x*.*

Apart from their theoretical relevance [5], combinatorial optimization problems have a very important practical impact due to their applicability to many real–world scenarios [7]. In fact, one of the biggest challenges in solving real-world problems using search is the modelling of the problems. Reformulating these usually complex problems as combinatorial optimization problems makes it easier to develop more efficient and robust search algorithms that are able to exploit the combinatorial structures. For example, many real-world problems such as routing, scheduling, timetabling, economic systems, production planning and management can be formulated as combinatorial optimization problems [8]. Examples of non combinatorial optimization problems include classification and regression problems.

In principle, since the set of all feasible solutions **S** is finite or countably infinite, a combinatorial optimization problem can in essence be exactly solved by any algorithm that basically enumerates all the elements in **S** and outputs the element with the best objective function value [7]. However, such an approach is very inefficient for solving real-world problems due to the fact that the number of feasible solutions often grows exponentially with the size of the problem instance that needs to be solved. As a result, a variety of approaches including exact, approximation and heuristic approaches have been developed to solve combinatorial optimization problems [7]. The approaches are discussed in Section 2.4, Section 2.5 and Section 2.6 respectively.

2.4 EXACT APPROACHES

As mentioned earlier, among the various approaches that have been developed to solve combinatorial optimization problems are exact or classical approaches [7]. Some examples of these classical approaches include Branch and Bound [9] and Dynamic Programming [10]. The two approaches

solve a problem by breaking it down into sub-problems and then combining the solutions to the sub-problems. They are considered as divide-and-conquer approaches and the main difference between them is in the way they divide a problem into its sub-problems. In the Branch and Bound algorithm, the original problem is partitioned into independent sub-problems which are then solved and the best feasible solution found along the search is returned as the optimal solution to the original problem. In Dynamic Programming, the original problem is divided into sub-problems which are further divided into non-independent sub-sub-problem such that each sub-sub-problem is solved only once. Dynamic Programming also requires that the original problem has an optimal substructure in addition to the problem being divided into non-independent sub-problems [7]. More information on the two approaches can be found in [7].

2.5 APPROXIMATION APPROACHES

Approximation approaches, unlike exact approaches, aim to find a solution that provides some form of approximation-guarantee on the quality of the solution found [7]. In most cases, the solution is suboptimal but for many hard computational problems, approximation approaches provide the best that can be done in terms of providing some guarantee on the quality of the solutions obtained. A variety of approximation approaches have been developed in the literature [11], [12], [13]. And some examples of these approaches are greedy algorithms, relaxation based algorithms, local search algorithms, sequential algorithms and random algorithms [7].

2.6 HEURISTIC APPROACHES

Exact approaches often require exponential computation time in solving large problem instances and as a consequence are not guaranteed to find an optimal solution in polynomial time since they take too long to run. For this reason, they are not used in practice and heuristic algorithms are preferred. A heuristic can be broadly described as a method designed for finding a satisfactory solution to a problem more quickly when exact approaches are too slow. Although no general framework exists with regard to the design of a good heuristic approach to guarantee finding good solutions for any problem, most heuristic approaches are designed with the basic principles of avoiding getting trapped at a local optima and exploiting promising areas in the solution space where good solutions are likely to be found[7]. Based on these principles, a variety of heuristic search techniques including

metaheuristics such as evolutionary algorithms [14], simulated annealing [15], scatter search [16], GRASP (Greedy Randomized Adaptive Search Procedures) [17], great deluge [18], late acceptance [19], ant colony optimization [20], variable neighborhood search [21] and iterated local search [22], have been developed. These techniques have greatly improved our ability to obtain good solutions to difficult combinatorial optimization problems.

2.6.1 Meta-Heuristics

Meta-heuristics are heuristic search techniques commonly applied to solve computationally hard problems such as combinatorial optimization problems [7]. Meta-heuristics basically combine basic heuristic approaches in some higher level framework with the aim of efficiently exploring the set of potential or candidate solutions for a given combinatorial optimization problem. The main idea is that by using some knowledge from the solution space, search algorithms can be able to explore (i.e. visit new areas or regions in the solution space to look for better solutions) and exploit (i.e. search for better solutions within the neighbourhood of the current solution) the solution space more efficiently. Meta-heuristics are also able to escape being trapped in the local optima (i.e. the best solution to the problem within a small neighborhood of possible solutions) by jumping to other regions within the solution space where better solutions are likely to exist. Meta-heuristics can be broadly categorized into single-solution (single-point) and population-based (multi-point) meta-heuristics. Population-based meta-heuristics such as evolutionary algorithms [14], improve and maintain a collection (usually referred to as a population) of potential solutions while single-point meta-heuristics such as tabu search [23], maintain and improve a single solution.

One of the most interesting issues that immediately comes to mind when solving combinatorial optimization problems is deciding which meta-heuristic to use. And since there is no particular way of comparing all the available meta-heuristics with each other, this issue is quite difficult to address. A simple approach would be the trial-and-error approach where all the meta-heuristics are tried, but this not practical for many real-world applications. Another problem is that meta-heuristics, just like heuristics, are manually designed and often fine tuned for a specific problem domain or instance. As a result, they usually produce high quality solutions for some problem instances while performing poorly on other instances including those of the same problem. To address these issues, more general search methodologies such as hyper-heuristics (HHs) have been proposed.

2.6.2 Hyper-Heuristics

As mentioned in Section 2.6.1, meta-heuristics although capable of obtaining high quality solutions for some problem instances do not generalize very well across different problem instances or domains. Furthermore, the need to redevelop the algorithms for every new problem instance is both time consuming and challenging as it requires a deep understanding of the algorithm itself and the structure of the problem instances. Hyper-heuristics attempt to address these issues in two main ways: by exploring whether a suitable combination of existing heuristics can offset the weaknesses of any one of them so that each heuristic is applied only when it performs well and by attempting to discover new heuristics mostly through the use of some meta-heuristic. Consequently, hyper-heuristics are distinguished between those which select heuristics to apply during problem solving (i.e. selection hyper-heuristics) and those which generate new heuristics for solving problems (i.e. generation hyper-heuristics). The heuristics themselves can either be *constructive* (i.e. heuristics that build a solution from scratch and iteratively add elements to it to obtain feasible solution) or *perturbative* (i.e. heuristics that iteratively modify an existing solution to improve its quality). Hyper-heuristics usually employ a variety of techniques during the heuristic selection and heuristic generation processes [24].

In general, hyper-heuristics explore the space of heuristics instead of the solution space. The exploration is done automatically using some meta-heuristic with no intervention by a domain expert. Hyper-heuristics have been successfully applied to solve various combinatorial optimization problems such as the 1-D bin packing, examination timetabling, boolean satisfiability and vehicle routing problems to mention but a few.

2.7 SUMMARY

This chapter introduced combinatorial optimization problems and briefly discussed some of the methods commonly used to solve these computationally hard problems. Combinatorial optimization problems are optimization problems that aim to find the best or optimal element from a finite or countably infinite set of alternatives. Many real-world problems can be represented as abstract combinatorial optimization problems. These problems are often too hard and complex to solve in a reasonable amount of time using exact methods and as a result heuristic techniques are used. Heuristic techniques are usually designed manually by problem domain experts through trial and error. As a result, they

perform very poorly in some cases. To address this issue, researchers proposed new search strategies called meta-heuristics. These strategies improve upon the basic heuristic techniques by allowing for the exploration as well as exploitation the solution search space. Meta-heuristics are however designed and fine tuned for a specific problem which limits their generality. Hyper-heuristics on the other hand search the heuristic space instead of the solution space and are therefore more general search methodologies. Due to their generality, hyper-heuristics have since emerged as the preferred general search methodology for solving combinatorial optimization problems.

This research is concerned with automating the generation of pertubative heuristics to solve combinatorial optimization problems. As such, the approach proposed in this research can be considered as some form of a generation perturbative hyper-heuristic. And since most generation perturbative hyper-heuristics in the literature employ evolutionary algorithms such as genetic programming (GP) [1] and grammatical evolution (GE) [25], this thesis will also present a chapter on each of the two evolutionary algorithms. The next chapter discusses GP in more detail.

CHAPTER 3 Genetic Programming

3.1 INTRODUCTION

The process of manually writing computer programs or algorithms to solve real-world problems is extremely challenging and time consuming. For many years, computer scientists had been looking for ways to automate this process. Fortunately, the recent developments in the fields of artificial intelligence and machine learning have made it possible to develop techniques which allow computers to solve problems on their own without necessarily telling them how to. One such technique is Genetic programming (GP).

GP, first introduced by Cramer [26] and made popular by Koza in his publication entitled “Genetic Programming: On the Programming of Computers by Natural Selection” [1], is a very powerful evolutionary algorithm capable of automatically evolving computer programs to solve various problems. It uses ideas from Darwin’s theory of evolution —"survival of the fittest", to iteratively transform a population of random computer programs into a new generation of computer programs, often better ones, by applying analogues to naturally occurring biological processes such as sexual recombination (crossover), mutation and reproduction [27].

Although GP is a random and stochastic search technique which provides no guarantees that a solution will always be found, it has been successfully used in many real-world application domains as a computing technique to get computers to automatically solve problems without the need to explicitly tell them how to. This chapter presents an overview of GP. In particular, it describes the basic terminology and tools used to describe how potential solutions are represented, how initial populations are generated and how the concepts of fitness, selection, reproduction, crossover and mutation are used to generate new programs. The generational GP algorithm is also introduced and this is followed by

a discussion on the preparatory steps and control parameters required for a successful GP run. The chapter ends with a summary of some of the main discussion points.

3.2 GP ALGORITHM

The generational GP algorithm is similar to other evolutionary algorithms. Firstly, a initial population of individuals is randomly created. The quality (fitness) of each individual in the population is then determined by assigning to the individual, using some well-defined function (see Section 3.7), a fitness value which represents the degree to which the individual is able to solve the problem at hand. Once all the individuals in the population have been assigned a fitness value, one or two individuals, usually those with best fitness values, are the selected as parents to generate new offspring through the application of genetic operators such as crossover, reproduction and mutation [28]. The pseudocode for a GP algorithm is given in Algorithm 1

Algorithm 1 Pseudocode for a GP algorithm

1: Randomly generate a population of computer programs as solutions from the identified primitives
2: Determine the fitness of each computer program in the population by executing it on some test or fitness cases.
3: **while** termination criterion not met **do**
4: Select one or two programs with the highest fitness using a selection method
5: Create new programs for the next generation by applying genetic operators to selected programs
6: Execute each new program to determine its fitness
7: Replace all the programs in the old population with the new programs
8: **end while**
9: **return** program with the highest fitness as the best solution

3.2.1 GP Preparatory Steps

In general, to solve a problem using GP, a user has to make several initial decisions which are commonly referred to as preparatory steps. These steps include:

1. Identifying and specifying a set of terminals for creating computer programs,

2. Identifying and specifying a set of functions for creating computer programs,
3. Identifying and specifying a fitness measure i.e. identifying a way of determining the quality of the evolved computer program,
4. Specifying the parameters to use for controlling the GP run, and
5. Specifying the criterion for terminating the GP run

The next sections discuss some of the terminologies and tools used in GP. A more detailed discussion on each of the preparatory steps is also presented.

3.3 REPRESENTATION

The various subclasses of evolutionary algorithms can be distinguished by how they represent the individuals (or solutions) [14]. In GP, the individuals are computer programs and these were initially represented as *syntax trees* when the technique was first introduced by Koza [1]. The current variants of GP use a much broader set of more complex representations including graphs, grammars, probability distributions and linear structures. This chapter however focuses on the initial tree-based representation as it is easier to understand.

In the tree-based GP, the syntax trees are composed of nodes and *links* (arcs), where the nodes specify which instructions to execute while the links specify the number of arguments (i.e. arity) for the instructions. The nodes without children (leaf nodes) are called *terminals* and the nodes with children are called *functions or non terminals*. The set of all the functions available to the GP system is called the *function set* while the set of all the available terminals is called the *terminal set*. The function and terminal sets collectively form what is usually referred to as primitive set of the GP system [1], [29].

An example of a typical GP individual is depicted in Figure 3.1. In the example, a node is represented by an ellipse and a link is represented by a straight line. The functions include the *max*, + and - nodes. The *max* node is also the root node of the tree. Terminals include the x, y and z nodes. All the functions in the example have an arity of two since they take two arguments. The terminals have an arity of zero.

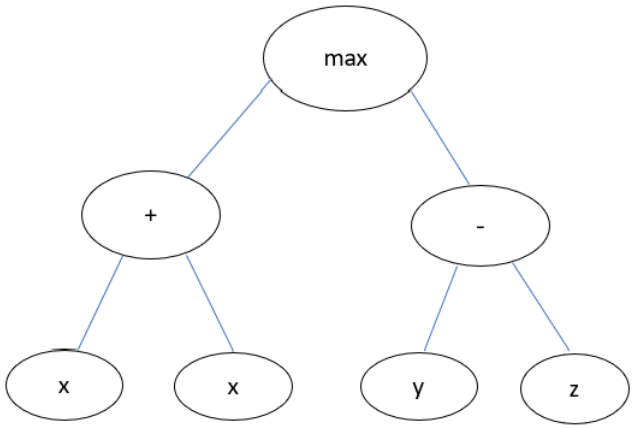

Figure 3.1. A simple GP Individual

Another commonly used representation of GP individuals is the Lisp S-expression. In this representation, the *prefix notation* where functions come before their arguments is often preferred. For example, the tree in Figure 3.1 can be equivalently represented as (max (+ x x) (- y z)) using the S-expression.

3.4 TERMINAL SET

The terminal set can be described as a set of all the inputs that will be needed by the as-yet-undiscovered computer program to solve, or approximately solve, the problem. In practice, the terminal set contains variables which represent the inputs to the computer program, functions with an arity of zero which are executed by the program and numerical constants which may be needed by the program. The terminal set is problem dependent. For example, in a symbolic regression problem, the terminal set may consist of all the attributes or variables in the input dataset and some numerical constants. In some problems, *ephemeral random constants* may be desirable. These are random constants selected from some range during the generation of the initial population. The purpose of this is to allow for constants to be chosen from a range rather then including all the constants which will increase the search space.

3.5 FUNCTION SET

The function set specifies all the functions that will be applied to the elements of the terminal set. These functions may range from simple arithmetic functions {+, -, *, / }, trigonometric functions {cos, sin, exp, log}, logical functions{ AND, OR, NOT}, decision structures such as *if-then-else* and iterative structures such as loops, to more complex domain-specific functions. The decision on which functions to use also depends on the problem one is trying to solve. For example, logical functions are usually used for logical problems. In non-linear problems, non-linear functions may be preferred.

3.6 INITIAL POPULATION

GP, just like other population-based algorithms requires an initial population of candidate solutions to be generated as the first step in solving a problem. For the tree-based GP, this step entails the construction of syntactically valid trees. Usually, the trees are randomly constructed from the terminal and function sets chosen for the problem. As the tree construction process is random, it is possible to generate very large trees which may lead to problems such as overfitting and code bloat [30] —a tendency of GP trees to grow in an uncontrolled manner. In an attempt to avoid these problems, there is usually a limit on the maximum depth of the trees during the generation of the initial population.

The three methods commonly used to generate the initial population of candidate solutions in GP are the *grow*, *full* and *ramped half-and half* methods [1].

3.6.1 The Grow Method

This method creates individuals one at a time. The individuals may be trees of any depth but not more than the *maximum tree depth* limit. Algorithm 2 shows the grow method as a recursive function which takes as an argument the depth of the *node* to be generated and returns the generated *node*. The function is first called with *depth* 0. It then checks whether this *depth* is less than the *maximum tree depth*. If it is, then a node is selected randomly from the function and terminal sets. Next the grow method is called to generate the children of the node depending on whether the selected node is a function or terminal. If the *depth* is equal to the maximum tree depth, the node is selected from the

terminal set. Figure 3.2 shows an example of a tree with a maximum tree depth of 2 constructed using the grow method .

Algorithm 2 *node* **growMethod**$(depth)$

```
if depth < maximum depth then
    node ← getRandom (Terminal or Function)
    for i = 1 to number of children of node do
        child_i = growMethod(depth + 1)
    end for
else
    node ← getRandom(Terminal)
end if
Return node
```

The root node is created first at $t = 1$, followed by the first child x which is a terminal at $t = 2$. The second child $+$ which is a function is created at $t = 3$ but because of the depth limit set by the maximum tree depth, the children of node $+$ are forced to become terminals.

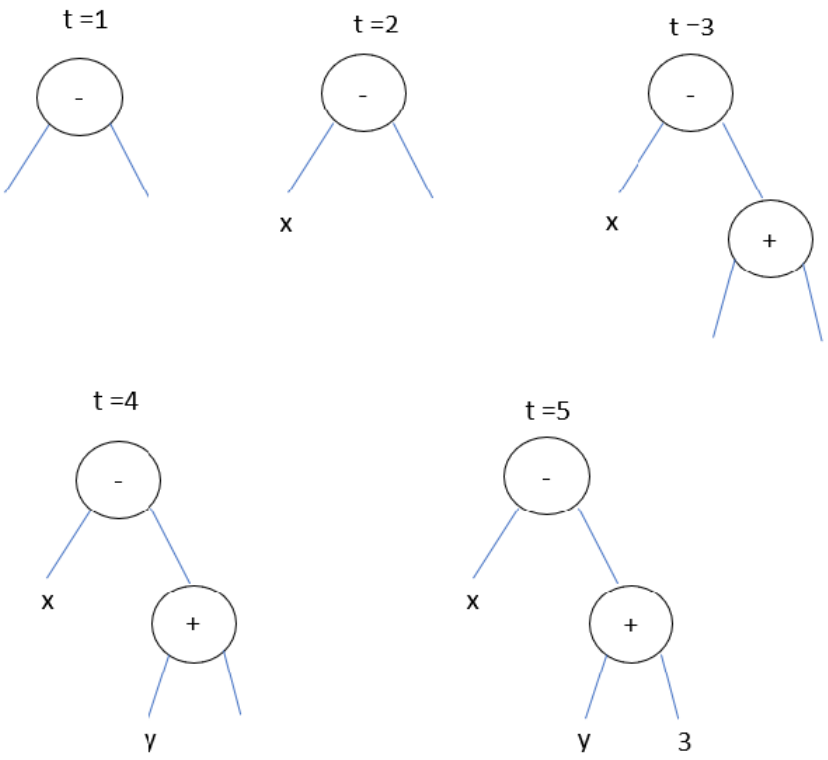

Figure 3.2. Construction of a syntax tree with a maximum depth of 2 using the grow method (t = time)

The grow method provides no guarantee that the individuals will be of a certain depth (although they will not be deeper than the maximum depth). However, it produces individuals of various shapes and sizes which increases individual diversity in the population to some extent.

3.6.2 The Full Method

The full method is similar to the grow method but differs in that the non-terminal nodes are always selected from the function set as long as the *depth* of the tree is not more than the maximum tree depth allowed. If the *depth* is equal to the maximum tree depth, then the node is selected from the terminal set. The full method guarantees that nodes from the terminal set will always be at a certain depth. The algorithm for the full method is given in Algorithm 3. An example of a tree with a maximum tree depth of 2 constructed using the full method is shown in Figure 3.3.

Algorithm 3 *node* **fullMethod**$(depth)$

if $depth <$ maximum depth **then**
  $node \leftarrow$ getRandom (Function)
  for $i = 1$ to *number of children of node* **do**
    $child_i = fullMethod(depth+1)$
  end for
else
  $node \leftarrow getRandom(Terminal)$
end if
Return *node*

The root node $-$ is created first at $t = 1$, followed by the first child *cos* which has one child at $t = 2$. The child for the node *cos* is then created at $t = 3$. This is followed by the node $+$ which has two children. The children for nodes *cos* and $+$ are then forced to become terminals because of the maximum tree depth limit.

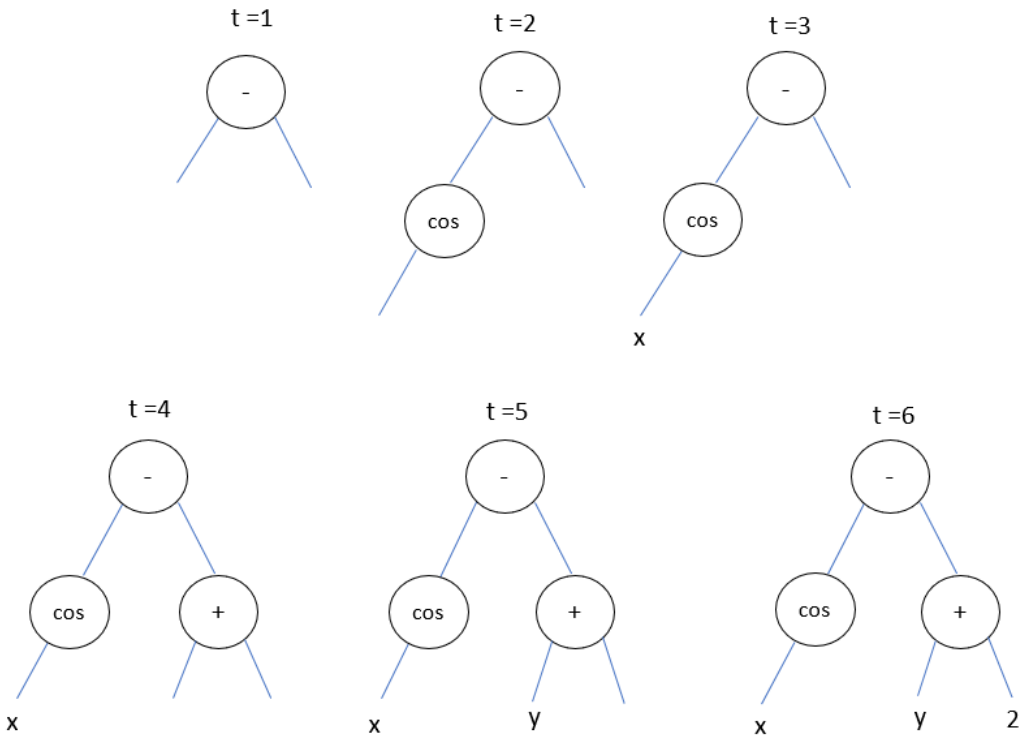

Figure 3.3. Construction of a full tree with a maximum depth of 2 using the full method (t = time)

The full method ensures that the constructed trees are full and of the same size although the number of nodes may not be the same. Due to this, there is less individual diversity in the population.

3.6.3 The Ramped half-and half Method

This method was introduced in order to increase the variation in the structure of the trees generated since both the grow and full methods do not provide enough variation on their own. The ramped haf-and-half combines both the grow and the full methods. In the method, the population is often divided into D-1 groups, where D is the *maximum tree depth*. Each group then uses different maximum tree depths (i.e. 2,....,D), to construct an equal number of trees at each depth with half of the trees constructed using the grow method and the other half using the full method. This leads to increased individual diversity in the population as the constructed trees have a variety of sizes and shapes.

3.7 FITNESS EVALUATION

The terminal and function sets can be considered as indirectly defining the search space that the GP algorithm will eventually explore since they specify the building blocks of all the computer programs that can be constructed by combining the primitives in all possible ways. Fitness evaluation provides a mechanism for specifying, to the GP algorithm, the regions of the search space which may contain programs capable of solving the problem. In GP, the *fitness* of an individual refers to the degree to which the individual solves the problem being addressed [31]. It is represented by a *fitness value*. The *fitness value* is often calculated by means of some well-defined procedure that involves applying the evolved individual to a set of test or fitness cases.

The *fitness value* of an individual may be expressed as a *raw fitness*, a *standardized fitness*, an *adjusted fitness* and a *normalized fitness*. The *raw fitness* is the simplest and most natural way of calculating how well a program solves a particular problem. For example, in the artificial ant problem, the *raw fitness* would be the number of pieces of food that are actually picked up after the program is executed. In this case, the better program would be the one with a larger fitness value. The *standardized fitness* is an adjustment of the *raw fitness* such that lower fitness values are better. The *adjusted fitness* is a reformulation of the *standardized fitness* such that the fitness values are reduced to a range of values between 0 and 1. The *normalized fitness* is the ratio of the *adjusted fitness* of an individual to the sum of the *adjusted fitness* of all the individuals in the population.

The decision on the procedure to use in order to measure the fitness of an individual is considered the most difficult and also the most important one in GP. For many problems, fitness is measured in terms of the error produced by the evolved program with respect to some supplied target outputs. In general, the most appropriate procedure to use depends on the type of problem one is trying to solve. For example, in examination timetabling problems [17], fitness is usually measured in terms of the cost of soft constraints violated by the timetable. In vehicle routing problems [32], fitness is measured in terms of the total cost of all the routes taken by the vehicles. In any case, the most important factor to keep in mind when making a decision about the fitness measure to use is to understand that a good fitness measure will help the GP algorithm to explore the solution search space more efficiently and effectively. A bad fitness measure can easily make the GP algorithm to get trapped in a local optimum solution and lose the power to discover good solutions.

3.8 SELECTION METHODS

During evolution, three things can happen to an individual in the population: the genetic material of the individual can be copied(as it is) to the next generation; or the individual can be selected for regeneration and genetic operators can be applied to the individual; or the individual can be left out completely from the next generation. In the case of the individual been selected for regeneration, a variety of methods, commonly referred to as *Selection methods* are used to do this. The selection of individuals for regeneration is a very important step in GP since it affects the pace and success of the evolutionary process [31]. The whole evolutionary process would otherwise result in a random search.

There are many selection methods but the most widely used in GP are *fitness proportionate* and *tournament* selection methods. Both of these methods have a bias towards individuals with better fitness i.e. individuals with good fitness have a higher probability of being selected than those with bad fitness.

3.8.1 Fitness proportionate selection

This method selects an individual based on the proportion of the individuals fitness to that of the entire population. In this way, fitter individuals have a higher chance of being selected more frequently than the weaker ones. The probability of selecting an individual is calculated using the steps shown in Algorithm 4

Algorithm 4 Pseudocode for fitness proportionate selection

1: Reformulate the individuals *raw fitness* in terms of *standardized fitness*

2: Reformulate the *standardized fitness* in terms of *adjusted fitness* using Equation (3.1).

$$adjFitness(i) = \frac{1}{1 + stdFitness(i)} \tag{3.1}$$

where *adjFitness(i)* and *stdFitness(i)* are the adjusted fitness and the standardized fitness for indivdual *i* respectively.

3: Calculate the *normalized fitness* using Equation (3.3).

$$normFitness(i) = \frac{adjFitness(i)}{\sum_{k=1}^{N} adjFitness(k)} \tag{3.2}$$

where *normFitness(i)* represents the normalized fitness of indivdual *i*. N represents the number of individuals in the population.

4: Probability of selection (ps) is:

$$ps(i) = \frac{normFitness(i)}{\sum_{k=1}^{N} normFitness(k)} \tag{3.3}$$

3.8.2 Tournament selection

Tournament selection, shown in Algorithm 5, is the most widely used selection method in many GP systems due to its simplicity and efficiency. It involves creating several tournaments where *n* randomly selected individuals from the population compete against each other. The winner of each tournament, usually the *fittest* individual in the tournament, is then selected for regeneration. The number of individuals competing in each tournament is referred to as the *tournament size*. The *tournament size* has a major influence on the *selection pressure* (i.e. the likelihood of an individual to participate in a tournament). If the tournament size is large, the weaker individuals will have a smaller chance of getting selected since they will have to compete with fitter individuals. On the other hand, if the tournament size is small, then the weaker individuals may have a higher chance of getting selected which in turn helps to preserve the diversity in the population. It is therefore vital that a good *tournament size* is chosen since a large tournament size may lead to a high selection pressure. And a high selection pressure may lead the GP algorithm to prematurely converge to a local optimum.

Algorithm 5 Tournament Selection

1: Randomly select *n* individuals from the population and conduct a tournament amongst them

2: Select the individual with the best fitness from the *n* individuals as winner of the tournament (and ultimately as a parent)

Tournament selection has several advantages over fitness proportionate selection. These include better time complexity, no fitness scaling and low susceptibility to takeover by fitter individuals.

3.9 GENETIC OPERATORS

Genetic operators aim to create new individuals from existing ones [29]. This is usually done by duplicating, combining or altering the genetic material of parent individuals. The underlying idea behind genetic operators is that within the population some individuals may be able to solve some parts of the problem and combining these useful parts may improve the overall performance of the algorithm. Genetic operators therefore transform a population into a new population (hopefully of better individuals) in an attempt to find a solution. In GP, genetic operators can be distinguished between those that *explore* different areas of the program space (i.e. global search operators) and those that *exploit* neighbouring areas of the program space (i.e. local search operators). Although a variety of genetic operators exist, the most commonly used ones are the reproduction, crossover and mutation operators.

3.9.1 Reproduction

The reproduction operator simply copies the selected individual (as it is) into the new population. It is similar to having one individual surviving into the next generation. The main advantage of this operator is that it allows the good genes of the individual to be carried over to the next generation unaltered. This can have a significant effect on the overall time taken by the GP system to converge. For example, Koza [1] allowed no more than 10% of the population to reproduce and this led to a 10% reduction in the required time to assign a fitness to the individuals in the population since there was no need to test the fitness of the reproduced individual as it was already known.

3.9.2 Crossover

The crossover operator introduces some variation in a population by producing new offspring consisting of genetic material from two different parents. Selection methods are usually used to decide which parents to select. In tree-based GP, *subtree crossover* is commonly used. In this form of crossover, a

random node from each parent tree is selected as a crossover point. To create offspring, the subtree which is rooted at the selected crossover point in the first parent is replaced with the subtree which is rooted at the crossover point in the second parent and vice versa. The process is depicted in Figure 3.4. During crossover, copies of the parents are used instead of the original individuals. In this way, the genetic material of the parents can be freely used if they are selected more than once.

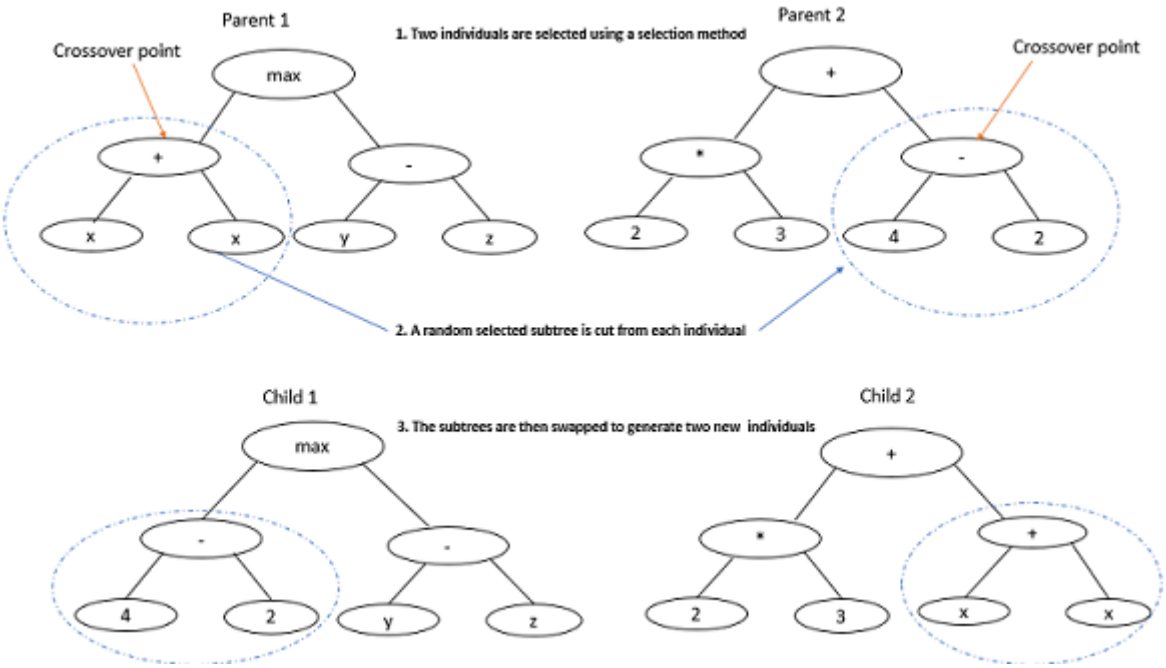

Figure 3.4. An example of subtree crossover with two generated offspring

Although crossover points are randomly selected, they are not usually selected with a uniform probability from the terminals and functions that make up the parent trees. For many GP trees, the average branching factor is at least two and this means that the majority of the nodes will be terminals. Therefore, using a uniform probability will lead to the exchange of very small amounts of genetic material since there is likely to be a higher chance of performing simple crossover operations such as swapping two terminals or small subtrees. To address this issue, Koza [1] proposed that functions should have a higher chance of being selected than terminals. In fact, it was proposed in his initial work on GP that functions should be chosen at least 90% of the time with terminals chosen the remaining 10%.

3.9.3 Mutation

The mutation operator is another genetic operator that introduces some variation in a population by producing new offspring. Unlike the crossover operator, the mutation operator only requires one parent.

In tree-based GP, the mutation operator generates new offspring by substituting a subtree rooted at a randomly selected mutation point with another randomly generated subtree. This process is commonly referred to as *subtree mutation* and it is depicted in Figure 3.5.

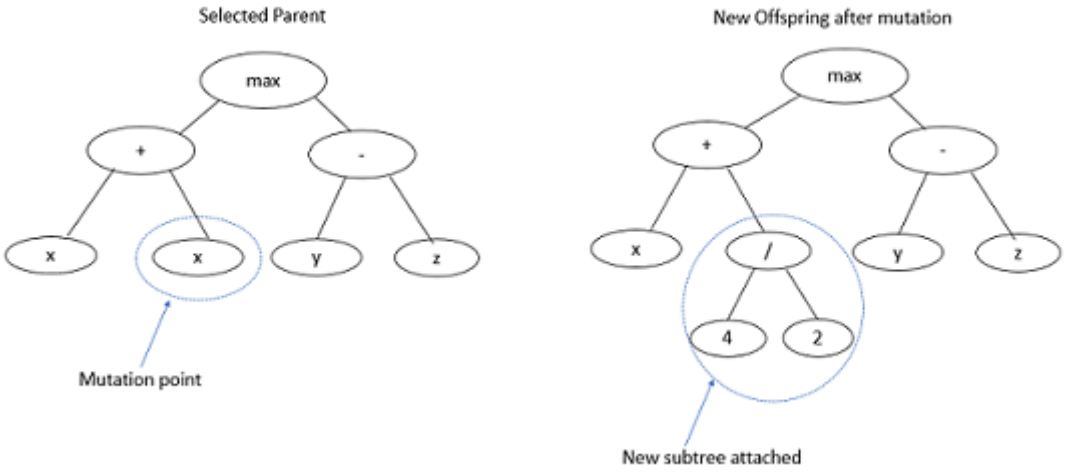

Figure 3.5. An example of subtree mutation

Another commonly used form of mutation is *point mutation* where a node is randomly selected and the primitive stored in the node is simply replaced with another primitive of the same arity randomly selected from the primitive set. If a primitive with the same arity does not exist in the primitive set then nothing happens to the selected node although other nodes may still be mutated. Point mutation unlike subtree mutation therefore allows for multiple nodes to be mutated in a single application [29].

The choice of which genetic operator to use is determined by some *operator rate* which is the probability of applying the genetic operator. Typically, the crossover operator is applied with the highest probability. The probability of applying the mutation and reproduction operator is often much smaller (usually around the 1% region). The *operator rates* are discussed in more detail in Section 3.11.

3.10 POPULATION REPLACEMENT

A population replacement strategy defines which parents and children survive into the next generation. There are two commonly used population replacement strategies in GP, namely, steady-state and generational. In the steady state strategy, a specified number of individuals in the current population is replaced by an equivalent number of offspring. The generational strategy on the other hand replaces

the current population with a new one generated after the application of crossover, reproduction and mutation operators. The generational strategy is much easier to implement than the steady state approach mainly due to the fact that it has less computational overheads. For example, the steady state approach may introduce computational overheads when determining and selecting the optimal number of individuals to replace.

3.11 GP CONTROL PARAMETERS

A GP algorithm run requires the user to specify a number of parameters for controlling the run. These parameters are often referred to as control parameters and some of the most important ones are:

- Size of population: The population size has a major impact on the success of a GP run. A significantly large population size increases the chances of evolving a solution as it enables greater exploration of the search space at every generation. Koza [1] proposes that the population size should be at least 500 or more. In general the size of the population may differ depending on how complex the problem is. Complex problems generally need larger population sizes.
- Number of generations: The number of generations also has an effect on the GP algorithm's ability to evolve a solution. Choosing a good value is important so that the algorithm converges within specified the number of generations. Too small a value will result in the algorithm stopping before it converges and will not reach an optimum. Too large a value will result in unnecessary runtime as once the algorithm converges no matter how many more generations it is run for there will be no change/improvement. The greater the number of generations, the higher the chances that a solution will be evolved. A good value typically ranges from 10 to 50 generations since the most productive search happens in the earlier generations and a solution is unlikely to be found in later generations if it is not found then [29].
- Initial population generation method : This parameter allows the GP user to set the initial population generation method. As mentioned in Section 3.6, the three methods commonly used to generate an initial population of individuals are the *full*, *grow* and *ramped-half and half*.
- Selection method : This parameter allows the GP user to specify the method for selecting individuals to be used to generate new offspring. The two most popular methods in GP are the *fitness proportionate* and *tournament* selection methods. However, most GP algorithms

use tournament selection rather than fitness proportionate selection due to its simplicity and efficiency. The tournament size mostly ranges from 2 to 10 individuals.

- Genetic operator rates : This parameter allows the GP researcher to specify the application rates for the genetic operators. A decision however has to be made with regard to whether the GP algorithm should have higher rates for exploring the search space (i.e. higher mutation rate) or for exploiting promising areas of the search space (i.e. higher crossover rate). Although the decision on the application rates to use is problem dependent, most GP algorithms have a higher application rate for the crossover operator. In their work, Poli et al.[29] also propose that 80% - 90% of the individuals in the population undergo crossover with the remaining individuals undergoing either mutation or reproduction.
- Other parameters include the maximum depth for tree-based GP. This parameter helps to limit the size of the trees generated by the GP system. This is particularly helpful in reducing the amount of computational resources required to run a GP system.

3.12 TERMINATION CRITERIA

The final preparatory step is the specification of a termination criterion for the GP run. The most commonly used termination criteria is the maximum number of generations specified for the GP run or some problem-specific success rate (e.g. a 99% classification accuracy in the case of a classification problem). It is also possible to specify a method for designating the result of a GP run after termination. For most problems, the best individual found so far is usually designated as the result of the GP run but one can also return additional data or individuals if necessary.

3.13 SUMMARY

This chapter presented an overview of GP, a very powerful evolutionary algorithm that is capable of automatically evolving computer programs to solve various types of problems. The basic terminology and tools used in GP were introduced. In particular, a discussion on the initial tree-based representation in which GP individuals (computer programs) are represented as syntax trees composed of terminal and function nodes was presented. Terminal nodes represent the inputs required by the computer programs while the function nodes represent the functions (e.g. +, -, *) to be applied to the terminal nodes. The set of all terminal and function nodes in a GP system is referred to as the terminal and function sets

respectively. Both the function and terminal sets are problem dependent and once these have been identified, the next step involves the generation of an initial population of syntactically valid computer programs (individuals). The three commonly used methods for generating the initial population, namely *full*, *grow* and *ramped-half and half* were discussed. In addition, a discussion on the concept of *fitness of a GP individual* i.e. the degree to which the individual solves the problem being addressed, was also presented. Furthermore, a discussion on the evolutionary process starting with an overview of the two methods commonly used in GP to select individuals(parents) to participate in the production of offspring for future generations and proceeding to discuss the genetic operators used to evolve a population in an attempt to improve its fitness. The basic GP algorithm was also introduced and this was followed by a discussion on the preparatory steps necessary for the GP algorithm run.

Despite some criticisms of GP [33], its flexibility has made it a very popular technique for solving hard computational search problems. In fact, GP has proven to be effective for the generation of hyper-heuristic approaches to solve various combinatorial optimization problems [34], [35]. However, in recent years, another technique called grammatical evolution (GE), itself a variant of GP, has become more popular especially in the domain of automated design of heuristics to solve combinatorial optimization problems. The next chapter discusses grammatical evolution.

CHAPTER 4 Grammatical Evolution

4.1 INTRODUCTION

Since its introduction, GP has enjoyed widespread use and popularity as a technique for automatically generating computer programs to solve hard computational problems. However, the initial version of GP used Lisp as the target language and this became a challenge for many other researchers who preferred to use other programming languages. In addition, the tree-based representation lacked a clear genotype to phenotype distinction which is present in living organisms. Furthermore, as a random searching technique, GP was found to be susceptible to redundant code, i.e. introns and bloat. As a way of tackling these issues, O'Neil and Ryan [25] proposed grammatical evolution (GE), an algorithm similar to GP, but one that allows for the evolution of complete computer programs in an arbitrary programming language and also facilitates the genotype to phenotype mapping of individuals. Unlike GP, the evolutionary process in GE is performed on variable-length binary strings rather than the actual computer programs. A mapping process using a BNF grammar governs the conversion of the binary string (genotype) into a syntactically correct computer program (phenotype). By enforcing a structure, the size of the search space is also reduced.

This chapter presents an overview of GE. Firstly, it introduces and discusses the three main components of a GE system, namely the grammar, search engine and a mapper. Thereafter, a discussion on the GE algorithm and some of its applications is presented. The chapter ends with a summary of some of the main discussion points.

4.2 GE SYSTEM COMPONENTS

A typical GE system is depicted in Figure 4.1 and is generally composed of three parts, namely the grammar, search engine and mapper.

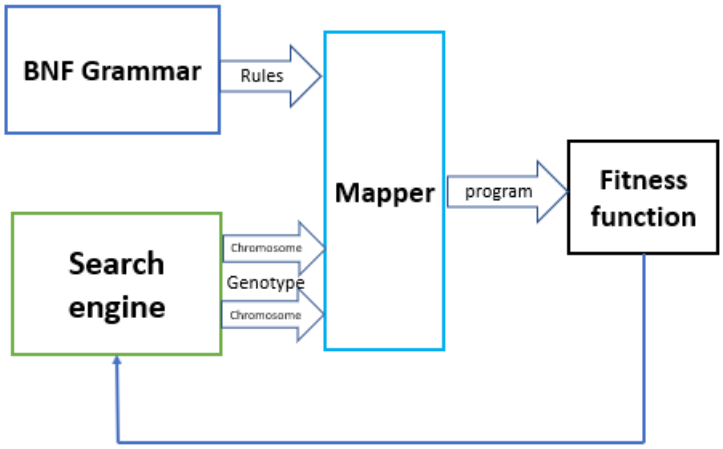

Figure 4.1. Components of a GE system

The next sections discusses the three parts in more detail.

4.2.1 BNF Grammar

As mentioned earlier, GE uses a grammar to convert the variable length binary string (or genotype) into a computer program (or phenotype). In fact, almost all evolutionary algorithms that produce computer programs implicitly or explicitly use grammars. Grammars describe how the variables (terminals) and operators (functions) can be legally combined to generate executable programs. In this way, grammars incorporate problem domain knowledge. GE uses a grammar represented in Backus Naur Form (BNF). Formally, a BNF grammar is a four-tuple $\langle T, N, T, P \rangle$ where T represents the terminal set, N is the function set or set of non-terminals, S is the start symbol and a member of N, P is the set of production rules that map the elements of N to T. In some cases, more than one production rule can be used within a particular N. For such cases, the production rules are separated by the 'I' symbol. For example, the syntax of arithmetic expressions consisting of numerals composed of simple addition, subtraction, multiplication and division operators can be defined using the following BNF grammar: $T = \{+, -, *, /, 0, 1, 2, 3, 4, 5\}$, $N = \{expr, num, op, digit\}$, $S = \{expr\}$ and P can be represented as shown

in Figure 4.2. In the example, the non-terminals $\langle expr \rangle$ and $\langle num \rangle$ have two production rules each while $\langle op \rangle$ and $\langle digit \rangle$ have four and six production rules respectively. Production rules are indexed from 0.

$$
\begin{aligned}
\langle \text{expr} \rangle &\models \langle \text{num} \rangle \mid \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \\
\langle \text{op} \rangle &\models + \mid - \mid * \mid / \\
\langle \text{num} \rangle &\models \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{num} \rangle \\
\langle \text{digit} \rangle &\models 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5
\end{aligned}
$$

Figure 4.2. BNF grammar for arithmetic expressions

The design of a suitable grammar is often the first step taken when solving a problem using GE. This step is also considered the most important as it defines the search space for the solution to the problem. The definition of the grammar varies from one problem to the other.

4.2.2 GE Search engine

GE mostly uses a genetic algorithm (GA) [36], an evolutionary algorithm, as the search engine. Using a GA, a potential solution to the problem at hand (i.e. a chromosome or genotype) is represented by a 1-D variable length string array where the gene in each chromosome is an 8-bit binary number called a *codon*. An example of a genotype with 6 codons is depicted in Figure 4.3.

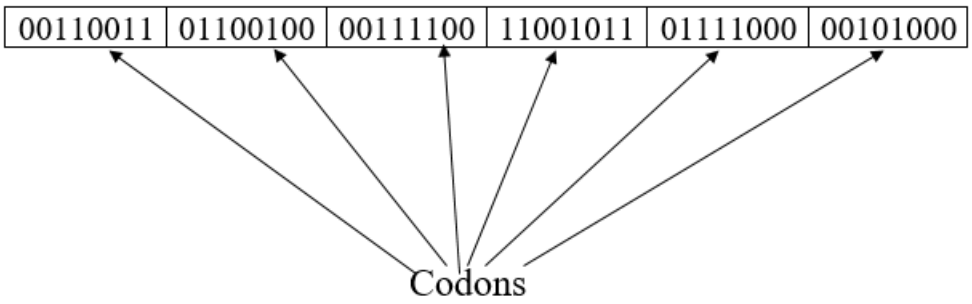

Figure 4.3. An example of a genotype with 6 codons

The mapper (discussed in the next section) uses the codon values to determine which production rule to select when converting one or more non-terminals into one or more terminals. As is the case with

all evolutionary algorithms, GA first generates an initial random population of chromosomes. The fitness (goodness) of each chromosome in the population is determined by executing its corresponding program (phenotype). One or two best performing programs may then be selected as parents to evolve new offspring (solutions) by applying the genetic operators (i.e. mutation, crossover and reproduction) to the selected parents. At each generation, the mapper evaluates the evolved solutions by converting them into their corresponding programs and then replacing the worst solutions in the population with the evolved solutions that have better fitness. This process usually continues until some termination criteria is met.

4.2.3 Mapper

The mapper is responsible for converting the genotype (chromosome) into a valid phenotype (computer program). It takes as input the BNF grammar and the genotype [25]. The genotype to phenotype conversion is performed using following mapping rule:

Mapping rule = (*decimal value of codon*) % (*number of production rules for the non-terminal*)

The mapping process starts by first converting the genotype's binary codon values to their corresponding decimal or integer values (see Figure 4.4) and then applying the mapping rule to convert all the non-terminals in the grammar into terminals starting with the start symbol. An example of the mapping process involving the genotype in Figure 4.3 and the BNF grammar in Figure 4.2 is depicted in Table 4.1.

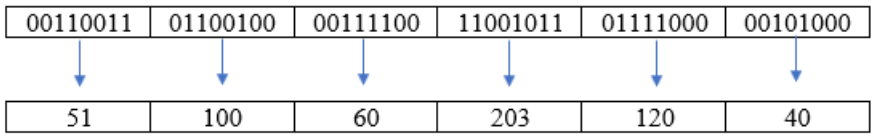

Figure 4.4. Conversion of binary codons to decimal values

Input	Number of choices	Mapping rule	Result
$\langle expr \rangle$	2	51 % 2 = 1	$\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$
$\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$	2	100 % 2 = 0	$\langle num \rangle$ $\langle op \rangle$ $\langle expr \rangle$
$\langle num \rangle$ $\langle op \rangle$ $\langle expr \rangle$	2	60 % 2 = 0	$\langle digit \rangle$ $\langle op \rangle$ $\langle expr \rangle$
$\langle digit \rangle$ $\langle op \rangle$ $\langle expr \rangle$	6	203 % 6 = 5	5 $\langle op \rangle$ $\langle expr \rangle$
5 $\langle op \rangle$ $\langle expr \rangle$	4	120 % 4 = 0	5 + $\langle expr \rangle$
5 + $\langle expr \rangle$	2	40 % 2 = 0	5 + $\langle num \rangle$
5 + $\langle num \rangle$	2	51 % 2 = 1	5 + $\langle digit \rangle$ $\langle num \rangle$
5 + $\langle digit \rangle$ $\langle num \rangle$	6	100 % 6 = 4	5 + 4 $\langle num \rangle$
5 + 4 $\langle num \rangle$	2	60 % 2 = 0	5 + 4 $\langle digit \rangle$
5 + 4 $\langle digit \rangle$	6	203 % 6 = 5	5 + 45
5 + 45			

Table 4.1. Example of the genotype to phenotype mapping process

In the example above, the mapper first selects the start symbol $\langle expr \rangle$ and then reads the first decimal codon value (51) of the genotype. According to the grammar, there are two production rules for $\langle expr \rangle$ and using the mapping rule i.e. 51 % 2 = 1, the second production rule ($\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$) is selected. But since this production rule still has non-terminals, the mapping rule is applied again starting with the leftmost non-terminal in the production rule but using the next codon value. This process is repeated until all the non-terminals in the grammar are converted to terminals. In the example (see Table 4.1), the complete expression containing only terminals is 5 + 45. In cases where all codon values are exhausted but the expression is still not complete (i.e. it still contains non-terminals), the codon values are wrapped around, i.e. the mapper continues the convertion process but starts reading the codon values again from left to right for a predefined number of iterations. If a complete expression cannot be generated even after the wrapping process, then the genotype is assigned the lowest fitness value. On the hand, if all the non-terminals in the grammar are replaced with terminals but not all codons are used, the mapper simply ignores the unused codon values (usually referred to as *introns*). The derivation tree for the mapping procss in Table 4.1 is shown in Figure 4.5.

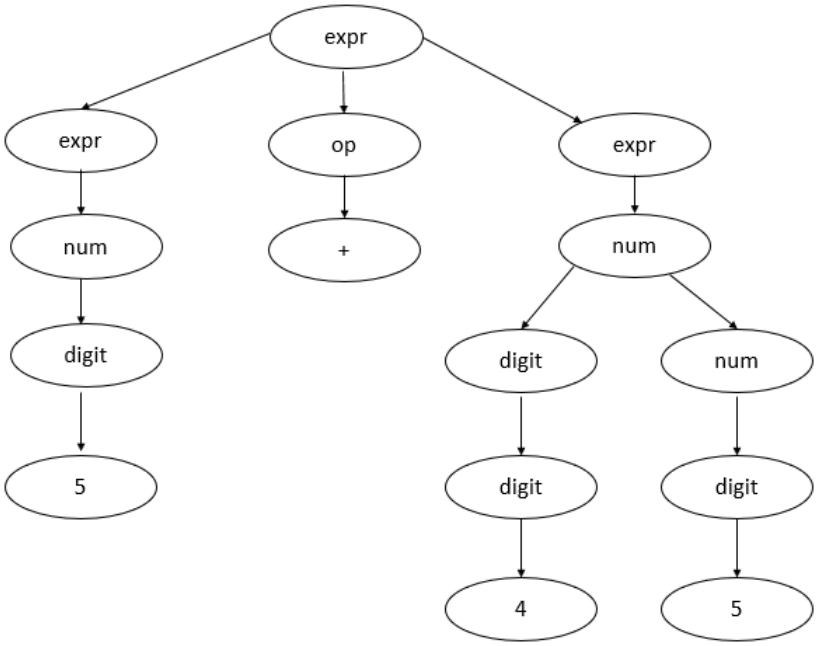

Figure 4.5. Derivation tree for the 5 + 45 expression

4.3 GE ALGORITHM

The GE Algorithm, just like all evolutionary algorithms, requires that an initial population of potential solutions be generated first. In GE, this step involves the generation of variable length binary strings (genotypes or chromosomes) which are then mapped to their respective programs using a BNF grammar. Each program is then executed and assigned a fitness value. Once the fitness of each program in the population has been established, the evolution process may begin. The pseudocode for a GE algorithm is shown in Algorithm 6.

Algorithm 6 Pseudocode for a GE algorithm

1: Generate an initial population of variable length binary strings
2: Map all the binary strings to their corresponding programs using a BNF grammar
3: Determine the fitness of each program in the population by executing it on some test or fitness cases
4: **while** termination criterion not met **do**
5: Select programs with the best fitness for regeneration using the tournament selection method
6: Create new programs for the next generation by applying genetic operators to selected programs
7: Execute each new program to determine its fitness
8: Replace all the programs in the old population with the new programs
9: **end while**
10: **return** program with the best fitness in the population as the best solution

4.3.1 Initial population

As can been seen from Algorithm 6, the first step in a GE algorithm involves the initialisation of a population of variable length binary strings (genotypes or chromosomes). The size of the initial population and the variable length limits are specified by the user. The genotype's codon values are used to select the appropriate production rules from the grammar to build a derivation tree (phenotype). The original GE algorithm used a random population initialization method but it was later discovered that this method presented a number of problems. Firstly, the derivation trees, although using the same genome lengths at the beginning of the evolution process, were not identical or did not have similar sizes after evolution. Secondly, it was observed that more than 50% of the genotypes failed to map to valid phenotypes in most cases. To overcome these problems, the Random initialisation with valids and no duplicates (RVD) method was proposed [37]. In this initialisation method, a randomly generated variable length genotype string is re-sampled if it does not map to a sequence of terminals only or if it maps to an already mapped to valid expression. This brute-force approach, although simple was found to be very effective according to the research on GE initialization methods conducted by Nicolau [37].

4.3.2 Genotype-Phenotype Mapping

This step involves the mapping of the genotypes to their corresponding phenotypes (computer programs). The mapping process has been discussed in Section 4.2.3.

4.3.3 Fitness Evaluation

The process of determining the fitness of the computer programs is similar to the one discussed for GP in Section 3.7. Basically, each computer program in the population is executed on a set of test or fitness cases and assigned a fitness value based on its performance.

4.3.4 Selection methods

As with GP, the two methods commonly used to select parents to generate offspring in GE are fitness proportionate and tournament selection. The methods are applied as discussed in Section 3.8.

4.3.5 Genetic operators

In GE, the most commonly applied genetic operators are crossover and mutation. One-point crossover is also the most widely used form of the crossover operator. In this type of crossover, a crossover point is randomly selected from the genotype of each of the two parents selected for reproduction. The codons on the right of the crossover point are then swapped between the two parents resulting in two children, with each child carrying some genetic material from both parents. Figure 4.6 illustrates the one point crossover operation.

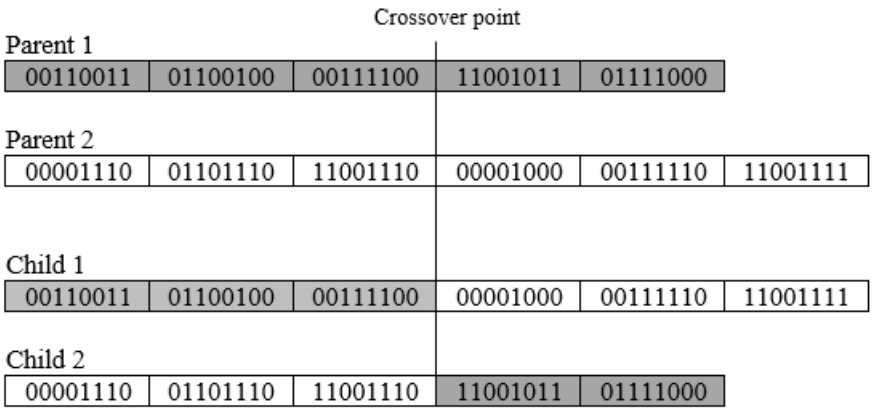

Figure 4.6. One-point Crossover

Bit flip mutation, where a random bit in a codon is flipped (i.e. 0 is replaced by 1 and vice versa), is the most commonly used type of mutation operator.

4.3.6 Population Replacement

The population replacement strategies used in GP (see Section 3.10) can also be used in GE. These strategies include the steady state and generational population replacement strategies.

4.3.7 GE Control Parameters

GE also requires some control parameters to be specified. The important ones are: size of the initial population, selection method, maximum number of generations, genetic operator rates, population replacement strategy, size of the chromosome and number of wraps (if allowed).

4.3.8 Termination criteria

The termination criteria used in GE are also similar to the ones discussed for GP in Section 3.12. Typically, a GE run is terminated after a specified number of generations or when the desired solution is obtained.

4.4 SUMMARY

This chapter presented an overview of GE, a grammar based variant of GP that is capable of evolving computer programs in any programming language. Unlike GP, GE provides a clear method of distinguishing between a genotype (chromosome) and a phenotype (computer program). The evolutionary process is also performed on a variable length binary string rather than the actual computer programs. The general GE system comprises three procedures, namely the grammar, search engine and mapper. The grammar describes how the variables (terminals) and operators (functions) can be legally combined to generate executable programs. The search engine is concerned with evolving the best programs for solving the problem at hand while the mapper is responsible for converting the genotype (chromosome) into a valid phenotype (computer program). GE has been widely used as a technique for automatic programming due to the simplicity with which it can represent computer programs as well as its ability to avoid problems inert to GP such as *code bloat*. Code bloat refers to the production of unnecessarily long, slow and wasteful programs.

GE, just like GP is a meta-heuristic. And as mentioned in Section 2.6.2, meta-heuristics although capable of obtaining high quality solutions for some problem instances do not generalize very well across different problem instances or domains. Furthermore, the need to redevelop the algorithms for every new problem instance is both time consuming and challenging. Due to the above-mentioned challenges, researchers have proposed the use of hyper-heuristics as more general search techniques for solving computationally hard problems such as combinatorial optimization problems. The next chapter discusses hyper-heuristics in more detail.

CHAPTER 5 Hyper-heuristics

5.1 INTRODUCTION

As was mentioned in Chapter 1, combinatorial optimization problems are associated with large and heavily constrained search spaces. Solving these problems usually requires the design of suitable heuristics since exact techniques often fail to find good solutions in a reasonable amount of time. A variety of heuristics as well as meta-heuristics have since been proposed in the literature but these techniques do not generalize very well across multiple problem domains and therefore cannot be used as general search techniques. Hyper-heuristics were introduced as a way of producing more generalised solutions through the use of easier, cheaper and general algorithms than the problem specific meta-heuristics [38]. To do this, hyper-heuristics were designed to explore the heuristic space instead of the solution space during problem solving [39]. This was achieved by letting the hyper-heuristics select the most appropriate heuristics to use from those available or by letting the hyper-heuristics generate new heuristics from the components making up the heuristics.

Due to their generality and ability to find good solutions, hyper-heuristics have been successfully applied to solve combinatorial optimization problems in various domains such as scheduling, routing, resource management and production planning [40]. This chapter presents a general overview of hyper-heuristics. It first introduces the definition of hyper-heuristics adopted in this research and thereafter provides a discussion on the two abstract components of a typical hyper-heuristic, namely the low and high level components. A discussion on the different classes of hyper-heuristics is also presented but the chapter focuses more on hyper-heuristics which generate perturbative heuristics as this is the class of hyper-heuristics that is relevant to this research.

5.2 DEFINITION

There are many definitions of the term hyper-heuristic in the literature [41],[42]. Cowling et al. [41], define hyper-heuristics as *"heuristics to choose heuristics"*. In this context, a hyper-heuristic is considered as a high-level approach that can select, apply and generate the most appropriate heuristics from a number of low-level problem specific heuristics at each decision point in the problem solving process [43]. Burke et al. [42] defines hyper-heuristic as *"an automated methodology for selecting or generating heuristics to solve hard computational search problems"*. From this definition, hyper-heuristics are categorised in terms of the processes of *heuristic selection* and *heuristic generation*. Heuristic selection is concerned with developing approaches for selecting or choosing existing low-level heuristics while heuristic generation is concerned with developing approaches for generating new heuristics from primitive components of existing low-level heuristics. This research adopts the definition proposed by Burke et al. [42] as it captures the idea of developing methodologies for automating the design of heuristics which is the main aim of the work presented in this dissertation.

The next section looks at a typical hyper-heuristic and discusses its main components.

5.3 COMPONENTS OF A TYPICAL HYPER-HEURISTIC

As mentioned earlier, the main aim of hyper-heuristics is to design general methodologies that can be used to solve different types of hard computational search problems such as real-world problems. In order to do this, a hyper-heuristic separates a low-level component containing problem domain specific information such as problem-specific heuristics from a high-level component containing some meta-heuristic to search through the space of available low-level heuristics [24]. The high-level component is therefore problem independent. The next section discusses the low-level component in more detail.

5.3.1 Low-level component

The low level component is problem specific and in addition to some problem specific heuristics, may also contain additional information such as the problem objective function and problem representation type [24]. The problem-specific heuristics found in the low-level component are typically referred to as

low-level heuristics (llhs) [24]. These heuristics are categorised as either *constructive* or *perturbative* [44].

Constructive heuristics build a solution from scratch and iteratively add elements to it to obtain a feasible solution [43]. As such, constructive heuristics are mainly used to construct initial solutions to problems. For example, constructive heuristics are used in examination timetabling problems to construct initial feasible timetables (which are the solutions) based on a measure of how difficult it is to schedule each examination. The largest degree, largest enrolment degree, saturation degree, largest weighted degree and largest colour degree heuristics are some of the most commonly used constructive heuristics for solving examination timetabling problems [45]. In general, constructive heuristics are used in order to produce initial solutions that serve as better starting points for many optimization techniques than the initial solutions constructed randomly.

Although constructive heuristics may produce better solutions than those constructed randomly, the quality of these solutions is not always the best and in most cases can be improved further by making some perturbations to the initial solutions. This is generally what perturbative heuristics do. In a nutshell, perturbative heuristics iteratively modify an existing solution in an attempt to improve its quality. The modifications may be simple operations such as moving, swapping, adding and deleting some elements of the solution [46]. By iteratively making changes to an existing solution, pertubative heuristics have the same effect as the move operators commonly used when exploring the neighbourhood of a search point in local search. Hence, they are also known as local search operators.

During problem solving, the low-level heuristics often compete with each other and most hyper-heuristics employ techniques such as reinforcement learning, local search, case based reasoning and genetic programming etc., to determine the most suitable low-level heuristic to apply at each decision point when constructing or improving a solution [44].

5.3.2 High-level component

The high-level component usually comprises a search algorithm that searches the space of available low-level heuristics [43]. It is problem independent and often contains minimal information such as

the number of available low-level heuristics and other non problem domain statistical information [40]. The high-level component can therefore be easily applied to different problem domains. In addition, different meta-heuristics may be employed as search algorithms depending on whether the hyper-heuristic aims to select or generate new heuristics [43]. GP and GE are the techniques commonly employed by hyper-heuristics which aim to generate new heuristics [34],[40].

5.4 CLASSIFICATION OF HYPER-HEURISTICS

Based on the definition of hyper-heuristics adopted in Section 5.2 and the categories of low-level heuristics established in Section 5.3.1, hyper-heuristics are typically classified as selection constructive, selection perturbative, generation constructive and generation perturbative [24], [44].

Selection constructive hyper-heuristics, as the name suggests, intelligently select the most suitable low-level constructive heuristic to apply at each decision point during the construction of an initial solution [24]. As these are selection hyper-heuristics, they rely on pre-existing low-level constructive heuristics and usually employ techniques such as case-based reasoning, adaptive methods, population-based methods, local search methods and hybrid methods to perform the selection [24]. Selection constructive hyper-heuristics have been applied to various combinatorial optimization problems such as examination timetabling, course timetabling, 1-D bin packing and job shop scheduling problems [44].

Selection perturbative hyper-heuristics on the other hand intelligently select the most suitable low-level perturbative heuristic to apply at each decision point during the solution improvement process [24]. As these are also selection hyper-heuristics, they rely on pre-existing low-level perturbative heuristics and can perform either a single-point or multi-point search on the heuristic space [24]. A single-point search hyper-heuristic consists of two components, a *heuristic selection* component and a *move acceptance* component [44]. The heuristic selection component selects the low-level perturbative heuristics to apply while the move acceptance component decides whether to accept or reject the moves made by the selected low-level perturbative heuristics. Different techniques may be employed by the hyper-heuristic for both heuristic selection and move acceptance [44]. Multi-point search hyper-heuristics use population-based algorithms such as evolutionary algorithms to explore the heuristic space and

therefore do not require separate components for heuristic selection and move acceptance since the processes are done naturally.

Generation constructive hyper-heuristics generate new low-level constructive heuristics for a given problem domain [24]. The new heuristics are mainly generated from either existing low-level heuristics or primitive components of the existing low-level heuristics or problem domain characteristics mostly using GP (see Chapter 3). The generated heuristics may be *disposable* or *reusable* [43]. Disposable heuristics are used to solve a problem instance while reusable heuristics may be used to solve more than one problem instance. Reusable heuristics are generated from a training set consisting of one or more problem instances and a test set consisting of unseen problem instances.

Generation perturbative hyper-heuristics generate new low-level perturbative heuristics for a given problem domain [24]. The new low-level perturbative heuristics are also generated from existing low-level heuristics or primitive components of the existing low-level heuristics mostly using GP. In addition, various move acceptance criteria may be used in combination with the existing low-level heuristics to generate new heuristics [40].

Burke et al. [42] further distinguishes between a learning and a non-learning hyper-heuristic. In the context of selection hyper-heuristics, a learning hyper-heuristic uses some feedback information on the performance of each selected low-level heuristic from the search process. A non-learning hyper-heuristic does not keep track of the previous heuristic performance. It simply selects a low-level heuristic either uniformly at random or in a prefixed order from some existing pool. The learning can be *online* or *offline*. In online learning, the hyper-heuristic learns while solving a problem instance and as such it uses some task-dependent properties to determine the most suitable heuristic to apply. In offline learning, the hyper-heuristic gathers knowledge from a set of training instances and the trained hyper-heuristic is then expected to solve other unseen problem instances. The notion of online and offline learning can also be extended to generation hyper-heuristics. In this context online and offline learning hyper-heuristics are analogous to disposable and reusable hyper-heuristics respectively.

Other categories of hyper-heuristics include hybrid methodologies that either combine constructive with perturbative heuristics or heuristic selection with heuristic generation [24].

As this research is concerned with generating perturbative heuristics to solve combinatorial optimization

problems, the rest of the discussion in this chapter will be limited to generation perturbative hyper-heuristics.

5.5 GENERATION PERTURBATIVE HYPER-HEURISTICS

Perturbative heuristics, as mentioned in Section 5.3.1, play a crucial role in improving the quality of solutions obtained by constructive heuristics. For most problems, these heuristics are designed manually by domain experts who mostly rely on experience and intuition. This manual design process is however very challenging and time consuming as it requires a deep understanding of the problem domain. Hence, some initiatives have been proposed by researchers with the ultimate aim of automating the heuristic design process and thereby helping to reduce the burden on human experts. Furthermore, automating the design process is likely to lead to the discovery of new heuristics that human experts would otherwise not be able to think of.

Hyper-heuristics have been used to generate perturbative heuristics to solve various problems [34], [35], [47], [40]. These hyper-heuristics commonly referred to as generation perturbative hyper-heuristics employ either GP or GE to evolve the new perturbative heuristics. This is usually done by combining pre-existing low-level perturbative heuristics or primitive components of the low-level perturbative heuristics with conditional branching operators and iterative constructs [44]. The various types of low-level perturbative heuristics that have been generated by these hyper-heuristics include local search operators, meta-heuristics and algorithms [44].

5.5.1 Generation of Local Search Operators

Local search operators are operators which attempt to improve the quality of a candidate solution by iteratively looking for better solutions, within its direct neighbourhood, to replace it [48]. The neighbourhood of a candidate solution refers to the set of solutions that can be generated by making small changes (or perturbations) to the candidate solution such as moving, swapping, adding and deleting some elements of the solution. Local search operators have largely been evolved using GP or its variations.

Among the earlier works to generate local search operators was the study conducted by Fukunaga [35]. In the study, GP was used to generate local search operators for solving the Boolean satisfiability (SAT) problem using components of existing and well-known human-designed perturbative heuristics such as Novelty, WALKSAT and GSAT. The generated operators outperformed the state-of-the-art algorithms at the time. In a similar study, Bader-el-den and Poli [49], used a grammar-based strongly typed GP to generate local search operators for solving 3-SAT problems. The grammar was used to specify how the components of existing human-designed SAT perturbative heuristics (i.e. GSAT, HSAT, GWSAT, and WalkSAT) should be combined. The generated operators obtained solutions that were on par with the best SAT solvers.

Apart from GP, GE has also been used to generate local search operators. For example, Burke et al. [50] used GE to generate local search operators for solving the 1-D bin packing problem. The generated operators were mainly *ruin* (where some pieces were removed from the solution) and *recreate* (the earlier removed pieces were repacked with a faster constructive heuristic) operators which produced good results for some problem instances. Sabar et al. [40] also used GE to generate local search operators for solving the examination timetabling and vehicle routing problems. In the study, various human-designed perturbative heuristics (referred to as neighborhood structures in the study) were combined with different move acceptance criteria such as improving only moves(IM), all moves (AM), simulated annealing (SA), exponential monte carlo (EMC), great deluge (GD) and naive acceptance (NA). An adaptive memory mechanism was used to maintain a diverse population of solutions which was regularly updated during the heuristic generation process. The generated operators obtained solutions that were either on par or better than those obtained by the state-of-the-art techniques for most of the problem instances in the two problem domains. Another study by Stone et al. [38] used GE to generate local search operators for the multi-dimensional knapsack and travelling salesman problems. The approach in the study was however only applicable to problems that could be represented as a graph. Nevertheless, the evolved operators obtained good solutions for some problem instances in the two problem domains.

5.5.2 Generation of Meta-heuristics

Apart from generating local search operators, hyper-heuristics have also been used to evolve meta-heuristics. For example, Keller and Poli [47] used linear GP to generate meta-heuristics for solving

the travelling salesman problem. In the study, the meta-heuristics were evolved from meta-heuristic components and pre-existing human-derived low-level perturbative heuristics for the problem domain. A grammar was used to specify the syntax of valid meta-heuristics and it incorporated conditional branching constructs (e.g. IF2-CHANGE) as well as iterative constructs (e.g. REPEAT-UNTIL-IMPROVEMENT). The evolved meta-heuristics performed better than the hill-climbing algorithm for one of the problem instances.

5.5.3 Generation of Algorithms

The other type of perturbative heuristics that have been evolved by hyper-heuristics are algorithms. In particular, algorithms have been evolved to solve the travelling salesman and automatic clustering problems [51]. The research in this domain have mainly employed GP to evolve algorithms from problem domain specific terminals and standard algorithm constructs such as conditional branching constructs, iterative constructs (while loops), if-then-else statements and logical operators.

In general, generation perturbative hyper-heuristics have been very successful in generating perturbative heuristics to solve various computationally hard problems such as combinatorial optimization problems. Despite this success, the domain has not been as well-research and very few works have actually been conducted in the area. In fact, the literature shows that the majority of the research effort in hyper-heuristics has been focussed on the selection and generation of constructive heuristics rather than perturbative heuristics. A detailed critical analysis of the literature is presented in Chapter 6.

5.6 SUMMARY

The chapter presented an overview of hyper-heuristics, a more general methodology for solving combinatorial optimization problems. Hyper-heuristics were defined as *" automated methodologies for selecting or generating heuristics to solve hard computational search problems"*. In a typical hyper-heuristic, the low-level component comprising domain specific low-level heuristics (and other domain specific information) is separated from a high-level component comprising a search algorithm which performs a search on the space of low-level heuristics. The low-level heuristics may be *constructive* (i.e. heuristics that build a solution from scratch and iteratively add elements to it to obtain feasible solution) or *perturbative* (i.e. heuristics that iteratively modify an existing solution

to improve its quality). Hyper-heuristics can be generally classified into four main classes, namely selection constructive, selection perturbative, generation constructive and generation perturbative hyper-heuristics depending on whether they select or generate either constructive or perturbative heuristics. In most cases, generation perturbative hyper-heuristics employ either GP or GE to generate new low-level perturbative heuristics to solve combinatorial optimization problems. The various types of low-level perturbative heuristics that have been generated include local search operators, meta-heuristics and algorithms.

The next chapter presents a critical analysis of the literature discussed so far and also introduces the methodology used to met the objectives outlined in Chapter 1.

CHAPTER 6 Methodology

6.1 INTRODUCTION

This chapter discusses the research methodology used to meet the objectives outlined in Chapter 1 of this dissertation. Section 6.2 presents a critical analysis of the related work in the literature and provides reasons for undertaking this research. Section 6.3 discusses the research methodologies that were adopted for the study while section 6.4 discusses the problem domains the proposed approach was evaluated on. The performance measures used to evaluate the approach are discussed in section 6.5 and the technical specifications of both the software and hardware used in the study are presented in section 6.6. The chapter summary is provided in section 6.7.

6.2 CRITICAL ANALYSIS OF LITERATURE SURVEY

From the literature survey on hyper-heuristics presented in Chapter 5, it can be observed that generation hyper-heuristics mostly employ GP or GE to generate heuristics. Although these hyper-heuristics have been successfully used to generate good quality heuristics for solving various combinatorial optimization problems (e.g. 1-D bin packing, Boolean satisfiability (SAT), vehicle routing, examination timetabling and travelling salesman problems), the following general observations can be made about generation hyper-heuristics.

- The vast majority of the research effort in the domain of generation hyper-heuristics has been directed towards the development of approaches that generate constructive heuristics rather than those that generate perturbative ones. There has been few works in the area of generation perturbative hyper-heuristics.

- Most of the proposed generation perturbative hyper-heuristics rely on human-derived low-level heuristics (or primitive heuristic components) to generate new perturbative heuristics. This restricts their applicability to problem domains where the human-derived heuristics (or primitive heuristic components) may not be available.
- The few generation perturbative hyper-heuristics that have been proposed to generate perturbative heuristics from scratch either impose special restrictions on problem representation (e.g. problems must be represented as graphs) or are tailored to a single problem domain (e.g. 1-D bin packing problem). This raises the question as to what extent these hyper-heuristics generalise to other problem domains.

Motivated by the observations above, this research proposes a new approach for automating the generation of perturbative heuristics. The proposed approach will take several heuristic components (i.e. basic operations and components of the solution) as input and will automatically generate perturbative heuristics by selecting the most suitable combination of the heuristic components using GE. GE is chosen over GP due to the simplicity with which it can represent heuristic components as well as its ability to tackle the problem of code bloat usually encountered in traditional GP. The main differences between the proposed approach and other generation perturbative hyper-heuristics in the literature are:

1. The proposed approach will generate new problem specific perturbative heuristics from scratch. It will not rely on any existing human-derived low-level heuristics (or heuristic components) to do so.
2. The proposed approach will not impose any special restrictions on how the problem should be represented. The approach will instead focus on identifying and using the components of the solution. It will not matter how the problem is represented.
3. The proposed approach will not be tailored to a single domain. It will be possible to apply the same approach to several problem domains. The system user will only need to specify the solution components for the new problem.
4. The proposed approach will be easy to extend and customise so that a wider range of perturbative heuristics can be generated. For example, it will be possible to specifically tailor the approach to a single domain in order to generate more problem domain specific heuristics. To do so, the system user will simply add the necessary basic operations and solution components.

The next section discusses the research methodologies adopted in this study.

6.3 RESEARCH METHODOLOGIES

A wide range of methodologies for conducting research in computer science have been proposed in the literature [52], [53]. The most commonly used methodologies are *mathematical proofs*, *proof by demonstration*, *empiricism* and *hermeneutics*. The *mathematical proof* methodology applies formal mathematical reasoning techniques to evaluate a hypothesis. In this methodology, a theory that some good property will hold in a given system is first established and attempts are made to either verify or refute the theory using mathematical arguments. The *proof by demonstration* methodology requires that an algorithm be initially developed. The developed algorithm is then iteratively tested and refined until the desired result is obtained or the obtained result cannot be improved any further. At each iteration, if the desired result is not obtained the reasons for failure are identified and corrected so that the algorithm can gradually move towards the desired result. The proof by demonstration methodology may also be applied to stochastic algorithms. The *empiricism* methodology is mainly used to evaluate a hypothesis and can be summarised by four stages namely, hypothesis generation, method identification, result compilation and conclusion. The *hermeneutics* methodology was adopted from the field of sociology to deal with concerns about whether there was a relationship between the mathematical models developed in computer science and the reality they intend to represent. In practice, this methodology involves deploying and observing the operation or use of the artefact in its intended working environment.

The main aim of this research is to design an approach that can be used to automatically generate good quality perturbative heuristics for solving combinatorial optimization problems. In order to achieve this aim, a GE approach will be developed and implemented. The viability of the proposed approach will be determined by testing the generated perturbative heuristics on benchmarks sets from three well-known problems domains. In particular, the performance of the generated perturbative heuristics will be compared to that of the human-designed perturbative heuristics commonly used for the benchmark sets as well as that of the perturbative heuristics generated by existing generation perturbative hyper-heuristics in the literature. Thus, the *proof by demonstration* and *empiricism* research methodologies are considered to be appropriate in this research. The next section describes how the proof by demonstration research methodology is used to achieve some of the objectives outlined in Chapter 1.

6.3.1 Proof by Demonstration

The proof by demonstration research methodology will be used to meet the following objectives:

- *To develop an approach that automatically generates perturbative heuristics for more than one problem domain using grammatical evolution.*
- *Test the generality of the proposed approach on three different problem domains.*

As per proof of demonstration methodology requirement, a GE approach will be developed and implemented. The developed approach will then be tested on some benchmark sets from three problem domains (discussed in Section 6.4). The decision on whether refinements to the developed approach should be made or not will be depend on how it performs with respect to the performance measures described in Section 6.9.

As this is a GE approach, it will require some initial decisions to be made among which the identification of heuristic components and the design of a suitable grammar are the most challenging. The design of the initial system will be based on an analysis of the currently existing literature (see Section 6.2) and the GE control parameters will be adopted from the related studies that produced good results. Once a suitable grammar has been designed and the GE parameters determined, the GE approach will be iteratively tested and refined until the desired results are obtained or the system cannot be improved any further. The steps that will be performed using the proof by demonstration methodology can be summarised as follows:

1. Design and implement a GE-based approach for the automated generation of perturbative heuristics.
2. Test the performance and generality of the proposed approach by applying the generated perturbative heuristics on initial solutions constructed for each instance in the benchmark sets. A minimum of 30 runs will be performed for each instance during the training phase due to the stochastic nature of GE. The best heuristic identified after the training phase will be used during the testing phase.
3. If the generated heuristic fails to improve the initial solutions, then changes will be made to following aspects of the GE approach:

- Grammar elements which specify the heuristic components and their valid combinations. Changes may also include designing a new grammar.
- GE parameters such as application rates for genetic operators, selection methods and number of generations.

4. The refined approach will be re-tested and further refined if any failures occur until the performance of the generated heuristics improves.

Apart from the proof by demonstration research methodology, this study will also make use of the empiricism research methodology. The empiricism methodology will be used to achieve the remaining objectives of this study. The next section describes how the empiricism research methodology will be used to achieve these objectives.

6.3.2 Empiricism

The empiricism research methodology will be used to meet the following objectives:

- *Compare the performance of the perturbative heuristics generated by the proposed approach to that of the human-designed move operators.*
- *Compare the performance of the perturbative heuristics generated by the proposed approach to that of the perturbative heuristics generated by existing generation perturbative hyper-heuristics.*

The empiricism research methodology is characterised by four main stages and these are:

1. Hypothesis generation: This stage identifies the ideas that will be tested by the research. The following hypotheses will be tested in this research.

   - **Hypothesis H1:** The perturbative heuristics generated in this study obtain better solutions than the human-derived ones.
   - **Hypothesis H2:** The perturbative heuristics generated in this study obtain better solutions than the perturbative heuristics generated by existing generation perturbative hyper-heuristics in the literature.

2. Method identification: This stage identifies the techniques that will be used to establish the hypothesis. In this research several experiments will be carried out for each of the generated perturbative heuristics. Each experiment will involve applying the best heuristic evolved during the training phase to test instances from each benchmark set. The solutions obtained by the heuristics will be captured.
3. Result compilation: This stage is concerned with the presentation and compilation of the results gathered in the previous stage. In this research, the Friedman test [54] will be used to test the statistical significance of the obtained results (see Section 6.9).
4. Conclusion: This stage states whether the hypothesis is accepted or rejected.

The proposed approach will be tested on benchmark sets from three problem domains. These benchmark sets have been extensively used by hyper-heuristic researchers in the field and will be used for comparison purposes. The next section discusses the problem domains.

6.4 PROBLEM DOMAINS

The proposed GE approach will be tested on three different problem domains, namely examination timetabling, vehicle routing and boolean satisfiability. These problem domains are discussed in more detail in the following sections.

6.4.1 Examination timetabling problem

The examination timetabling problem is a well-known combinatorial optimization problem that involves assigning examinations to a limited number of timetable periods and rooms such that all the hard constraints of the problem are satisfied and the number of soft constraints violated is as minimal as possible. In this study, the ITC 2007 examination timetabling benchmark set [1] is used. The benchmark set was selected due to its complexity as it incorporates an increased number of real world constraints. The benchmark set has also been widely used to test the performance of most generation perturbative hyper-heuristics in the literature.

The hard constraints for the benchmark set are:

[1] The benchmark set is available from http://www.cs.qub.ac.uk/itc2007/index.htm

- A student must not sit for more than one examination at a time.
- The capacity of the room must not be exceeded.
- The duration of the examination must not exceed the duration of the period.
- All period-related constraints must be satisfied e.g. assigning two examinations in the same period.
- All the room related constraints should be satisfied e.g. assigning only one examination in a particular room.

The following are the soft constraints:

- The number of students taking two consecutive examinations in a row should be as minimal as possible.
- The number of students taking two examinations on the same day should be as minimal as possible.
- The examinations must be well spread out for the students.
- The number of examinations with mixed durations within a period must be minimised.
- The larger examinations should appear at the beginning of the timetable.
- The usage of certain periods must be kept to a minimum.
- The usage of certain rooms must be kept to a minimum.

A timetable is feasible if all the examinations are assigned to a period and room such that all hard constraints are satisfied. The aim is to obtain a timetable with zero hard constraint violations and a minimum number of soft constraint violations. Each soft constraint violation has a numerical weighted cost [55]. The *objective value* for a feasible timetable is therefore the sum of the weighted cost of the hard and soft constraints violated by the timetabling. A timetable which violates any hard constraint is heavily penalised by assigning it largest possible cost. The specification of the ITC 2007 benchmark set is presented in Table 6.1.

Instances	Periods	Exams	Rooms	Students	Conflict Density
1	54	607	7	7891	0.05
2	40	870	49	12743	0.01
3	36	934	48	16439	0.03
4	21	273	1	5045	0.15
5	42	1018	3	9253	0.009
6	16	242	8	7909	0.06
7	80	1096	15	14676	0.02
8	80	598	8	7718	0.05
9	25	169	3	655	0.08
10	32	214	48	1577	0.05
11	26	934	40	16439	0.03
12	12	78	50	1653	0.18

Conflict Density = number of potential conflicts / (number of exams). It is a measurement of the number of conflicts examinations i.e. how tightly the problem is constrained in terms of student enrolments [56].

Table 6.1. Specification of the ITC 2007 benchmark set showing the attributes of each instance.

6.4.2 Capacitated vehicle routing problem

The capacitated vehicle routing problem is another well-known combinatorial optimization problem that involves finding a set of routes with the minimum cost for serving a set of customers without violating any hard constraints. This study uses the Golden and Christofides benchmark sets [2]. The benchmark sets were selected for comparison purposes with currently existing generation perturbative hyper-heuristics in the literature.

Both benchmark sets have the following hard constraints:

- A vehicle must start and end at the depot after making all the deliveries.

[2]The benchmark sets are available from http://www.vrp-rep.org/datasets.html

- The total demand for each route must not exceed the vehicle capacity.
- Each customer must be visited only once and by exactly one vehicle in the route.
- The duration of each route must not exceed a global upper bound.

As mentioned earlier, the aim is to find a set of routes with the minimal cost. The *objective value* of a solution is equivalent to the sum of the cost of all the routes with the cost of each route calculated as the sum of the distance between customers on the route. The specifications of the Christofides and Golden benchmark sets are presented in Table 6.2 and Table 6.3 respectively.

Instances	Number of Vehicles	Vehicle Capacity	Customers	Max. length	Service time
1	5	160	51	∞	0
2	10	140	76	∞	0
3	8	200	101	∞	0
4	12	200	151	∞	0
5	17	200	200	∞	0
6	6	160	51	200	10
7	11	140	76	160	10
8	9	200	101	230	10
9	14	200	151	200	10
10	18	200	200	200	10
11	7	200	121	∞	0
12	10	200	101	∞	0
13	11	200	121	720	50
14	11	200	101	1040	90

∞ means there is no restriction on the maximum length that a vehicle may travel.

Table 6.2. Specification of the Christofides Benchmark set showing the attributes of each instance.

Instances	Number of Vehicles	Vehicle Capacity	Customers	Max. length	Service time
1	10	550	240	650	0
2	10	700	320	900	0
3	10	900	400	1200	0
4	12	1000	480	1600	0
5	5	900	200	1800	0
6	8	900	280	1500	0
7	9	900	360	1300	0
8	11	900	440	1200	0
9	14	1000	255	∞	0
10	16	1000	323	∞	0
11	18	1000	399	∞	0
12	19	1000	482	∞	0
13	27	1000	252	∞	0
14	30	1000	320	∞	0
15	34	1000	396	∞	0
16	38	1000	480	∞	0
17	22	200	240	∞	0
18	22	200	300	∞	0
19	33	200	360	∞	0
20	41	200	420	∞	0

∞ means there is no restriction on the maximum length that a vehicle may travel.

Table 6.3. Specification of the Golden Benchmark set showing the attributes of each instance.

6.4.3 Boolean satisfiability problem

The boolean satisfiability problem is yet another combinatorial optimization problem that involves determining whether an assignment of values to variables for a given boolean expression exists such that the expression evaluates to true [57]. The expression is represented as a conjuction of clauses. Each clause is a disjunction of variables. The expression is *satisfiable* if an assignment of values for the variables exists, otherwise it is not. This study uses uniform random 3-SAT benchmark sets

(i.e. each clause in the data instances consists of exactly three variables) from the well-known SatLib library [3] [58] as well as Suite A and B from the Gotlieb Library [59]. The benchmark sets were also selected for comparison purposes with currently existing generation perturbative hyper-heuristics in the literature.

The aim is to find the solution in a minimum number of flips. A flip basically involves changing the value of the boolean variable, i.e. converting the 1 to 0 and vice versa. For these problems, there is usually a limit on the number of flips required to find a solution. A *success rate* (SR) representing the percentage of successful runs where a solution is found within the specified number of flips is commonly used as the *objective value*. As in the work conducted by Fukunaga [35], the learning (training) phase uses the 50 and 100 variable 3-SAT Benchmark instances while the testing phases uses the Gottlieb benchmark test suites A and B [59]. An overview of the benchmark instances is provided in Table 6.4 and Table 6.5.

Instances	Number of Variables	Number of Clauses	Number of Instances
uf50	50	218	1000
uf100	100	430	1000

Table 6.4. Details of the Uniform Random 3-SAT Benchmark sets used for training

Instances	Total Instances	Instances for each n	Size of problem n
Suite A	12	3	30, 40, 50, 100
Suite B	150	50	50, 75, 100

Table 6.5. Details of the Gotlieb Benchmark sets used for testing

6.5 INITIAL SOLUTIONS

The GE approach proposed in this study generates perturbative heuristics. These heuristics are typically applied to an existing initial solution in order to improve its quality. The initial solutions are generally constructed randomly or with the help of some constructive heuristics. Although this research does not focus on optimizing the construction of initial solutions, a brief discussion on how these solutions were constructed for the three problem domains is presented here nevertheless.

[3] The benchmarks are available from http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

6.5.1 Examination timetabling problem

The initial solutions for this domain were constructed using a combination of three constructive heuristics, namely largest degree, saturation degree and largest enrolment [45]. The three constructive heuristics were applied hierarchically as follows:

1. The saturation degree heuristic was applied first;
2. In case of ties in saturation degree the largest degree heuristic was applied next.
3. If there were any further ties with the largest degree heuristic, then the largest enrolment degree heuristic was applied.

This process was repeated a number of times until a feasible solution was found. Suffice it to say that the process of constructing the initial solution was time consuming as there was no guarantee that a feasible solution would be found at every attempt to construct one. In order to reduce the amount of time spent on constructing the feasible initial solutions for each problem instance every time the proposed GE algorithm was run, all the initial solutions were constructed only once i.e. the first feasible solution to be found for the problem instance was used as the initial solution in all the experiments involving that particular problem instance. In the study, a feasible solution was created for each instance.

6.5.2 Capacitated vehicle routing problem

The initial solutions for both the Golden and Christofides benchmark sets were constructed using the Clark Wright savings algorithm [60]. This algorithm has been widely used in the literature and it is adopted in this research as well.

6.5.3 Boolean satisfiability problem

The initial solutions for all the problem instances were randomly created, each variable in the boolean expression was assigned a random value.

6.6 HUMAN-DERIVED HEURISTICS

To evaluate the performance of the perturbative heuristics generated by the proposed approach, the generated heuristics will be compared to some commonly used human-derived perturbative heuristics in the literature. The human-derived perturbative heuristics that will be considered for each problem domain are described in the following sections.

6.6.1 Examination timetabling problem

For this problem domain, the human-derived perturbative heuristics that will be considered are presented in Table 6.6. These heuristics are widely used in many local search algorithms for solving examination timetabling problems [61].

Heuristic	Description
phe1	Moves a randomly selected exam to a new feasible timeslot
phe2	Swaps the timeslots of two randomly selected exams if feasible
phe3	Swaps the exams in two randomly selected timeslots
phe4	Randomly exchanges the timeslots of three randomly selected exams
phe5	Moves the exam causing the highest soft constraint violation to a new feasible timeslot
phe6	Moves two randomly selected exams to new feasible timeslots
phe7	Applies the Kempe chain operator to a randomly selected exam and timeslot
phe8	Moves a randomly selected exam to a new randomly selected room if feasible
phe9	Swaps the rooms of two randomly selected exams if feasible

Table 6.6. Human-derived perturbative heuristics for the examination timetabling problem

6.6.2 Capacitated vehicle routing problem

The human-derived perturbative heuristics that will be considered for this domain are presented in Table 6.7.

Heuristic	Description
phv1	Moves a randomly selected customer to a new feasible route
phv2	Swaps the routes of two randomly selected customers
phv3	Reverses a part of a tour between two selected customers on a randomly selected route
phv4	Randomly exchanges the routes of three randomly selected customers
phv5	Applies the 2-opt operator on a randomly selected route
phv6	Applies the 2-opt operator on all routes
phv7	Swaps the first portions of two randomly selected distinct routes
phv8	Swaps the adjacent customers of two customers selected from two distinct routes
phv9	Swaps the first and last portions of two randomly selected distinct routes
phv10	Move a randomly selected customer to another position in the same route

Table 6.7. Human-derived perturbative heuristics for the capacitated vehicle routing problem

6.6.3 Boolean satisfiability problem

The human-derived perturbative heuristics considered for this domain are presented in Table 6.8.

6.7 EXISTING GENERATION PERTURBATIVE HYPER-HEURISTICS

Apart from comparing the performance of the perturbative heuristics generated by the proposed GE approach with human-derived perturbative heuristics, the proposed approach will also be compared to other generation perturbative hyper-heuristics in the literature. The hyper-heuristics considered in this research are described in the following sections.

6.7.1 Examination timetabling problem

Research into the usage of generation perturbative hyper-heuristics to solve examination timetabling problems is in its inception [62]. There has not been much work in the area apart from the work conducted by Sabar et al. [40] where a grammatical evolution hyper-heuristic referred to as GE-HH is used to generate disposable local search operators from three heuristic components, namely acceptance criteria, neighbourhood structures and neighbourhood combinations. This research will therefore also

Heuristic	Description
GWSAT(p)	Flips a random variable with the highest net gain in a randomly selected unsatisfied clause with some probability p. The net gain of a variable is the number of clauses that remain unsatisfied in the boolean expression when the variable is flipped. If two or more variables have the same net gain, the ties are randomly broken.
Walksat(p)	Selects a random unsatisfied clause and randomly flips any variable with a net gain of 0 in the clause. Otherwise it uses some probability p to select a random variable to flip from the clause.
Novelty+(p,p_w)	Selects a random unsatisfied clause and with a probability p_w selects a random variable to flip. Otherwise, it flips the variable with the maximal net gain unless the variable has a minimal age. The age of the variable is the number of other variable flips that have occurred since the variable was last flipped. If the variable has minimal age in the unsatisfied clause, then the variable is selected for flipping with probability (1-p).

Table 6.8. Human-derived perturbative heuristics for the SAT problem

consider some selection perturbative hyper-heuristics that have produced good results for the ITC 2007 benchmark set in order to have better understanding of how well the proposed GE approach performs. The hyper-heuristics considered are described in Table 6.9.

Hyper-heuristic	Description
ETP-HH1	A generation perturbative hyper-heuristic proposed by Sabar et al. [40].
ETP-HH2	A selection perturbative hyper-heuristic proposed by Swan et al. [63].
ETP-HH3	A selection perturbative hyper-heuristic proposed by Burke et al. [64].
ETP-HH4	A selection perturbative hyper-heuristic proposed by Anwar et al. [65].

Table 6.9. Selected hyper-heuristics for the examination timetabling problem

6.7.2 Capacitated vehicle routing problem

The hyper-heuristics that will be considered for this domain are presented in Table 6.10.

Hyper-heuristic	Description
CVRP-HH1	A hyper-heuristic proposed by Sabar et al. [40].
CVRP-HH2	A hyper-heuristic proposed by Garrido et al. [66].
CVRP-HH3	A hyper-heuristic proposed by Meignan et al. [67].

Table 6.10. Selected hyper-heuristics for the vehicle routing problem

6.7.3 Boolean satisfiability problem

The generation perturbative hyper-heuristics that will be considered for this domain are presented in Table 6.11 below.

Hyper-heuristic	Description
SAT-HH1	A hyper-heuristic proposed by Bader and Poli [49].
SAT-HH2	A hyper-heuristic proposed by Fukunanga [35].

Table 6.11. Selected hyper-heuristics for the boolean satisfiability problem

6.8 BASELINE AND EXTENDED APPROACHES

Two GE approaches were developed and implemented during the course of this research. The first approach called the *Grammatical evolution baseline approach (GEBA)* was the initial approach proposed, based on the survey of existing literature, to demonstrate the main idea of generating new perturbative heuristics from heuristic components comprising basic operations and components of the solution. A basic grammar that used minimal domain knowledge was designed and applied to benchmark sets from three problem domains (see Section 6.4). After further research and in an attempt to improve the quality of results obtained by the GEBA, it was determined that including more domain information particularly from the solution space, generating other types of heuristics such as decision rules, was more likely to improve the efficiency and quality of the solutions obtained by the approach. As a result, the grammar for the GEBA was redesigned to include more domain specific information, new heuristic combination operators, new syntax of basic operations to cover a wider space of heuristics and decision rules. The new approach is called the *Grammatical evolution extended approach (GEEA)*. And the two approaches are discussed in more detail in Chapter 7 and Chapter 8.

6.9 PERFORMANCE MEASURES AND STATISTICAL TESTS

The viability of the proposed GE approach will be determined by considering three criteria, namely generality, consistency and efficiency.

6.9.1 Generality

This criterion will measure how well the proposed approach performs across the three problem domains and not only on the different instances of the one problem. This will entail comparing the quality of the results obtained by the perturbative heuristics evolved using the approach to those obtained by other techniques.

6.9.2 Consistency

This criterion will be considered during the training phase and will be used to measure the how the approach performs in terms of producing stable results when it is run several times for each problem instance. Since the approach is based on GE which is a stochastic search algorithm, it means that the solutions that will be obtained on each GE run will in most cases be different even if the initial solution is the same. In this study, the consistency of the approach will be measured in terms of average and standard deviation of the objective values (see Section 6.4) of the solutions obtained over 30 runs.

6.9.3 Efficiency

This criterion will be used to measure how close or far the results obtained by the proposed approach for each instance are to the initial results. Efficiency is measured in terms of percentage deviation (see Equation (6.1)) of the best solution found by the approach from the initial solution for the particular problem instance.

$$\Delta(\%) = \frac{best_{GE} - best^*}{best^*}\% \tag{6.1}$$

Where $best_{GE}$ is the best solution obtained by the proposed GE approach and best* is the initial solution for the problem instance.

Please note that the performance of the approach is measured by the quality of the solution (specified by an objective value) obtained by applying the heuristics generated by the approach to the given problem instance. Furthermore, in order to compare the performance of two approaches over a given set of instances, and draw confident conclusions, statistical tests will be conducted to judge which approach outperforms the other. Non parametric tests will be used instead of parametric ones due to the fact that parametric tests are usually based on strong assumptions that may not hold for the results obtained by heuristic algorithms [68]. Following Hutter et al. [69], the two-sided Wilcoxon signed-rank test will be employed to detect the potential differences between two approaches while the Friedman test with pairwise finner post hoc analysis will be employed to compare three or more approaches. In the tests, the null hypothesis states that the approaches in comparison have similar performance, and the 95% confidence level (i.e., p-values below 0.05 are statistically significant) will be considered unless otherwise stated.

6.10 TECHNICAL SPECIFICATIONS

The GE approaches were implemented in the Java programming language (version 1.8) using the GE package available in the ECJ toolkit [70]. The intellij IDEA 2018.2 IDE was used on a computer with an intel core i7 CPU, 2.3 GHz with 8gb RAM and running windows 10 64bit. The simulations were performed on the CHPC (Centre for High Performance Computing) multicore cluster. R was used to perform statistical tests on the results obtained.

6.11 SUMMARY

This chapter presented an overview of the methodology to be used in developing an algorithm for the automatic generation of perturbative heuristics to solve combinatorial optimization problems. The chapter also included an analysis of the related works and provided justification for this timely research. In addition, the benchmark sets, methods for constructing the initial solutions, the human-derived perturbative heuristics as well as the existing hyper-heuristics considered, the performance measures and the technical specifications of both the software and hardware used in the design, testing and evaluation of the approach were presented.

The next chapter presents the baseline GE appraoch proposed for generating perturbative heuristics.

CHAPTER 7 Grammatical Evolution Baseline Approach

7.1 INTRODUCTION

In this chapter, the grammatical evolution baseline approach is discussed. The baseline approach was the initial approach developed in the research to test the feasibility of the proposed methodology. Section 7.2 presents an overview of the GE approach including a discussion on the BNF grammar developed and the heuristic components. This is followed by a step by step discussion on all the aspects of the approach such as methods used in the generation of the initial population, genotype-phenotype mapping, fitness evaluation, selection method, genetic operators, population replacement strategy and the control parameters in section 7.3. The chapter summary follows in section 7.4.

7.2 OVERVIEW OF APPROACH

The main aim of this study is to design an approach for automating the generation of good quality perturbative heuristics to solve combinatorial optimization problems. A review of existing techniques in the literature shows that GP or GE are mostly employed to generate new heuristics from either existing heuristics or existing primitive components of the heuristics. This study adopts the second approach where new perturbative heuristics are generated from heuristic components rather than existing perturbative heuristics. GE is also selected over GP to do so. Specifically, the proposed approach takes as input several basic heuristic components and uses GE to automatically find the most appropriate combinations of these basic components to generate new heuristics. This approach not only leads to the generation of different types of heuristics but also to the discovery of new heuristics without the need for human experts.

The first step taken when solving a problem using GE is to design a suitable grammar. This step is very important because the grammar defines the search space for the solution to the problem. The next section discusses the basic grammar proposed for this approach in more detail.

7.3 BASIC BNF GRAMMAR

A good grammar is generally problem domain dependent but the goal of this study is to design a grammar that can be used across multiple problem domains. To do so, heuristic components are used as the basic elements of the grammar. The heuristic components were identified from a survey of existing literature and further motivated by the need to design a simple but flexible grammar that could be easily applied to different problem domains. The next section provides a detailed discussion on the identified heuristic components.

7.3.1 Heuristic Components

The heuristic components that were identified as appropriate and used as the basic elements of the BNF grammar are: *solution components*, *basic operations*, *combination operators* and *move acceptance criteria*.

7.3.1.1 Solution components

The solution components refer to the *parts* or *entities* making up the solution to the problem one is trying to solve. In the proposed approach, these are the only heuristic components that are problem dependent. For example, in examination timetabling problem the solution is a *timetable*. The timetable is made up of entities such as the examination, period and room. In the capacitated vehicle routing problem, solution components are the route and customer. The motivation for using solution components in the proposed approach is twofold: solution components are easier to identify and to make the approach less dependent on how the problem is represented (i.e. whether it is represented as a graph, permutation or set etc.). The ultimate goal here is to design the grammar in such a way that it can be easily applied to any problem domain by making only minor changes to the components of the solution. The solution components identified for the three problem domains investigated in this study are presented in Table 7.1.

Problem domain	Solution components
Examination timetabling problem (ETP)	Exam, Period and Room.
Capacitated vehicle routing problem (CVRP)	Route and Customer
Boolean satisfiability problem (SAT)	Clause and variable

Table 7.1. Identified Solution components for the three problem domains

The next section discusses the basic operations used in the proposed approach. To facilitate a general solution, the basic operations are problem domain independent.

7.3.1.2 Basic operations

The basic operations represent some of the commonly used operations to modify an existing solution in the literature. These are usually very simple operations such as moving an element from one position to another or swapping one element with another. Depending on the nature of the problem one is trying to solve, some of the operations may be specific to the particular problem. However, in an attempt to design a cross domain grammar, only the *move*, *swap*, *add (assign)*, *delete (unassign)* and *shuffle* operations are used in this study. These operations were identified as the most appropriate basic operations for generating heuristics across different problem domains. This set of basic operations can of course be extended to include more operations if necessary. Please also note the flip operation commonly used in the SAT problem can be implemented by the swap operator. A discussion on how these operations are used in this approach is presented in Section 7.3.2.

7.3.1.3 Combination operators

Combination operators are used to combine two or more generated heuristics into a single structure in order to combine the strengths of different heuristics. The literature shows that a combination of different heuristics can be very efficient in solving combinatorial optimization problems [71]. For example, Lu et al.[71] assessed the performance of combination operators (referred to as neighbourhood operators in their work) in solving university course timetabling problems within local search algorithms such as iterated local search, tabu search and steepest decent algorithm. The main aim of their work was to answer why some combination operators produce better results than others as well as to find out

the characteristics that constituted a good combination operator. Their conclusion was that combination operators can significantly improve the performance of a local search algorithm. The other studies that have looked at the benefits of combination operators include the work conducted by Johnson [72], Gaspero et al. [73] and Goeffon et al. [74]. For this basic approach, the heuristics are applied sequentially i.e. the heuristics are applied one after the other until the sequence ends.

7.3.1.4 Move acceptance criteria

This study generates perturbative heuristics. These heuristics are also called move operators since they have the same effect as the operators used in a local search. Peturbative heuristics are applied to an initial solution and a move acceptance criterion is used to decide whether the solution obtained after making a particular move (or perturbation) should be accepted or rejected. The decision is often made on the basis of whether the move led to an improvement in the objective value of the previous solution or not. A move is generally accepted if it leads to an improvement. In some cases, a non-improving move may also be accepted for the sole purpose of diversifying the search process to other regions of the search space. This is particularly important as it may prevent the search process from being stuck in a 'valley' or an area of local optima. A variety of move acceptance criteria have been proposed in the literature such as the *improving or equal only*, *all moves*, *simulated annealing*, *exponential monte carlo*, *record-to-record travel*, *great deluge*, *naive acceptance* and *adaptive acceptance* [40]. For this basic approach, only *improving or equal only* and *all moves* were used.

The four heuristic components discussed above formed the basic elements of the BNF grammar proposed in this study. The next step after determining the grammar elements involves the specification of the starting symbol (S), non-terminals (N), terminals (T) and the production rules (P) that will represent the heuristic components [40]. The next section presents the basic BNF grammar and also provides a description of the grammar elements.

7.3.2 Grammar specification

As discussed in Chapter 4 (Section 4.2.1), the BNF grammar is a four tuple consisting of the start symbol (S), terminals (T), non-terminals (N) and the production rules (P). For the proposed BNF grammar, a description of these grammar components is presented in Table 7.2.

Name	Symbol	Description
start symbol	<start>	The start symbol
Non-terminals	<heuristic>	Perturbative heuristic
	<action>	The move operator
	<swap>	The swap operation
	<move>	The move operation
	<shuffle>	The shuffle operation
	<add>	The add/reassign operation
	<delete>	The delete/unassign operation
	<comp>	Solution components
	<n>	integer constants
Terminals	comp1	first solution component
	comp2	second solution component
	.	.
	compn	n solution component

Table 7.2. Specification of BNF Grammar elements

The production rules specifying how the heuristic components are combined into perturbative heuristics are shown in Figure 7.1.

```
     <start> ::= <heuristic>
 <heuristic> ::= <action> <heuristic>
    <action> ::= <swap>|<shuffle>| <move>|
                 <add> | <delete>| <action>
      <swap> ::= <n> <comp> <comp>
   <shuffle> ::= <n> <comp>
      <move> ::= <n> <comp> <comp>
       <add> ::= <n> <comp>
    <delete> ::= <n> <comp>
      <comp> ::= comp1|....|compn
         <n> ::= 1|2|3|4|5|6|7|8|9|10
```

Figure 7.1. BNF grammar production rules

7.3.3 Perturbative heuristic

The grammar in Figure 7.1 above defines a perturbative heuristic in terms of two or more basic operations applied to components of the solution. It is a simple but flexible grammar that can easily be applied to any problem domain by simply specifying the solution components (represented by <comp>) for the particular domain. From the grammar, a perturbative heuristic can be composed of one or more of the following actions:

- **Swap**: Swaps *n* components specified in the *first* <*comp*> in the *second* <*comp*>. The *first* <*comp*> in the production rule must be contained in the *second* < *comp* >. For example, in the examination timetabling problem n examinations can be swapped in a particular period or room. The components are randomly selected.
- **Shuffle**: Shuffles *n* components in the solution, <*comp* > specifies the type of the component, e.g. a route. The components are randomly selected and shuffled one position to the right. For example, moving the position of n cities in a tour for the travelling salesman problem.
- **Move**: Moves *n* components specified in the *first* <*comp*> to a new position in the *second* <*comp*>. The *first* <*comp*> in the production rule must be contained in the *second* <*comp*>. The *first* <*comp*> also specifies the type of the component to move, e.g. an examination. Both the components and the new positions are randomly selected. For example moving *n* examinations to new periods in the examination timetabling problem.
- **Add**: Adds *n* components that were removed from the solution by a preceding delete action. The components are randomly selected from the deleted components and positions to place the components in the solution are also randomly selected.
- **Delete**: Deletes *n* randomly selected components from the solution. The deleted components are added to the list of components to be reallocated using the add action.

The range of *n* is 1 to 10. This range of values for *n* was determined based on the review of existing literature on the examination timetabling and vehicle routing problems. In most of the studies reviewed, it was discovered that the largest number of components (i.e. rooms, period and examinations for the examination timetabling problem or customers and routes for the vehicle routing problem) to be swapped or moved was three (3) [40]. It was therefore decided that the range be increased to ten(10) in an attempt to investigate whether better perturbative heuristics could be generated. An example of

a perturbative heuristic (in form of a parse tree) generated for solving the examination timetabling problem using the mapping process described in Section 4.2.3 and the solution components defined in Section 7.3.1.1 is illustrated in Figure 7.2

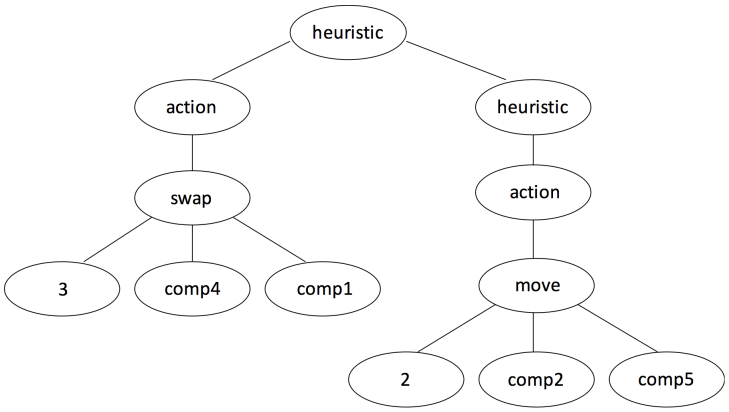

Figure 7.2. Example of a generated perturbative heuristic

The perturbative heuristic in Figure 7.2 is composed of two basic operations, swap and move, which are applied sequentially to the initial solution. The swap operation will swap three occurrences of a component of type comp4 in a component of type comp1 in a solution. This is followed by the move operation which moves two components of type comp2 to a new position in a component of type comp5 in the solution.

Having presented the BNF grammar that will be used to convert the genotypes (binary strings) to their corresponding phenotypes (perturbative heuristics), the next section discusses the GEBA. The search engine and mapper are implemented as in the original algorithm [25] and as discussed in Section 4.2.2 and Section 4.2.3 respectively.

7.4 GEBA

The proposed GE approach follows the basic strategy discussed in Chapter 4, and is represented in Figure 7.3 and Algorithm 7 below.

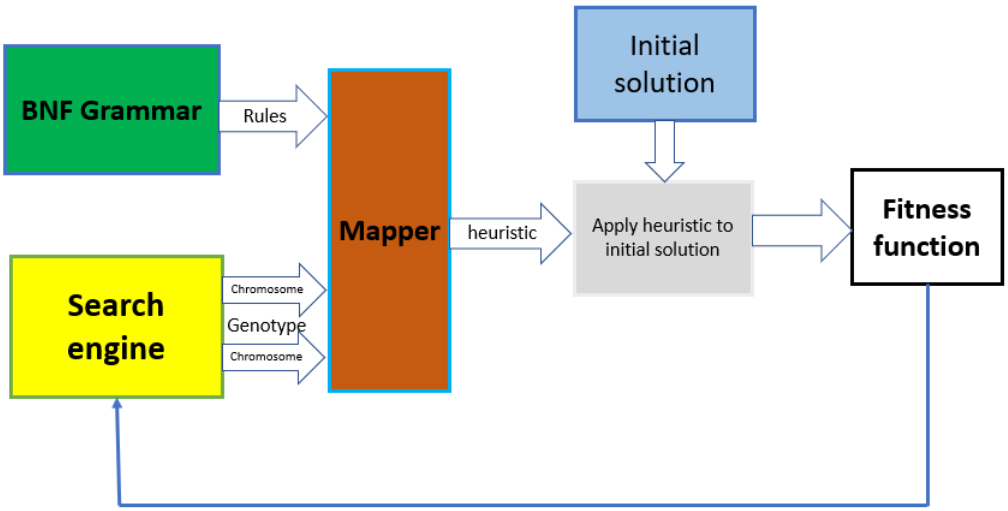

Figure 7.3. The Proposed GEBA

The approach starts with the design of the BNF grammar (see Section 7.3). It then initializes the parameters for the genetic algorithm used in the search engine and thereafter randomly generates an initial population of variable length binary chromosomes (genotypes). Using the RVD (see Section 4.3.1) initialization method, each chromosome in the population is then mapped to its corresponding program by the mapper. The quality of each program is determined by applying it to an initial solution constructed for the problem. The GEBA subsequently executes for a predefined number of generations with new offspring generated by applying selection, mutation and crossover operators at every generation. The generational population replacement strategy is used to replace the old population with a new one consisting of new offspring.

Algorithm 7 Pseudocode for the proposed GEBA

1: Generate an initial population of variable length binary strings
2: Map all the binary strings to their corresponding parse trees representing the perturbative heuristic (phenotype) using a BNF grammar
3: Apply each perturbative heuristic in the population to a provided initial solution to determine its fitness
4: **while** termination criterion not met **do**
5: Select the fitter perturbative heuristics for regeneration using the tournament selection method
6: Create new perturbative heuristics for the next generation by applying genetic operators to the earlier selected heuristics
7: Apply each new perturbative heuristic to a provided initial solution to determine its fitness
8: Replace all the perturbative heuristics in the old population with the new perturbative heuristics
9: **end while**
10: **return** perturbative heuristic with the best fitness as the best solution

The other aspects of the proposed GE approach are summarised in the following sections.

7.4.1 Initial population generation

The initial population of variable length binary strings (genotypes) is randomly generated. In order to make sure that all randomly generated genotypes map to their corresponding phenotypes and that there are no duplicates in the initial population, the RVD (see Section 4.3.1) initialization method is used.

7.4.2 Genotype-phenotype mapping

All the randomly generated binary strings are first converted to their denary values. The denary values are then mapped to their respective phenotypes (perturbative heuristics in this case), represented as parse trees, using the production rules of the grammar defined in Section 7.3. The mapping process is discussed in Section 4.2.3.

7.4.3 Fitness evaluation

Each individual in the population represents a perturbative heuristic. The fitness of the individual is evaluated by applying it to an initial solution. The objective value of the new solution obtained is used as the fitness for the individual. Initial solutions are typically constructed randomly or by using a constructive heuristic (see Section 6.5).

7.4.4 Selection method

The tournament selection method is used to select parents for regeneration. It is selected over the fitness proportionate method because it is simpler to implement, has a better run time, does not require any fitness scaling and it has a low susceptibility to takeover by fitter individuals.

7.4.5 Genetic operators

Three genetic operators, namely crossover, reproduction and mutation are used to generate new offspring. In particular, single point crossover and bit mutation are used and applied as discussed in Section 4.3.5. Following the proposal by Poli et al. [29], in the GEBA, 90% of individuals undergo crossover with the remaining 10% undergoing either reproduction or mutation. This is done on every generation.

7.4.6 Population Replacement

The generational population replacement strategy is used. This strategy basically replaces the old population with a new one generated after applying crossover and mutation operators.

7.4.7 GE Parameters

The values for the control parameters used to run the proposed GE approach were empirically determined by performing trial runs and carefully adapting the default parameters provided in the ECJ toolkit. The adapted values are listed in Table 7.3.

Control parameter	Value
population size	1024
Initial length of chromosome	60
Maximum number of wraps	5
Crossover rate	0.9
Mutation rate	0.01
Reproduction rate	0.09
Tournament size	7
Number of Generations	50

Table 7.3. Control parameter values

7.4.8 Termination criteria

The GE algorithm runs from one generation to the next and terminates after the specified number of generations (i.e. 50 generations in this case) is reached. The individual (perturbative heuristic) with the best fitness after the 50 generations is returned as the best solution.

7.5 SUMMARY

This chapter discussed the GEBA that was initially proposed to generate perturbative heuristics to solve combinatorial optimization problems. The approach took as input four heuristic components, namely *solution components*, *basic operations*, *combination operators* and *move acceptance criteria*, and automatically combined them into a perturbative heuristic. The grammar that was used to specify how the heuristic components were combined was presented. A discussion on the GEBA was also presented.

The next chapter discusses the GEEA which is an extension of the GEBA. The GEEA can be considered as an improvement over GEBA.

CHAPTER 8 Grammatical Evolution Extended Approach

8.1 INTRODUCTION

This chapter presents the grammatical evolution extended approach (GEEA) which extends the grammar used in the GEBA and is therefore considered as an improved approach. Section 8.2 presents a discussion on the main differences between the two approaches. The extended grammar is discussed in more detail in section 8.3. This is followed by a detailed discussion on the GEEA in section 8.4. The chapter summary is presented in section 8.5.

8.2 DIFFERENCES BETWEEN GEEA AND GEBA

Although the GEEA and GEBA (discussed in Chapter 7) share a similar overall strategy in that both take as input four heuristic components, namely *solution components*, *basic operations*, *combination operators* and *move acceptance criteria*, and use GE to automatically combine them into perturbative heuristics, the two approaches have some significant differences with respect to the actual heuristic components used in their grammars. And since the grammar has a significant impact on the type and quality of solutions obtained by the GE algorithm, it is therefore imperative that a good grammar is designed. For this reason, the BNF grammar in the GEEA was redesigned with some challenges to the structure of the heuristic components. The main differences between the heuristic components used in the grammars designed for the two approaches (i.e. the GEEA and GEBA) are:

1. *Solution components selection methods* : In the GEBA, solution components were selected at random. The selection methods have been extended in the GEEA to include five methods,

namely *lowest cost*, *highest cost*, *smallest size*, *largest size* and *random* selection methods.

2. *Conditional Constructs* : The grammar in the GEBA was used to evolve only functions. In the GEEA, the grammar has been extended to include if_then_else conditional operators in order to evolve decision rules in addition to the functions.
3. *Information from solution space*: The grammar in the GEEA has also been extended so that more information from the solution space can be used. This information includes the number of times the evolved heuristic has been applied, the fitness of the previous solution, the fitness of the current solution after applying the heuristic and the changes in the fitness of the solution each time the heuristic is applied. The GEBA only considered the fitness of the current solution after applying the evolved heuristic.
4. *Extended Syntax of Basic operations*: The syntax for the basic operations has been made more flexible so that a wider space of heuristics can be covered (see Section 8.3.3).

8.3 BNF GRAMMAR

As in the GEBA discussed in Chapter 7, the goal here is to design a grammar that can be used to generate perturbative heuristics for any problem domain. As a result, the GEEA also uses the same four heuristic components used in the GEBA. However, the basic grammar in the GEBA has been substantially extended and redesigned. The next section describes the extended grammar.

8.3.1 Extended Grammar

The start symbol (S), terminals (T), non-terminals (N) and the production rules (P) for the new grammar are specified in Table 8.1.

Name	Symbol	Description
start symbol	<start>	The start symbol
Non-terminals	<accept>	Move acceptance criteria
	<heuristic>	Perturbative heuristic
	<cop>	Combination operators
	<cond>	Conditional operators
	<rop>	Relational operators
	<h_value>	Heuristic value (stores information returned from the solution space)
	<comp>	Solution component e.g. exam
	<prob>	Solution component probability of selection
	<compSel>	Solution component selection method e.g. route with lowest cost
	<n>	Integer constants
Terminals	IO	Improving or equal only move acceptance criterion
	AM	All moves acceptance criterion
	c1	First solution component
	c2	Second solution component
	.	.
	cn	nth solution component
	$\cup$	Union combination operator
	$\rightarrow$	Random gradient combination operator
	$\leq$	Less or equal to
	$<$	Less than
	$>$	Greater than
	$\geq$	Greater or equal to

Table 8.1. Specification of BNF Grammar elements

The production rules specifying how the above heuristic components are combined into perturbative heuristics are shown in Figure 8.1.

```
   <start>   ::= <accept><heuristic>
   <accept>  ::= (IO)|(AM)
<heuristic>  ::= (swap <n><compSel>)|(move <n><compSel>)|(add <n><compSel>)|
                 (delete <n><compSel>)|(shuffle <n><compSel>)|
                 <heuristic><cop><heuristic>|
                 <heuristic>|
                 if(<cond>, <heuristic>,<heuristic>)
     <cond>  ::= (<rop><h_value><h_value>)|
                 If(<cond>, <cond>,<cond>)
  <h_value>  ::= prevFitness|curFitness|diffFitness|
                 curIteration|totalIterations
<compSel>    ::= lowestCost(<comp>) |
                 highestCost(<comp>)|
                 smallestSize(<comp>)|
                 largestSize(<comp>)|
                 random(<comp>)|
                 If(<prob>, <compSel>,<compSel>)
     <comp>  ::= c1|...|cn
      <cop>  ::= ∪|→
      <rop>  ::= <=|<|>|>=
     <prob>  ::= 25|50|75
       <n>   ::= 1|2|3|4|5|6|7|8|9|10|<n>|<n><n>
```

Figure 8.1. New grammar production rules

As in the initial grammar presented in Figure 7.1, perturbative heuristic are defined in terms of one or more basic operations applied to components of the solution. These basic operations and solution components are combined in the same manner as described in Section 7.3.2. In addition, perturbative heuristics are also evolved in form of decision rules which can be quite powerful in generating very good heuristics. The grammar elements are discussed in more detail in the following sections.

8.3.2 Acceptance criteria

The acceptance criteria ($< accept >$) decides whether to accept or reject a solution obtained after applying the generated perturbative heuristic. In the grammar, only two acceptance criteria i.e. Improving or equal only (IO) and All Moves (AM) are used. A variety of move acceptance criteria exist (see Section 7.3.1.4) but only the two are selected here as these are the most commonly used in the literature and have been shown to produce good results. The IO criterion basically accepts the obtained solution if it is superior to the previous one. A solution is considered superior in the examination timetabling and vehicle routing problem domains if it has a lower objective value. In

the boolean satisfiability problem domain, solutions with higher objective values are superior. In a situation where two solutions have the same objective value, then the solution that was discovered in fewer iterations is considered superior. The AM criterion on the other hand accepts all obtained solutions with no regard to their quality. It is included here so that the approach can be able to explore other areas of the search space and thereby avoid being trapped in a local optima.

8.3.3 Basic operations

The same five basic operations used in the GEBA are used in the GEEA. These are the *swap*, *move*, *shuffle*, *add* and *delete* operators. The syntax of the swap and move operators has been made more flexible in order to cover a wider range of heuristics. For example, the requirement that one component must be contained in another in order to perform a move or swap operation has been removed. Using the new syntax, a swap operation can be applied to two similar components. For example, a room can be swapped with another room. This will of course be equivalent to swapping the exams in the two rooms.

8.3.4 Solution Components

As described earlier, solution components($< comp >$) refer to *the parts making up a solution to the problem being addressed*. These components are problem domain specific. In the new grammar, the solution component to use when generating a heuristic can be selected using five methods i.e. the *lowestCost*, *highestCost*, *smallestSize*, *largestSize* and *random* selection. The *lowestCost* method selects the solution component with the lowest cost while the *highestCost* method selects the solution component with the highest cost. For the SAT problem, *cost* refers to the variable gain score while *size* refers to the age of the variable. A cost value is considered low or high if it is less than or greater than the median cost value of the particular solution component. If more than one component has the same cost value, then the component is selected randomly from those with the same cost value. The *smallestSize* and *largestSize* methods select solution components with the smallest and largest size respectively. A component is considered small if its size is smaller than the median size for that solution component. Similarly, a large component is one with a size that is bigger than the median size for the solution component. If more than one component has the same size, then the component is selected randomly from those with the same size. For example, the size of the solution

component *exam* in the examination timetabling problem refers to the number of students taking the exam. The median size of the *exam solution component* will be the median value of the sorted list of exams. The *random* method selects solution components randomly. Probabilistic branching has also been included in the grammar to deal with cases where a decision may need to be made when choosing a component between two different selection methods(e.g. *highestCost* and *smallestSize*). The parameter <prob> represents the probability of selecting the first argument. For example, the rule *if (25, lowestCost(exam), smallestSize(exam)* implies that the probability of choosing the lowestCost(exam) over the smallestsize(exam) is 25%. Notice that each selection method has a argument represented by <comp>. <comp> specifies the actual type of the solution component to select (e.g. whether the solution component is an exam, room or period in the case of the examination timetabling problem). The type of solution component is randomly selected. As an example, *lowestCost(exam)* specifies that the exam with the lowest cost should be selected.

8.3.5 Information from Solution Space

When a heuristic is applied to a solution, some information from the solution search space is recorded. In the grammar, this information consists of numerical values for the fitness of the previous solution, fitness of the current solution, current iteration, total iterations so far and the differences between the fitness of the solutions between the iterations. This information forms part of the conditions for evolving heuristics and it also assists in breaking ties between two or more evolved heuristics that have the same fitness. In the GEEA, ties between the evolved heuristics are first broken using their fitness values. If the heuristics have the same fitness value, then the number of iterations it took to evolve the heuristics is considered. If the tie is still not broken, then the best heuristic is selected randomly amongst them.

8.3.6 Relational and Conditional Operators

A number of operators are used. These include relational operators ($< rop >$) such as the less than, greater than, less or equal to, greater or equal to. These binary operators are only used in the condition of the if-then-else. The if-then-else conditional operator is useful for producing heuristics in the form of decision rules. The operator has three arguments, the first argument is always a boolean, while the second and third arguments share the same object type (e.g. a heuristic, solution component or integer

value). The boolean argument determines the output of this operator. If the value is true, then output is the second parameter, otherwise the function returns the third parameter.

8.3.7 Combination operators

These are represented by $(< cop >)$ in the grammar and are used to combine two or more generated heuristics into a single structure. The main reason for doing so is to combine the strengths of different heuristics. In the grammar, the union $(\cup)$, Random Gradient $(\rightarrow)$ and Token-Ring Search(no symbol) operators are used. Whenever two heuristics are combined with no symbol (e.g. Figure 8.2), it means that the token ring operator is used and the heuristics are applied sequentially. A discussion on these operators can be found in Table 8.2.

Combination operator	Description
Union ($(\cup)$)	This operator combines two or more different heuristics. For example, given two perturbative heuristics P1 and P2, the union operator consecutively applies P1 followed by P2 and then calls the acceptance criterion to accept or reject the solution. This operator therefore combines the strengths of different heuristics which may be very useful in solving problems [40], [71].
Random Gradient $(\rightarrow)$	This operator repeatedly applies one heuristic until there is no further improvement in the objective value of the solution. This is then followed by the application of the other heuristics in a similar manner (i.e. until there is no improvement in the objective value of the solution). For example, given two perturbative heuristics P1 and P2, the random gradient operator continuously applies P1 until there is no improvement. It then applies P2 starting from last obtained optimum solution by P1 [46], [42].
Token-Ring Search(no symbol)	This operator consecutively applies the heuristics one after the other until the sequence ends. For example, given two perturbative heuristics P1 and P2, the token-Ring consecutively applies P1 and P2 until the sequence ends. When the search is restarted, it starts from the first heuristic in the sequence using the optimum obtained by the last heuristic in the previous sequence [71],[73].

Table 8.2. Description of combination operators

8.3.8 Perturbative heuristics

The grammar in Figure 8.1 can be used to generate different types of perturbative heuristics. In addition, two or more heuristics can be combined to form a composite heuristic. Some examples of perturbative heuristics in form of parse trees that can be generated for solving the examination timetabling, capacitated vehicle routing and the boolean satisfiability problems using the mapping process described in Section 4.2.3 and the solution components defined in Section 7.3.1.1 are shown in Figure 8.2, Figure 8.3 and Figure 8.4 respectively.

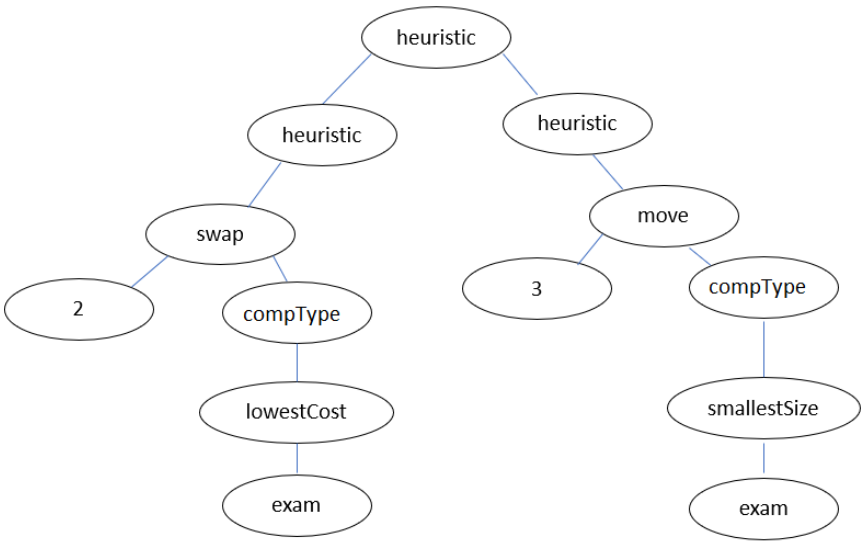

Figure 8.2. An example of a composite heuristic combining swap and move operators

The composite perturbative heuristic in Figure 8.2 combines two heuristics into one. The first heuristic swaps two lowest cost exams while the second heuristic moves three smallest sized exams to other locations in the solution. The two heuristics are applied consecutively.

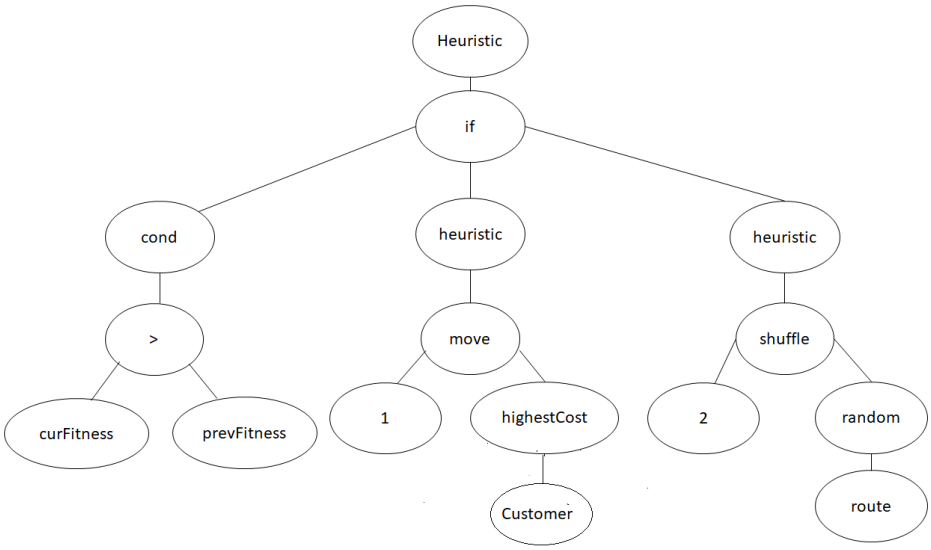

Figure 8.3. An example of a generated decision rule

The decision rule shown in Figure 8.3 first checks if the current fitness of the solution is greater than the previous fitness. If it is true, then the heuristic in the second branch is selected otherwise the heuristic in the third branch is selected.

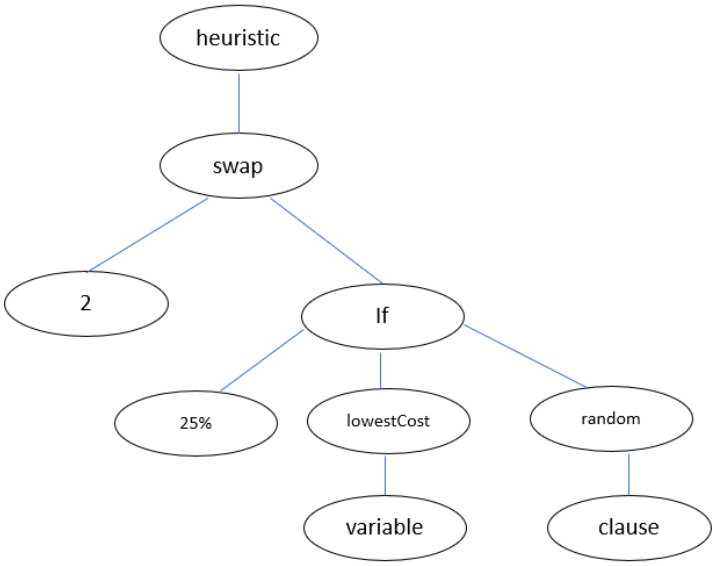

Figure 8.4. An example of a generated SAT heuristic

The heuristic in Figure 8.4 swaps (flips) the lowest cost variable in the clause if the probability of choosing the lowest cost variable is 25% otherwise it swaps (flips) a random variable in a random clause. Note that lowest cost variable in this case refers to the variable with the minimum gain score.

8.4 GEEA

The GEEA follows a similar strategy to the one described in Section 7.4. The algorithm steps are summarised in the following sections.

8.4.1 Initial population generation

Similar to the initial approach, the initial population of variable length binary strings (genotypes) is randomly generated. The RVD (see Section 4.3.1) initialization method is also used to ensure the genotypes map to their corresponding phenotypes and that there are no duplicates in the initial population.

8.4.2 Genotype-phenotype mapping

The binary strings are similarly first converted to their denary values and thereafter mapped to their respective computer programs, represented in the form of parse trees, using the production rules of the grammar defined in Figure 8.1. The mapping process discussed in Section 4.2.3 is also used here.

8.4.3 Fitness evaluation

One of the goals of the new approach is to generate reusable heuristics. In order to do this, all the generated heuristics are first trained on a training set and the best performing heuristic is then selected to be tested on a testing set. The process of determining the fitness of a generated heuristic is however the same in that the heuristic is applied to an initial solution provided for each problem instance in the training and testing sets. To make it simpler to understand and capture the fitness of the generated heuristic on both the training and testing sets, the solution objective value has been reformulated to a number between 0 and 1 such that the best possible fitness value of the heuristic is equal to the number of instances the heuristic is applied to. For example, if the heuristic is applied to 4 problem instances, then the fitness value of the heuristic is the sum of the four outcomes from each application on the problem instances. Therefore, the best possible fitness value in such a case is 4 while 0 is the worst.

8.4.4 Selection method

The tournament selection method is also used in this approach (see Section 7.4.4).

8.4.5 Genetic operators

The genetic operators discussed in Section 7.4.5 are applied in this approach as well.

8.4.6 Population Replacement

The generational population replacement strategy is used to replace individuals in the population.

8.4.7 GE Parameters

The parameter values which produced good results for the GEBA were also adopted in this approach. The values are listed in Table 7.3.

8.4.8 Termination criteria

The termination criteria was set to the maximum number of generations i.e. 50 generations in this case.

8.5 SUMMARY

This chapter discussed the GEEA for generating perturbative heuristics to solve combinatorial optimization problems. The new approach took as input four heuristic components, namely *solution components*, *basic operations*, *combination operators* and *move acceptance criteria*, and automatically combined them into a perturbative heuristic. However, the grammar was redesigned and includes new methods for selecting solution components, conditional constructs, utilizes some information from the solution space and extends the syntax of the basic actions in order to cover a wider range of heuristics.

The next chapter presents and discusses the results obtained after applying the two approaches to the problem instances from the three problems domains discussed in Section 6.4 .

CHAPTER 9 Results and Discussion

9.1 INTRODUCTION

This chapter presents the results obtained after applying the GEBA and GEEA to the benchmark sets from the three problem domains described in Section 6.4. The results presented here include also those obtained by the human-derived perturbative heuristics (discussed in Section 6.6) as well as the perturbative heuristics generated by existing generation perturbative hyper-heuristics (described in Section 6.7). Section 9.2 presents and discusses the results obtained by the GEBA while section 9.3 presents and discusses the results obtained by the GEEA. The chapter summary is presented in section 9.4.

9.2 RESULTS OF THE GEBA

This section presents the results obtained by applying the GEBA to benchmark sets from the three problem domains discussed in Section 6.4. In particular, it reports on the objective values of the solutions obtained after iteratively applying the best perturbative heuristic evolved by the GEBA to the initial solutions (see Section 6.5) constructed for the benchmark sets. In order to assess the performance of the approach (using the performance measures described in Section 6.9), the solutions obtained by the evolved perturbative heuristic are further compared with:

1. The solutions obtained by the human-derived perturbative heuristics (discussed in Section 6.6);
2. The solutions obtained by the perturbative heuristics generated by other generation perturbative hyper-heuristics in the literature (described in Section 6.7);

The statistical significance of the results is evaluated using the non parametric Friedman test with a post hoc pairwise Finner test with the GEBA used as the control method. The level of significance α of 0.05 is used for the test.

Please note that both the training and test results are presented in one table to make it easier to compare the results over all the benchmark instances with other approaches. As mentioned earlier, 50 % of the instances were used for training and the remaining 50% used for testing. The instances that were used for training are highlighted in light gray.

9.2.1 Generated perturbative heuristic

This section presents the results obtained by the best perturbative heuristics evolved by the GEBA. The results presented in Table 9.1, Table 9.2, Table 9.3 and Table 9.4 show the objective values (discussed in Section 6.4) of the initial solution, solution obtained by the evolved heuristic and percentage of improvement ($\Delta(\%)$) of the results. The percentage of improvement ($\Delta(\%)$) is calculated using Equation (9.1).

$$\Delta(\%) = \frac{Initial - best_{GE}}{Initial}\% \tag{9.1}$$

Where $best_{GE}$ is the objective value of the solution obtained by the evolved heuristic and *Initial* is the objective value of the initial solution.

9.2.1.1 Examination timetabling problem

The ITC 2007 benchmark instances were used for this domain. As the benchmark set contains only 12 instances, some of the instances were used for training and others were used for testing. The training and testing results obtained by the best perturbative heuristic (shown in prefix notation Figure 9.1) evolved by the GEBA are presented in Table 9.1.

```
(shuffle (move (move (move 2 exam room)
    (shuffle (move 5 exam period) (swap 2 exam room))
    5) (swap (move (move 5 exam room) (move 3 exam
    period) (move (swap 3 exam room) (swap
    period (move 2 period room))(move 1 exam room))) (move (shuffle 1 room) (swap
    5 exam room) (swap 5 period room))) (move 5 exam period))
    (shuffle 2 period))
```

Figure 9.1. Best perturbative heuristic evolved by the GEBA for the ETP

The generated pertubative heuristic shown in Figure 9.1 has combined the *swap*, *move* and *shuffle* operations to create a composite perturbative heuristic. In the figure, the operation *(move 2 exam room)* moves 2 randomly selected exams to new rooms which can accommodate them. The operation *(shuffle 2 period)* shuffles 2 randomly selected periods which according to the approach used in this research, whereby a timetable is considered as a table composed of periods as rows and rooms as columns, is equivalent to randomly moving exams in the two periods to any rooms that can accommodate them during that period. The operation *(swap 2 exam room)* randomly swaps the rooms of two exams. The other operations can be interpreted in a similar manner.

Instance	Initial	GEBA	$\Delta(\%)$
1	12980	9120	30
2	1420	1008	29
3	14400	9920	31
4	24042	17840	26
5	5830	4250	27
6	36574	28630	22
7	10630	6810	36
8	15395	10230	34
9	1920	1502	22
10	31080	29001	7
11	46824	44320	5
12	8065	7410	8

The table shows the objective values of the initial solution (Initial) and the solution obtained by the perturbative heuristic evolved by the GEBA. The highlighted rows show the results for the instances that were used for training.

Table 9.1. Performance of the best perturbative heuristic evolved by the GEBA on the ITC 2007 benchmark instances

9.2.1.2 Capacitated vehicle routing problem

The Christofides and Golden benchmark sets were used for this domain. As these benchmark sets contain 14 and 20 instances respectively, some of the instances were used for training and others for testing. The training and testing results obtained by the best perturbative heuristic (shown in prefix notation Figure 9.2) evolved by the GEBA are presented in Table 9.2 and Table 9.3.

```
(shuffle (move (delete 2 customer)(add 2 customer)
(swap 5.0 (shuffle 3 route) (move 4.0 customer route))
(shuffle 2.0 route)) (swap 3.0 customer route))
```

Figure 9.2. Best perturbative heuristic evolved by the GEBA for the CVRP

The generated pertubative heuristic shown in Figure 9.2 has combined the *swap*, *delete*, *add*, *move* and *shuffle* operations to create a composite perturbative heuristic. In the figure, the operation *(delete 2 customer)* randomly deletes 2 customers from a randomly selected route. The operation *(add 2 customer)* reassigns the previously deleted customers to a new randomly selected route where the customer can be serviced. The operation *(shuffle 3 route)* randomly selects 3 routes and randomly shuffles the order in which customers are serviced on the routes. The operation *(swap 3.0 customer route)* randomly selects 3 customers on a randomly select route and randomly swaps the order in which they are serviced on the route. The other operations can also be interpreted in a similar manner.

Instance	Initial	GEBA	$\Delta(\%)$
1	885.30	575.20	35
2	1277.20	880.10	31
3	1290.35	870.65	33
4	1680.20	1220	27
5	1770.04	1502.10	15
6	780.20	585.30	25
7	1250.30	1060.4	13
8	1265.02	1100.2	13
9	1500.30	1260.30	16
10	1810.02	1580.6	13
11	1370.50	1098.2	20
12	1100.90	877.01	20
13	1830.40	1600.10	13
14	1100.60	877.03	20

The table shows the objective values of the initial solution (Initial) and the solution obtained by the perturbative heuristic evolved by the GEBA. The highlighted rows show the results for the instances that were used for training..

Table 9.2. Performance of the best perturbative heuristic evolved by the GEBA on the Christofides instances

Instance	Initial	GEBA	Δ(%)
1	7980.20	6200.80	22
2	11920.42	9110.40	24
3	14090.50	12010.3	15
4	18042.70	13540.20	25
5	9830.95	7700.10	22
6	10574.34	9080.65	14
7	12730.01	10980.6	14
8	15995	13740.1	14
9	860.25	685.30	20
10	920.64	810.40	12
11	1280.12	970.10	24
12	1520.95	1220.3	20
13	1100.62	877.20	20
14	1520.04	1160.10	24
15	1900.85	1490.7	22
16	1950.62	1724.2	12
17	950.36	760.8	20
18	1210.10	1120.60	7
19	1890.42	1550.63	18
20	2100.30	1980.4	6

The table shows the objective values of the initial solution (Initial) and the solution obtained by the perturbative heuristic evolved by the GEBA. The highlighted rows show the results for the instances that were used for training.

Table 9.3. Performance of the best perturbative heuristic evolved by the GEBA on on the Golden instances

9.2.1.3 Boolean satisfiability problem

For this problem domain, the 50 and 100 variable uniform random 3-SAT instances were used for training while the Gottlieb testing suites, namely Suite A and Suite B were used for testing. The training and testing results obtained by the best perturbative heuristic evolved by the GEBA (shown in Figure 9.3) are presented in Table 9.4 and Table 9.5 respectively.

```
(swap 1.0 variable clause)
```

Figure 9.3. Best perturbative heuristic evolved by the GEBA for the SAT

The generated pertubative heuristic shown in Figure 9.3 simply swaps(flips) a random variable in a randomly selected clause.

Benchmark set	Initial	GEBA	$\Delta(\%)$	# variable flips
uf50	98.20	100	2	599
uf100	97.50	99.20	2	3092

The values for the Initial and GEBA represent the success rate (%).

Table 9.4. Performance of the best perturbative heuristic evolved by the GEBA on on 3-SAT training instances

Benchmark set		Initial	GEBA	$\Delta(\%)$	# variable flips
Suite A	40	98.60	100	1.40	1240
	50	89.50	93.60	4.10	6230
	100	69.30	75.00	5.70	38120
Suite B	50	90.20	94.50	4.30	12630
	75	72.60	81.30	8.70	29120
	100	54.30	60.70	6.40	30440

The values for the Initial and GEBA represent the success rate (%).

Table 9.5. Performance of the best perturbative heuristic evolved by the GEBA on the Gottlieb testing instances

9.2.1.4 Discussion of results

From the results presented in Table 9.1, Table 9.2, Table 9.3, Table 9.4 and Table 9.5, it can be seen that the heuristics generated by the GEBA were able to improve the objective values of the initial solutions for all the benchmark sets. This demonstrates that the approach is capable of generating heuristics that improve the quality of the initially obtained solutions which is what perturbative heuristics do. The next section compares the results presented here and obtained by the best perturbative heuristic evolved by the GEBA with those obtained by the commonly used human-derived perturbative heuristics.

9.2.2 GEBA vs Human-derived heuristics

In these experiments, the solutions obtained by the best perturbative heuristic evolved by the GEBA were compared with the solutions obtained by the human-derived perturbative heuristics (described in Section 6.6). All the human-derived heuristics were specifically implemented, tested and applied to the same initial solutions that the perturbative heuristic evolved by the GEBA was applied to (see Section 9.2.1). The performance of each heuristic was determined by repeatedly applying the heuristic to the initial solution until the objective value of the solution could not be improved any further. The termination criterion for each run was empirically set to 10 consecutive iterations of non-improvement after 30 non-improving heuristic application steps [75].

The results obtained by each heuristic after the termination of the run were captured and are presented in Section 9.2.2.1, Section 9.2.2.2 and Section 9.2.2.3. The Friedman test with the level of significance α of 0.05 was used to evaluate the statistical significance of the results obtained by each approach. Whenever the results were found to be significantly different, a posthoc pairwise test using the Finner method with the GEBA as the control method was carried out to determine which of the approaches performed differently. To confirm the observed results, a contrast estimation test reflecting the value of the differences in the objective values of solutions obtained by the different approaches was also performed.

The next section presents and compares the results obtained by the perturbative heuristic evolved by the GEBA with those obtained by the human-derived perturbative heuristics on the examination timetabling problem.

9.2.2.1 Examination timetabling problem

In this section, the results obtained by the perturbative heuristic evolved by the GEBA are compared with those obtained by the human-derived perturbative heuristics. The ITC 2007 benchmark set was used for this domain and the human-derived perturbative heuristics considered are represented in Table 9.6. A comparison of the results obtained by each heuristic is given in Table 9.7 with the statistical test results presented in Table 9.8, Table 9.9 and Table 9.10.

Heuristic	Description
phe1	Moves a randomly selected exam to a new feasible timeslot
phe2	Swaps the timeslots of two randomly selected exams if feasible
phe3	Swaps the exams in two randomly selected timeslots
phe4	Randomly exchanges the timeslots of three randomly selected exams
phe5	Moves the exam causing the highest soft constraint violation to a new feasible timeslot
phe6	Moves two randomly selected exams to new feasible timeslots
phe7	Applies the Kempe chain operator to a randomly selected exam and timeslot
phe8	Moves a randomly selected exam to a new randomly selected room if feasible
phe9	Swaps the rooms of two randomly selected exams if feasible

Table 9.6. Human-derived perturbative heuristics for the examination timetabling problem

Instance	GEBA	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
1	**9120**	10930	11325	12224	11100	11195	12785	11998	12633	12530
2	**1008**	1195	1130	1210	1140	1205	1310	1210	1132	1250
3	**9920**	13009	12920	13040	12752	12138	13396	12670	12834	12397
4	**17840**	22740	21368	22875	21653	21521	21430	20174	21874	20631
5	**4250**	4874	4710	4900	4784	4590	4889	5074	5034	4995
6	**28630**	31652	30456	31478	32014	30745	31654	31096	33264	32541
7	**6810**	10345	10032	10423	11004	10325	10457	10659	10475	10087
8	**10230**	13036	12524	12952	12420	12352	13470	12785	12987	12210
9	**1502**	1668	1600	1710	1640	1603	1620	1630	1640	1608
10	**29001**	29036	29100	29120	29090	29050	29110	29050	29130	29180
11	**44320**	46095	46100	46020	46090	46070	46060	46025	46032	46001
12	**7410**	7750	7630	7960	7720	7730	7850	7640	7700	7830

The results show the objective values of the solutions obtained by each heuristic. The best values are shown in bold.

Table 9.7. Comparison of the performance of the best perturbative heuristic generated by the GEBA and the human-derived heuristics on ITC 2007 instances

To evaluate the statistical significance of the results presented in Table 9.7, the non-parametric Friedman test with the post hoc Finner test was carried out. Table 9.8 shows the results of the Friedman's rank sum test performed in order to test the first (*null*) hypothesis which states that all the heuristics perform equally. The level of significance α of 0.05 was used for the test.

Friedman's rank sum test	
Friedman's chi-squared	47.554
df	9
p-value	3.094e-07

Table 9.8. Friedman test results for the GEBA and human-derived perturbative heuristics on ITC 2007 instances

From the results obtained in Table 9.8, it can be observed that the p-value (3.094e-07) is less than the level of significance α of 0.05. This means that there is at least one perturbative heuristic that

performs differently from the rest. As a result, the *null* hypothesis was rejected and a posthoc pairwise comparison test using the Finner method with the perturbative heuristic evolved by the GEBA as the control method was performed. The results for the posthoc test showing the differences in performance between the approaches are shown in Table 9.9.

	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
GEBA	3.59e-5	1.17e-2	7.83e-7	1.44e-4	1.17e-2	6.22e-7	4.53e-4	1.53e-6	1.82e-4

Table 9.9. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

The results presented in Table 9.9 show that there is a significant difference in the performance of the perturbative heuristic evolved by the GEBA and all the human-derived perturbative heuristics since all the obtained p-values are less than the level of significance α of 0.05. To confirm that the GEBA evolved perturbative heuristic performs better than the human-derived heuristics as can be observed from Table 9.7, a contrast estimation test based on medians was performed. In this test the performance of the perturbative heuristics is reflected by the value of the differences in the objective values of their solutions. A negative value for the heuristic in a given row indicates that the heuristic performs better than the heuristic in a given column. The results for the contrast estimation test are presented in Table 9.10.

	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
GEBA	-2000.3	-1896.8	-2106.7	-1979.3	-1896.0	-2117.5	-1985.2	-2091.3	-1987.6

Table 9.10. Results for the contrast estimation test on ITC 2007 instances

From the results in Table 9.10, it can be concluded that the perturbative heuristic evolved by the GEBA performs better than the human-derived heuristics on the ITC 2007 benchmark based on the negative values obtained with respect to the differences in the objective values of the solutions.

The next section presents the results for the capacitated vehicle routing problem.

9.2.2.2 Capacitated vehicle routing problem

This section presents the results obtained by applying the perturbative heuristic evolved by the GEBA and the human-derived perturbative heuristics to the Christofides and Golden instances. The human-derived perturbative heuristics considered for this problem domain are described in Table 9.11. A comparison of the results obtained by each heuristic is presented in Table 9.12 and Table 9.16. The statistical test results are given in Table 9.13, Table 9.14 and Table 9.15.

Heuristic	Description
phv1	Moves a randomly selected customer to a new feasible route
phv2	Swaps the routes of two randomly selected customers
phv3	Reverses a part of a tour between two selected customers on a randomly selected route
phv4	Randomly exchanges the routes of three randomly selected customers
phv5	Applies the 2-opt operator on a randomly selected route
phv6	Applies the 2-opt operator on all routes
phv7	Swaps the first portions of two randomly selected distinct routes
phv8	Swaps the adjacent customers of two customers selected from two distinct routes
phv9	Swaps the first and last portions of two randomly selected distinct routes
phv10	Move a randomly selected customer to another position in the same route

Table 9.11. Human-derived perturbative heuristics for the capacitated vehicle routing problem

Instance	GEBA	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
1	**575**	796	745	790	691	601	599	676	642	651	746
2	**880**	1128	1096	1115	1057	921	896	987	968	972	1001
3	**871**	1180	1099	1184	1113	976	924	991	998	1005	1220
4	**1220**	1523	1345	1515	1426	1301	1296	1347	1387	1452	1326
5	**1502**	1773	1658	1694	1535	1575	1510	1500	1610	1631	1645
6	**585**	796	645	663	681	621	595	648	625	632	655
7	**1060**	1128	1111	1090	1100	1136	1090	1109	1116	1110	1125
8	**1100**	1180	1099	1125	1127	1191	1096	1105	1135	1121	1107
9	**1260**	1523	1389	1366	1396	1360	1300	1396	1375	1350	1395
10	**1581**	1773	1648	1694	1675	1624	1610	1645	1620	1647	1631
11	**1098**	1305	1210	1198	1146	1120	1100	1214	1186	1191	1263
12	**877**	950	910	911	900	900	901	936	915	926	925
13	**1600**	1626	1640	1681	1646	1630	1607	1646	1641	1634	1632
14	**877**	950	921	900	902	896	891	900	896	910	916

The results show the objective values of the solutions obtained by each heuristic. The best values are shown in bold.

Table 9.12. Comparison of the performance of the best perturbative heuristic generated by the GEBA and the human-derived heuristics on Christofides instances

As in Section 9.2.2.1, the non-parametric Friedman test with the post hoc Finner test was used to evaluate the statistical significance of the results presented in Table 9.12. The results for Friedman's rank sum test as performed in order to test the first (*null*) hypothesis are presented in Table 9.13.

Friedman's rank sum test	
Friedman's chi-squared	84.38
df	10
p-value	6.917e-14

Table 9.13. Friedman test results for the GEBA and human-derived perturbative heuristics on Christofides instances

From the results in Table 9.13, it can be seen that the p-value of 6.917e-14 is significantly less than the level of significance α of 0.05 adopted for the test which indicates that there is a significant difference in the performance of the heuristics. As explained earlier, the Friedman test does not show which heuristics perform differently and therefore a posthoc pairwise test using the Finner method with the heuristic evolved by the GEBA as the control was performed. The results of the posthoc test are shown in Table 9.14.

phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
5.40e-11	1.33e-6	1.38e-7	2.55e-6	1.46e-2	2.92e-1	1.12e-5	2.37e-4	2.42e-5	1.38e-7

Table 9.14. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

The results in Table 9.14 show that there is a significant difference in the performance of the perturbative heuristic evolved by the GEBA and most of the human-derived heuristics apart from phv6 (p-value = 0.29). In order to determine which of the two perturbative heuristics performs better than the other, a contrast estimation test based on medians was used. The results for the contrast estimation test are presented in Table 9.15.

	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
GEBA	-192.5	-96.2	-116.4	-92.8	-55.4	-36.7	-85.1	-72.6	-80.4	-101.8

Table 9.15. Results for the contrast estimation test on Christofides instances

From the results obtained in Table 9.15, it can be concluded that the perturbative heuristic evolved by the GEBA performs better than the human-derived phv6 heuristic since the value of the difference in the objective value of the solutions obtained by the GEBA generated heuristic and phv6 is -36.7 which is negative. In addition, the other results from Table 9.15 show negative values for the difference in the objective value of the solutions obtained by the heuristics. It can therefore be concluded that the perturbative heuristic evolved by the GEBA performs better than the human-derived heuristic on the Christofides instances.

Apart from the Christofides benchmark set, the performance of the perturbative heuristics was also evaluated on the Golden benchmark set. The objective values of the solutions obtained by each heuristic are presented in Table 9.16.

Instance	GEBA	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
1	**6841**	8745	7796	7106	7806	7101	7040	7450	7651	8146	8215
2	**11751**	14708	15696	12570	12956	12456	12226	12645	12965	13010	13000
3	**15220**	23351	22876	17683	18101	16988	16451	17980	18452	18895	18975
4	**18011**	34100	26954	24561	25301	23961	22076	23985	23874	25654	26786
5	**9651**	11900	12005	13451	15011	11520	10991	12321	12450	12624	12524
6	**11645**	15087	14635	15069	14529	13991	13766	14156	14231	15001	15075
7	**17566**	21443	21875	21687	19894	20795	19735	20946	20654	21110	21452
8	**15200**	23879	22915	23960	23357	22126	21985	23784	22965	23785	23651
9	**610**	989	915	912	874	840	830	921	981	996	962
10	**795**	1314	1246	1280	1251	1055	991	1053	1150	1257	1351
11	**951**	1596	1266	1206	1272	1161	1095	1165	1106	1245	1426
12	**1591**	2016	2050	2067	2001	2051	1950	2065	2058	2043	2055
13	**871**	1244	1190	1152	1166	1075	1063	1068	1206	1102	1101
14	**1400**	2904	2003	2626	2145	2695	2562	2201	2910	2001	2966
15	**1451**	8575	5987	5765	4966	4165	3125	5123	5280	5246	5678
16	**2195**	2411	2686	2600	2521	2341	2286	2415	2635	2765	2832
17	**750**	811	835	865	801	875	866	847	895	842	889
18	**1085**	1636	1256	1190	1206	1156	1126	1164	1125	1155	1197
19	**1501**	1765	1675	1624	1651	1682	1600	1665	1666	1642	1725
20	**1969**	2385	2563	2451	2163	2210	2169	2398	2195	2005	2397

phv1 - phv10 are the human derived heuristics (see Table 9.11). The best values are shown in bold.

Table 9.16. Comparison of the performance of the best perturbative heuristic generated by the GEBA and the human-derived heuristics on Golden instances

The non-parametric Friedman test with the post hoc Finner test was also used to evaluate the statistical significance of the results presented in Table 9.16. The results for Friedman's rank sum test performed

in order to test the first (*null*) hypothesis are presented in Table 9.17.

Friedman's rank sum test	
Friedman's chi-squared	110.62
df	10
p-value	2.2e-16

Table 9.17. Friedman test results for the GEBA and human-derived perturbative heuristics on Christofides instances

From the results in Table 9.17, it can be seen that the p-value of 2.2e-16 is significantly less than the level of significance α of 0.05 which indicates that there is a significant difference in the performance of the heuristics. A posthoc pairwise test using the Finner method with the GEBA evolved heuristic as the control method was therefore performed to show which heuristics perform differently. The results of the posthoc test are shown in Table 9.18.

phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
4.9e-13	4.1e-10	4.7e-9	5.4e-6	6.7e-4	1.2e-1	9.3e-6	2.6e-7	4.9e-8	4.9e-13

Table 9.18. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

From Table 9.18, it can be seen that there is a significant difference in the performance of the perturbative heuristic evolved by the GEBA and most of the human-derived heuristics apart from phv6 (p-value = 0.12). In order to determine which of the two perturbative heuristics performs better than the other, a contrast estimation test based on medians was performed. In this test, a negative value for the heuristic in a given row indicates that the heuristic performs better than the heuristic in a given column. The results for the contrast estimation test are presented in Table 9.19.

	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
GEBA	-873.0	-700.0	-639.9	-583.3	-420.9	-322.2	-530.6	-596.7	-655.5	-792.9

Table 9.19. Results for the contrast estimation test on Golden instances

From the results in Table 9.19, the value of the difference in the objective value of the solutions obtained by the GEBA generated heuristic and phv6 is -322.2. Since this value is negative, it can be concluded

that the perturbative heuristic evolved by the GEBA performs better than the human-derived phv6 heuristic. In addition, the other results from Table 9.19 also show negative values for the difference in the objective value of the solutions obtained by the heuristics. It can therefore be concluded that the perturbative heuristic evolved by the GEBA performs better than the human-derived heuristic on the Golden instances.

The next section presents the results obtained by the perturbative heuristic evolved by the GEBA and the human-derived perturbative heuristics on benchmark sets for the the boolean satisfiability problem.

9.2.2.3 Boolean satisfiability problem

This section presents the results obtained by applying the perturbative heuristic evolved by the GEBA and the human-derived perturbative heuristics to Gottlieb SAT benchmark sets for the boolean satisfiability problem. The human-derived perturbative heuristics considered for the Boolean satisfiability problem are described in Table 9.20. A comparison of the results obtained by the heuristics is presented in Table 9.21. The statistical test results are given in Table 9.22.

Heuristic	Description
GWSAT(p)	Flips a random variable with the highest net gain in a randomly selected unsatisfied clause with some probability p. The net gain of a variable is number of clauses that remain unsatisfied in the boolean expression when the variable is flipped. If two or more variables have the same net gain, the ties are randomly broken.
Walksat(p)	Selects a random unsatisfied clause and randomly flips any variable with a net gain of 0 in the clause. Otherwise it uses some probability p to select a random variable to flip from the clause.
Novelty+(p,p_w)	Selects a random unsatisfied clause and with a probability p_w selects a random variable to flip. Otherwise, it flips the variable with the maximal net gain unless the variable has a minimal age. The age of the variable is the number of other variable flips that have occurred since the variable was last flipped. If the variable has minimal age in the unsatisfied clause, then the variable is selected for flipping with probability (1-p).

Table 9.20. Human-derived perturbative heuristics for the SAT problem

Benchmark set		GEBA		GWSAT(0.5)		WalkSAT(0.5)		Novelty+(0.7, 0.01)	
		SR(%)	AF	SR(%)	AF	SR(%)	AF	SR(%)	AF
Suite A	40	**100**	1240	**100**	6062	**100**	2604	**100**	1767
	50	93.60	6230	90.33	31295	99.67	22219	**100**	3061
	100	75.00	38120	71.00	10104	81.00	24855	**90.0**	33148
Suite B	50	94.50	12630	89.56	20841	94.4	18609	**96.64**	13783
	75	81.30	29120	71.74	28584	82.48	32939	**89.3**	25833
	100	60.70	30440	56.6	17704	59.16	21824	**65.8**	24655

SR is the success rate and AF is the average number of flips to solution. The values for the success rate and number of flips as well as the parameter values for GWSAT, WalkSAT and Novelty+ were obtained from [35]. The best results are shown in bold.

Table 9.21. Comparison of the performance of the best perturbative heuristic generated by the GEBA and the human-derived heuristics on Gottlieb SAT instances

The non-parametric Friedman test with the post hoc Finner test was also used to evaluate the statistical significance of the results presented in Table 9.21. The results for Friedman's rank sum test performed in order to test the first (*null*) hypothesis which states that all the heuristics perform equally are presented in Table 9.22.

Friedman's rank sum test	
Friedman's chi-squared	13.56
df	3
p-value	0.00357

Table 9.22. Friedman test results for the GEBA and human-derived perturbative heuristics on Gottlieb SAT instances

From Table 9.22, it can be seen that the p-value of 0.00357 is less than the level of significance α of 0.05 adopted for the test. This indicates that there is at least one heuristic that performs differently. The *null* hypothesis was therefore rejected and the pairwise posthoc finner test with the GEBA as the control method was performed in order to show which heuristics perform differently. The results of the finner test are presented in Table 9.23.

	GWSAT(0.5)	WalkSAT(0.5)	Novelty+(0.7, 0.01)
GEBA	0.2050463	0.8230633	0.2050463

Table 9.23. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

The pairwise finner test results in Table 9.23 shows that there is no significant difference between the GEBA and the human-derived heuristics. It is therefore not possible to determine how the performance of the perturbative heuristic evolved by the GEBA compares to that of the human-derived perturbative heuristics using the Finner test. A contrast estimation test based on medians was therefore carried out. The results of the contrast estimation test are presented in Table 9.24.

	GWSAT(0.5)	WalkSAT(0.5)	Novelty+(0.7, 0.01)
GEBA	-4.57125	1.38250	5.47875

Table 9.24. Results for the contrast estimation test on Gottlieb SAT instances

From Table 9.24, it can be seen that the GEBA perform better than GWSAT but is outperformed by both WalkSAT and Novelty+. As mentioned earlier, this poor performance can be attributed to the fact the GEBA is a basic approach which incorporates only minimal domain knowledge when searching for perturbative heuristics. As a result, the perturbative heuristic evolved is not expected to compete with the best performing heuristics.

The next section compares the GEBA to other existing hyper-heuristics.

9.2.3 GEBA vs Other Hyper-heuristics

This section compares the results obtained by the GEBA to existing hyper-heuristics in the literature. Suffice to mention here that there have been very few works in the literature that have applied generation perturbative hyper-heuristics to the problem domains investigated in this research. As a result, the GEBA is also compared to some of the best performing selection perturbative hyper-heuristics. A description of the hyper-heuristics considered for each problem domain is presented in the respective sections. The next section presents the results obtained by the hyper-heuristics on the ITC 2007 benchmark set for the examination timetabling problem.

9.2.3.1 Examination timetabling problem

As mentioned in Section 6.7, most hyper-heuristics that have been applied to the examination timetabling problem have been constructive hyper-heuristics. The GE-HH by Sabar et al.[40] is the only generation hyper-heuristic that has been applied to the ITC2007 benchmark. For this reason, the GEBA is also compared to some of the best performing selection perturbative hyper-heuristics in the literature. The hyper-heuristics considered for the ITC2007 benchmark set are described in Table 9.25. The results obtained by the approaches are presented in Table 9.26 with the statistical test results given in Table 9.27, Table 9.28 and Table 9.29.

Hyper-heuristic	Description
ETP-HH1	A generation perturbative hyper-heuristic proposed by Sabar et al. [40].
ETP-HH2	A selection perturbative hyper-heuristic proposed by Swan et al. [63].
ETP-HH3	A selection perturbative hyper-heuristic proposed by Burke et al. [64].
ETP-HH4	A selection perturbative hyper-heuristic proposed by Anwar et al. [65].

Table 9.25. Selected hyper-heuristics for the examination timetabling problem

Instance	GEBA	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
1	9120	**4362**	5875	6235	11823
2	1008	**380**	893	2974	976
3	9920	**8991**	13352	15832	26770
4	17840	**15094**	24174	35106	—
5	4250	**2912**	4734	4873	6772
6	28630	**25735**	27510	31756	30980
7	6810	**4025**	5731	11562	11762
8	10230	**7452**	9507	20994	16286
9	1502	**1111**	—	—	—
10	29001	**14825**	—	—	—
11	44320	**28891**	—	—	—
12	7410	**6181**	—	—	—

The comparison is between the objective values for the best solutions obtained for each approach. The best values are shown in bold. "—" indicates that the values were not provided or a feasible solution was not obtained.

Table 9.26. Comparison of the performance of the GEBA and other HHs on ITC 2007 instances

The Friedman test was used to evaluate the statistical significance of the results presented in Table 9.26. The level of significance α of 0.05 was adopted for the test and the results of this test are presented in Table 9.27.

Friedman's rank sum test	
Friedman's chi-squared	22.97
df	4
p-value	0.0001283

Table 9.27. Friedman test results for the GEBA and other hyper-heuristics on ITC 2007 instances

From the results obtained in Table 9.27, the p-value (0.0001283) is less than the level of significance α of 0.05. This denotes that there is at least one approach that performs differently from the rest. As a result, the *null* hypothesis was rejected and a posthoc pairwise comparison test using the Finner method with the GEBA as the control method was performed. The results for the posthoc Finner test

showing the differences in performance between the GEBA and other hyper-heuristics are shown in Table 9.28.

	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
GEBA	0.04449193	0.2279483	0.897279	0.3804785

Table 9.28. P-values for the posthoc pairwise Finner test with the GEBA as the control method

The results presented in Table 9.28 show that there is a significant difference in the performance of the GEBA and ETP-HH1 (p-value = 0.04449193). There is however no significant difference in the performance of the GEBA and the remaining hyper-heuristics as can be seen from the p-values which are greater than the level of significance α of 0.05. Although the finner test shows which approaches perform differently, it is not possible to determine from the results whether the GEBA performs better than the other hyper-heuristics or not. To show the best approach among the five, the constrast estimation method based on medians was used. In this test the performance of the heuristics is reflected by the value of the differences in the objective values of their solutions and a negative value for the heuristic in a given row indicates that the heuristic performs better than the heuristic in a given column. The results for the contrast estimation test are presented in Table 9.29.

	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
GEBA	2558.375	306.375	-3622.750	-5120.320

Table 9.29. P-values for the posthoc pairwise Finner test with the GEBA as the control method

From Table 9.29, it can be concluded that the GEBA performs better than ETP-HH3 and ETP-HH4. The other hyper-heuristics, namely ETP-HH1 and ETP-HH2 perform better. This performance by the GEBA can be attributed to the fact the perturbative heuristic evolved by the GEBA is applied as it is with no optimization as is the case for ETP-HH1 and ETP-HH2. In addition, the GEBA is a baseline approach that uses minimum domain information to search for the best heuristics. As a result, it is not expected to find the best performing perturbative heuristic.

The next section presents the results for the capacitated vehicle routing problem.

9.2.3.2 Capacitated vehicle routing problem

This section compares the results obtained by the GEBA and other perturbative hyper-heuristics on the Christofides and Golden instances for the capacitated vehicle routing problem. The hyper- heuristics considered for the benchmark sets are described in Table 9.30. The results obtained by each approach are presented in Table 9.31 and Table 9.33. The Friedman test was used to evaluate the statistical significance of the results. The level of significance α of 0.05 was adopted for the test and the results of the statistical test are presented in Table 9.32 and Table 9.34.

Hyper-heuristic	Description
CVRP-HH1	A generation perturbative hyper-heuristic proposed by Sabar et al. [40].
CVRP-HH2	A selection perturbative hyper-heuristic proposed by Garrido et al. [66].
CVRP-HH3	A selection perturbative hyper-heuristic proposed by Meignan et al. [67].
CVRP-HH4	A selection perturbative hyper-heuristic proposed by Pisinger et al. [76].

Table 9.30. Selected hyper-heuristics for the vehicle routing problem

Instance	GEBA	CVRP-HH1	CVRP-HH2	CVRP-HH3
1	0.00	**0.00**	**0.00**	**0.00**
2	0.51	**0.00**	0.05	0.62
3	2.25	**0.00**	0.21	0.42
4	1.54	**0.11**	0.52	2.05
5	2.08	**1.33**	2.05	5.07
6	0	**0.00**	**0.00**	—
7	0.39	**0.00**	0.09	—
8	0.28	**0.00**	**0.00**	—
9	0.81	**0.20**	0.70	—
10	0.60	**0.53**	1.24	—
11	0.07	**0.00**	0.88	0.00
12	1.43	**0.00**	**0.00**	—
13	1.03	1.90	1.00	**0.00**
14	0.31	**0.00**	**0.00**	—

The values represent the percentage deviation from the best known results in the literature. The best values are shown in bold. "—" indicates that the values were not provided or a feasible solution was not obtained.

Table 9.31. Comparison of the performance of the GEBA and other HHs on Christofides instances

Friedman's rank sum test	
Friedman's chi-squared	5.44
df	3
p-value	0.1422

Table 9.32. Friedman test results for the GEBA and other hyper-heuristics on Christofides instances

From Table 9.32, it can be seen that the p-value of 0.1422 is greater than the level of significance α of 0.05 adopted for the test which indicates that there is not enough evidence to conclude that there is a significant difference in the performance of the heuristics. As a result, the *null* hypothesis which states that the perturbative heuristics perform equally cannot be rejected.

Instance	**GEBA**	CVRP-HH1	CVRP-HH4
1	7900.20	**5626.81**	5650,91
2	12110.40	**8446.19**	8469,32
3	16010.30	11081.60	**11047,01**
4	18540.20	13658.84	**13635,31**
5	9830.10	**6460.98**	6466,68
6	12080.65	8462.10	**8416,13**
7	17980.60	10202.24	**10181,75**
8	15740.10	**11690.82**	11713,62
9	685.30	**583.39**	585,14
10	910.40	**740.91**	748,89
11	970.10	**919.80**	922,7
12	1820.30	**1111.43**	1119,06
13	877.20	**857.19**	864,68
14	1460.10	**1083.59**	1095,4
15	1490.70	**1350.17**	1359,94
16	2604.20	**1631.91**	1639,11
17	760.80	**707.76**	708,9
18	1120.60	**1003.43**	1002,42
19	1550.63	**1368.12**	1374,24
20	2080.40	**1820.09**	1830,8

The comparison is between the best values obtained for each approach. The best values are shown in bold.

Table 9.33. Comparison of the performance of the GEBA and other HHs on Golden Instances

Friedman's rank sum test	
Friedman's chi-squared	32.5
df	2
p-value	8.764e-08

Table 9.34. Friedman test results for the GEBA and other hyper-heuristics on Golden instances

From Table 9.34, it can be seen that the p-value of 8.764e-08 is significantly less than the level of significance α of 0.05 adopted for the test which indicates that there is a significant difference in the performance of the approaches. A posthoc pairwise test using the Finner method with the GEBA as the control method was therefore performed to show which approaches perform differently. The results of the test are shown in Table 9.35.

	CVRP-HH1	CVRP-HH4
GEBA	3.13e-08	7.72e-05

Table 9.35. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

From Table 9.35, it can be seen that there is a significant difference, based on the p-values, in the performance of the GEBA and the other hyper-heuristics, namely CVRP-HH1 and CVRP-HH4. To determine the best approach among the three approaches, a contrast estimation method based on medians was also used. The results for the contrast estimation test are presented in Table 9.36.

	CVRP-HH1	CVRP-HH4
GEBA	541.67	533.99

Table 9.36. Results for the constrast estimation test on Golden instances

From Table 9.36, the value of the difference in the objective value of the solutions obtained by the GEBA generated heuristic and other hyper-heuristics are all positive. This indicates that the two hyper-heuristic approaches, namely CVRP-HH1 and CVRP-HH4 perform better than the GEBA. The poor performance by the GEBA can be attributed to the fact that the GEBA is a baseline approach which uses minimum domain information when searching for perturbative heuristics. For this reason, the approach is not expected to find heuristics capable of outperforming state of the art hyper-heuristic approaches. In addition, the heuristics evolved by the GEBA are applied as they are with no optimization and this has an effect on the quality of results obtained by the heuristic.

The next section presents and compares the results obtained by the GEBA and other hyper-heuristic approaches on the boolean satisfiability problem.

9.2.3.3 Boolean satisfiability problem

This section compares the results obtained by the GEBA and other generation perturbative hyper-heuristics on the uniform random 3-SAT instances for the boolean satisfiability problem. The generation perturbative hyper-heuristics considered for this domain are presented in Table 9.37. The results obtained by each approach are presented in Table 9.38 with the statistical test results given in Table 9.39.

Hyper-heuristic	Description
SAT-HH1	A generation perturbative hyper-heuristic proposed by Bader and Poli [49].
SAT-HH2	A generation perturbative hyper-heuristic proposed by Fukunanga [35].

Table 9.37. Selected generation perturbative hyper-heuristics for the boolean satisfiability problem

Benchmark set		GEBA		SAT-HH1		SAT-HH2	
		SR(%)	AF	SR(%)	AF	SR(%)	AF
Suite A	40	**100**	1240	—	—	**100**	538
	50	93.60	6230	**100**	12872	**100**	3416
	100	75.00	38120	92.00	54257	**100**	20754
Suite B	50	94.50	12630	**100.00**	18936	99.38	6590
	75	81.30	29120	95.00	26571	**98.48**	19791
	100	60.70	30440	74.00	41284	**78.8**	39909

The comparison is between the best values for the success rate for each approach. The best values are shown in bold. "—" indicates that the values were not provided.

Table 9.38. Comparison of the performance of the GEBA and other generation perturbative hyper-heuristics on Gottlieb SAT instances

Friedman's rank sum test	
Friedman's chi-squared	8.3158
df	2
p-value	0.01564

Table 9.39. Friedman test results for the GEBA and human-derived perturbative heuristics on Gottlieb SAT instances

From Table 9.39, the p-value of 0.01564 which is less than the level of significance α of 0.05 indicates that there is at least one approach that performs differently. The *null* hypothesis was therefore rejected and the pairwise posthoc finner test with the GEBA as the control method was performed in order to show which heuristics perform differently. The results of the finner test are presented in Table 9.40.

	SAT-HH1	SAT-HH2
GEBA	0.03983262	0.007189517

Table 9.40. P-values for the posthoc pairwise comparison Finner test with the GEBA as the control method

The pairwise finner test results in Table 9.40 shows that there is a significant difference in performance between the GEBA and the two other hyper-heuristics, namely SAT-HH1 and SAT-HH2. It is however not possible to determine which approach is the best among the three using this test. For this reason, a contrast estimation test based on medians was carried out. The results of the contrast estimation test are presented in Table 9.41.

	SAT-HH1	SAT-HH2
GEBA	13.433333	17.046667

Table 9.41. Results for the constrast estimation test on Gottlieb instances

From Table 9.41, it can be seen that the GEBA is outperformmed by both the SAT-HH1 and SAT-HH2 hyper-heuristics. This poor performance can also be attributed to the fact the GEBA is a basic approach which incorporates only minimal domain knowledge when searching for perturbative heuristics. As a result, the perturbative heuristic evolved is not expected to compete with the best performing hyper-heuristics.

The next section presents the results for the grammatical evolution extended approach (GEEA) which is considered as an improvement to the GEBA.

9.3 RESULTS OF THE GEEA

This section presents the results of applying the GEEA to benchmark sets from the three problem domains discussed in Section 6.4. GEEA is an improvement on GEBA.

9.3.1 Generated perturbative heuristics

The results presented in Table 9.42, Table 9.43, Table 9.44 and Table 9.45 show the objective values (discussed in Section 6.4) of the initial solution, solution obtained by the best GEEA generated heuristic and percentage of improvement ($\Delta(\%)$) of the results (calculated using Equation (9.2)).

$$\Delta(\%) = \frac{Initial - best_{GE}}{Initial}\% \tag{9.2}$$

Where $best_{GE}$ is the objective value of the solution obtained by the best heuristic generated by the GEEA and *Initial* is the objective value of the initial solution.

9.3.1.1 Examination timetabling problem

The best perturbative heuristic evolved by the GEEA for the examination timetabling problem is shown in Figure 9.4.

```
(shuffle 5.0 (swap 3.0 (move 2.0 (move 5.0 highestCost(exam))
    (swap 2.0 (move 2.0 highestCost(exam)) (shuffle
     2.0 largestSize(period))) (swap 2.0 (move 2.0 lowestCost(exam)) (move
     1.0 highestCost(exam)))) (swap 2.0 (move 1.0 (shuffle 2.0 (swap 3.0
     (move 5.0 highestCost(exam)) (swap 3.0 (shuffle 1.0 (swap 2.0
     (shuffle 1.0 random(period)) (swap 3.0 highestCost(room)) (move 2.0
     (swap 2.0 largestSize(exam))))) (move 5.0 (swap 5.0 largestSize(exam))
     (swap 2.0 (shuffle 2.0 highestCost(room))) (swap 3.0 smallestSize(period)))
     (move 5.0 highestCost(exam)))) (swap 5.0 random(exam)))
     (shuffle 5.0 random(period)) (shuffle 2.0 random(room)))
     (shuffle 1.0 (shuffle 2.0 highestCost(period))))))
```

Figure 9.4. Best perturbative heuristic evolved by the GEEA for the ETP

The generated pertubative heuristic shown in Figure 9.4 has combined the *swap*, *move* and *shuffle* operations to create a composite perturbative heuristic. The interpretation of the operations is similar

to the one discussed for the GEBA in Section 9.2.1. For example, in the figure, the operation *(move 5.0 highestCost(exam))* moves 5 exams contributing the highest cost to the objective value of the solution to new periods or rooms which can accommodate them. The operation *(shuffle 1.0 random(period))* shuffles 1 randomly selected period which according to the approach used in this research, whereby a timetable is considered as a table composed of periods as rows and rooms as columns, is equivalent to randomly moving the exams in the period to any room that can accommodate them during that period. The other operations can be interpreted in a similar manner.

Instance	Initial	GEEA	$\Delta(\%)$
1	7870	4901	37.73
2	620	476	23.22
3	12780	9064	29.08
4	20042	16432	18.01
5	5400	3432	36.44
6	30620	26577	13.20
7	8800	5434	38.25
8	10530	7820	25.74
9	1780	1140	35.95
10	24350	20958	13.93
11	42130	39420	6.43
12	7840	6354	18.95

Table 9.42. Performance of the best GEEA generated heuristic on ITC 2007 instances. The highlighted rows show the results for the instances that were used for training.

9.3.1.2 Capacitated vehicle routing problem

The best perturbative heuristic evolved by the GEEA for the capacitated vehicle routing problem is shown in Figure 9.5.

```
(swap 2.0 (move 3.0 highestCost(customer)) (swap 2.0 random(customer))
(shuffle 1.0 (swap 2.0 (shuffle 1.0 largestSize(route))
(move 3.0 (swap 3.0 (delete 2.0 highestCost(customer))))
(move 2.0 (add 2.0 highestCost(customer))))
(swap 2.0 (swap 3.0 largestSize(customer)))
(move 2.0 (move 3.0 smallestSize(customer))
(swap 2.0 largestSize(route))
(shuffle 2.0 (shuffle 1.0 random(route))
(swap 3.0 highestCost(customer))))))
```

Figure 9.5. Best perturbative heuristic evolved by the GEEA for the CVRP

In a similar manner as the composite heuristic discussed in Figure 9.4, the generated pertubative heuristic shown in Figure 9.5 has combined the *swap*, *move*, *delete*, *add* and *shuffle* operations to create yet another composite perturbative heuristic. The interpretation of the operations is also similar. For example, in the figure, the operation *move 3.0 highestCost(customer)* moves 3 customers contributing the highest cost to objective value of the solution to new routes or new locations within the same route where they are currently assigned. The operation *(swap 2.0 random(customer))* selects 2 random customers and swaps their locations. The operation *(delete 2.0 highestCost(customer))* deletes 2 customers causing the highest increase in the overall cost of the solution. The other operations can be interpreted in a similar manner.

Instance	Initial	GEEA	Δ(%)
1	785.30	524.61	33.20
2	1277.20	839.52	34.26
3	1290.35	844.72	34.53
4	1680.20	1044.28	37.84
5	1770.04	1318.25	25.52
6	780.20	555.43	28.80
7	1250.30	913.22	26.96
8	1265.02	868.35	31.35
9	1500.30	1172.74	21.83
10	1810.02	1404.30	22.42
11	1370.50	1042.87	23.91
12	1100.90	831.29	24.49
13	1830.40	1556.94	14.94
14	1100.60	869.04	21.03

Table 9.43. Performance of the best GEEA generated heuristic on Christofides instances. The highlighted rows show the results for the instances that were used for training.

Instance	Initial	GEEA	Δ(%)
1	7980.20	5761.06	27.81
2	11920.42	8601.05	27.84
3	14090.50	11307.28	19.75
4	18042.70	14007.31	22.37
5	9830.95	6707.84	31.77
6	10574.34	8700.49	17.72
7	12730.01	10253.17	19.46
8	15995	11754.68	26.51
9	860.25	584.30	32.08
10	920.64	740.34	19.58
11	1280.12	921.22	28.04
12	1520.95	1114.13	26.75
13	1100.62	858.50	22.00
14	1520.04	1000.19	34.20
15	1900.85	1352.17	28.87
16	1950.62	1641.18	15.83
17	950.36	708.80	25.60
18	1210.10	1001.66	17.23
19	1890.42	1369.19	27.89
20	2100.30	1822.65	13.21

Table 9.44. Performance of the best GEEA generated heuristic on Golden instances. The highlighted rows show the results for the instances that were used for training.

9.3.1.3 Boolean satisfiability problem

The best perturbative heuristic evolved by the GEEA for the examination timetabling problem is shown in Figure 9.6.

```
if((> prevFitness curFitness),
  swap 1.0 highestCost(variable),
  swap 1.0 lowestCost(variable))
```

Figure 9.6. Best perturbative heuristic evolved by the GEEA for the SAT

The generated pertubative heuristic shown in Figure 9.6 generates a decision rule. The interpretation of the rule is that whenever the objective value of the previous solution is greater than the objective value of the current solution, swap (flip) the variable with the highest cost (i.e. net gain) otherwise swap(flip) the variable with the lowest cost.

Benchmark set	Initial	GEEA	$\Delta(\%)$	# variable flips
uf50	97.40	100	2.60	460
uf100	97.10	100	2.90	2700

The values for the Initial and GEEA represent the success rate (%).

Table 9.45. Performance of the best GEEA generated heuristic on 3-SAT training instances

Benchmark set		Initial	GEEA	$\Delta(\%)$	# variable flips
Suite A	40	98.20	100	1.80	1080
	50	88.60	98.20	9.60	4320
	100	79.40	89.00	9.50	32040
Suite B	50	92.20	97.30	5.10	10520
	75	75.80	89.40	13.60	26530
	100	57.50	73.50	16.00	28102

The values for the Initial and GEEA represent the success rate (%).

Table 9.46. Performance of the best GEEA generated heuristic on Gottlieb testing instances

9.3.1.4 Discussion of results

From the results presented in Table 9.42, Table 9.43, Table 9.44, Table 9.45 and Table 9.46, it can be seen that the heuristics generated by the GEEA, just like those generated by GEBA, were able to improve the objective values of the initial solutions for all the benchmark sets. This demonstrates that

the GEEA is equally capable of generating heuristics that improve the quality of the initially obtained solutions. And this is what perturbative heuristics do. The next section compares the results presented here with those obtained by the GEBA.

9.3.2 GEEA vs GEBA

This section presents a comparison of the results obtained by the best perturbative heuristics generated by the GEEA and GEBA. For comparison purposes, the two approaches were applied to the same initial solutions. The results obtained by each approach including the statistical test results are presented in Section 9.3.2.1, Section 9.3.2.2 and Section 9.3.2.3.

9.3.2.1 Examination timetabling problem

In this section, the results obtained by GEEA are compared with those obtained by GEBA on the examination timetabling problem. As in the first approach, the ITC 2007 benchmark set is used. Table 9.47 shows a comparison of the results.

Instance	Initial	GEBA	GEEA	$\Delta(\%)$
1	7870	6250	**4901**	21.58
2	620	574	**476**	17.07
3	12780	10650	**9064**	14.89
4	20042	18020	**16432**	8.81
5	5400	4800	**3432**	28.50
6	30620	28440	**26577**	6.55
7	8800	6910	**5434**	21.36
8	10530	8200	**7820**	4.63
9	1780	1420	**1140**	19.72
10	24350	21990	**20958**	4.69
11	42130	41770	**39420**	5.63
12	7840	7100	**6354**	10.51

$\Delta(\%)$ is the percentage of improvement in the objective values of the solutions obtained by GEEA over those obtained by GEBA. The best objective values are shown in bold.

Table 9.47. Comparison of the performance of the best GEBA and GEEA generated perturbative heuristic on ITC 2007 instances

To evaluate the statistical significance of the results obtained in Table 9.47, the Wilcoxon signed rank test was carried out. The level of significance α of 0.05 was used for the test. Table 9.48 shows the results of the statistical test.

Wilcoxon signed rank test	
V	78
p-value	0.002218

Table 9.48. Wilcoxon test results comparing the performance of the GEBA and GEEA on ITC 2007 instances

From the results in Table 9.48, the p-value of 0.002218 is less than level of significance α of 0.05 adopted for the test. This means that the *null* hypothesis which states that the performance of the GEBA and GEEA is the same can be rejected and the alternative hypothesis accepted. And based on

the results in Table 9.47, it can therefore be concluded that GEEA outperforms GEBA on the ITC 2007 benchmark set.

The next section compares the performance of the GEBA and GEEA on some benchmark instances from the capacitated vehicle routing problem domain.

9.3.2.2 Capacitated vehicle routing problem

Instance	Initial	GEBA	GEEA	$\Delta(\%)$
1	785.30	575.20	**524.61**	8.80
2	1277.20	880.10	**839.52**	4.61
3	1290.35	870.65	**844.72**	2.98
4	1680.20	1220	**1044.28**	14.40
5	1770.04	1502.10	**1318.25**	12.24
6	780.20	585.0	**555.43**	5.10
7	1250.30	1060.40	**913.22**	13.88
8	1265.02	1100.20	**868.35**	21.07
9	1500.30	1260	**1172.74**	6.93
10	1810.02	1580.60	**1404.30**	11.15
11	1370.50	1098.20	**1042.87**	5.04
12	1100.90	877.1	**831.29**	5.21
13	1830.40	1600.10	**1556.94**	2.70
14	1100.60	877.03	**869.04**	0.91

$\Delta(\%)$ is the percentage of improvement in the objective values of the solutions obtained by GEEA over those obtained by GEBA. The best objective values are shown in bold.

Table 9.49. Comparison of the performance of the best GEBA and GEEA generated perturbative heuristic on Christofides instances

The Wilcoxon signed rank test was used to evaluate the statistical significance of the results obtained in Table 9.49. The level of significance α of 0.05 was adopted for the test. Table 9.50 shows the results

of the statistical test.

Wilcoxon signed rank test	
V	105
p-value	0.0009815

Table 9.50. Wilcoxon test results comparing the performance of the GEBA and GEEA on Christofides instances

From the results in Table 9.50, the p-value of 0.0009815 is less than level of significance α of 0.05 adopted for the test. This means that the *null* hypothesis which states that the performance of the GEBA and GEEA is the same can be rejected and the alternative hypothesis accepted. And based on the results in Table 9.49, it can therefore be concluded that GEEA outperforms GEBA on the Christofides instances.

Apart from the Christofides instances, both GEBA and GEEA were applied to the Golden instances. The results showing the performance of the two approaches on the Golden instances are presented in Table 9.51.

Instance	Initial	GEBA	GEEA	$\Delta(\%)$
1	7980.20	6200.80	**5761.06**	7.09
2	11920.42	9110.40	**8601.05**	5.59
3	14090.50	12010.3	**11307.28**	5.85
4	18042.70	16540.20	**14007.31**	15.31
5	9830.95	7700.10	**6707.84**	12.89
6	10574.34	9080.65	**8700.49**	4.18
7	12730.01	10980.60	**10253.17**	6.62
8	15995	13740.10	**11754.68**	14.45
9	860.25	685.30	**584.30**	14.74
10	920.64	810.40	**740.34**	8.65
11	1280.12	970.10	**921.22**	5.04
12	1520.95	1220.30	**1114.13**	8.70
13	1100.62	877.20	**858.50**	2.13
14	1520.04	1160.10	**1000.19**	13.78
15	1900.85	1490.70	**1352.17**	9.29
16	1950.62	1724.20	**1641.18**	4.78
17	950.36	760.0	**708.80**	7.03
18	1210.10	1120.60	**1001.66**	10.61
19	1890.42	1550.63	**1369.19**	12.09
20	2100.30	1980.40	**1822.65**	7.97

$\Delta(\%)$ is the percentage of improvement in the objective values of the solutions obtained by GEEA over those obtained by GEBA. The best objective values are shown in bold.

Table 9.51. Comparison of the performance of the best GEBA and GEEA generated perturbative heuristic on Golden instances

The statistical significance of the results obtained in Table 9.51 was evaluated using the Wilcoxon signed rank test with the level of significance α of 0.05. Table 9.52 shows the results of the statistical

test.

Wilcoxon signed rank test	
V	210
p-value	8.857e-05

Table 9.52. Wilcoxon test results comparing the performance of the GEBA and GEEA on Golden instances

From the results in Table 9.52, the p-value of 8.857e-05 is less than level of significance α of 0.05 adopted for the test.This means that the *null* hypothesis which states that the performance of the GEBA and GEEA is the same can be rejected and the alternative hypothesis accepted. And based on the results in Table 9.51, it can therefore be concluded that GEEA outperforms GEBA on the Golden instances.

The next section compares the performance of the GEBA and GEEA on some benchmark instances from the boolean satisfiability problem domain.

9.3.2.3 Boolean satisfiability problem

Benchmark set		GEBA		GEEA	
		Success Rate (%)	# Variable Flips	Success Rate (%)	# Variable Flips
Suite A	40	**100**	1240	**100**	1080
	50	93.60	6230	**98.20**	4320
	100	75.00	38120	**89.00**	32040
Suite B	50	94.50	12630	**97.30**	10520
	75	81.30	29120	**89.40**	26530
	100	60.70	30440	**73.50**	28102

The success rate (SR) is the percentage of runs where a solution is found within n variable flips. The best success rates are shown in bold.

Table 9.53. Comparison of the performance of the best GEBA and GEEA generated perturbative heuristic on Gottlieb SAT instances

The statistical significance of the results obtained in Table 9.53 was evaluated using the Wilcoxon signed rank test with the level of significance α of 0.05. Table 9.54 shows the results of the statistical test.

Wilcoxon signed rank test	
V	0
p-value	0.04311

Table 9.54. Wilcoxon test results comparing the performance of the GEBA and GEEA on Gottlieb SAT instances

From the results in Table 9.54, the p-value of 0.04311 is less than level of significance α of 0.05 adopted for the test.This means that the *null* hypothesis which states that the performance of the GEBA and GEEA is the same can be rejected and the alternative hypothesis accepted. And based on the results in Table 9.53, it can therefore be concluded that GEEA outperforms GEBA on the Gottlieb instances.

The next section compares the performance of the perturbative heuristics evolved by the GEEA with the human-derived perturbative heuristics.

9.3.3 GEEA heuristics vs Human-derived heuristics

In this section, the results obtained by the best perturbative heuristic evolved by the GEEA are presented and compared with the commonly used human-derived perturbative heuristics for the three problem domains. As mentioned in Section 9.2.2, all the human-derived heuristics were specifically implemented, tested and applied to the same initial solutions that the perturbative heuristic evolved by the GEEA was applied to (see Section 9.2.1). The performance of each heuristic was determined by repeatedly applying the heuristic to the initial solution until the objective value of the solution could not be improved any further. The termination criterion for each run was empirically set to 30 non-improving heuristic application steps. The results obtained by each heuristic after the termination of the run were captured and are presented in Section 9.3.3.1, Section 9.3.3.2 and Section 9.3.3.3. The Friedman test with the level of significance α of 0.05 was used to evaluate the statistical significance

of the results obtained by each approach. Whenever the results were found to be significantly different, a posthoc pairwise test using the Finner method with the GEEA as the control method was carried out to determine which of the approaches performed differently.

The next section presents and discusses the results obtained by the perturbative heuristic evolved by the GEEA and the human-derived perturbative heuristics on the examination timetabling problem.

9.3.3.1 Examination timetabling problem

The human-derived perturbative heuristics considered for this domain are the same as those described in Section 6.6. The heuristics are also represented in Table 9.55. A comparison of the results obtained by each heuristic is given in Table 9.56 with the statistical test results presented in Table 9.57, Table 9.58 and Table 9.59.

Heuristic	Description
phe1	Moves a randomly selected exam to a new feasible timeslot
phe2	Swaps the timeslots of two randomly selected exams if feasible
phe3	Swaps the exams in two randomly selected timeslots
phe4	Randomly exchanges the timeslots of three randomly selected exams
phe5	Moves the exam causing the highest soft constraint violation to a new feasible timeslot
phe6	Moves two randomly selected exams to new feasible timeslots
phe7	Applies the Kempe chain operator to a randomly selected exam and timeslot
phe8	Moves a randomly selected exam to a new randomly selected room if feasible
phe9	Swaps the rooms of two randomly selected exams if feasible

Table 9.55. Human-derived perturbative heuristics for the examination timetabling problem

Instance	GEEA	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
1	**4901**	6230	6300	6224	6100	5980	6080	6120	6200	6120
2	**476**	525	530	510	520	502	510	510	532	530
3	**9064**	10009	10920	10040	10752	10138	10140	10620	10830	10300
4	**16432**	17742	17368	17340	17200	17120	17500	17140	17220	17420
5	**3432**	4272	4117	4100	4084	4090	4480	4170	4034	4200
6	**26577**	28320	28154	28120	28080	28210	28430	28540	28610	28300
7	**5434**	6100	6030	6420	6002	6300	6410	6400	6420	6080
8	**7820**	8410	8500	8620	8410	8360	8430	8540	8430	8360
9	**1140**	1480	1500	1430	1392	1400	1420	1430	1420	1440
10	**20958**	22010	21980	21890	22070	21740	22110	22050	22130	22080
11	**39420**	41020	40980	40920	40990	40870	41030	40920	41010	40900
12	**6354**	6840	6730	6960	6720	6730	6650	6820	6790	6750

The results show the objective values of the solutions obtained by each heuristic. The best values are shown in bold.

Table 9.56. Performance comparison of the best perturbative heuristic generated by the GEEA and the human-derived heuristics on ITC 2007 instances

To evaluate the statistical significance of the results presented in Table 9.7, the non-parametric Friedman test with the post hoc Finner test was carried out. Table 9.8 shows the results of the Friedman's rank sum test performed in order to test the first (*null*) hypothesis which states that all the heuristics perform equally. The level of significance α of 0.05 was used for the test.

Friedman's rank sum test	
Friedman's chi-squared	48.676
df	9
p-value	1.909e-07

Table 9.57. Friedman test results for the GEEA and human-derived perturbative heuristics on ITC 2007 instances

From the results obtained in Table 9.57, the p-value (1.909e-07) is less than the level of significance α of 0.05. This denotes that there is at least one perturbative heuristic that performs differently from the rest. As a result, the *null* hypothesis was rejected and a posthoc pairwise comparison test using the Finner method with the GEEA evolved perturbative heuristic as the control method was performed. The results for the posthoc Finner test showing the differences in performance between the GEEA evolved heuristic and the human-derived heuristics are shown in Table 9.58.

	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
GEEA	1.62e-5	4.93e-5	2.91e-4	2.56e-2	1.55e-1	7.05e-5	1.23e-4	1.13e-5	2.53e-4

Table 9.58. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

The results presented in Table 9.58 show that there is a significant difference in the performance of the perturbative heuristic evolved by the GEEA and most of the human-derived perturbative heuristics except for phe5 (p-value = 1.55e-1). To show whether the perturbative heuristics evolved by the GEEA performs better than all the human-derived heuristics as can be seen from the results in Table 9.56, the constrast estimation method based on medians was used. As mentioned earlier, in this test the performance of the heuristics is reflected by the value of the differences in the objective values of their solutions. A negative value for the heuristic in a given row indicates that the heuristic performs better than the heuristic in a given column. The results for the contrast estimation test are presented in Table 9.59.

	phe1	phe2	phe3	phe4	phe5	phe6	phe7	phe8	phe9
GEEA	-895.50	-870.70	-866.35	-816.95	-776.90	-905.00	-868.05	-886.45	-857.10

Table 9.59. Results for the contrast estimation test on ITC 2007 instances

From Table 9.59, it can be concluded that the perturbative heuristic evolved by the GEBA performs better than the human-derived heuristics on the ITC 2007 benchmark based on the negative values obtained with respect to the differences in the objective values of their solutions.

The next section presents the results for the capacitated vehicle routing problem.

9.3.3.2 Capacitated vehicle routing problem

This section presents the results obtained by applying the perturbative heuristic evolved by the GEEA and the human-derived perturbative heuristics to the Christofides and Golden instances. The human-derived perturbative heuristics considered for this problem domain are described in Table 9.60. A comparison of the results obtained by each heuristic is presented in Table 9.61 and Table 9.65. The statistical test results are given in Table 9.62, Table 9.63 and Table 9.64.

Heuristic	Description
phv1	Moves a randomly selected customer to a new feasible route
phv2	Swaps the routes of two randomly selected customers
phv3	Reverses a part of a tour between two selected customers on a randomly selected route
phv4	Randomly exchanges the routes of three randomly selected customers
phv5	Applies the 2-opt operator on a randomly selected route
phv6	Applies the 2-opt operator on all routes
phv7	Swaps the first portions of two randomly selected distinct routes
phv8	Swaps the adjacent customers of two customers selected from two distinct routes
phv9	Swaps the first and last portions of two randomly selected distinct routes
phv10	Move a randomly selected customer to another position in the same route

Table 9.60. Human-derived perturbative heuristics for the capacitated vehicle routing problem

Instance	GEEA	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
1	**525**	590	582	585	590	580	585	610	602	600	580
2	**840**	908	914	912	910	912	898	910	918	902	901
3	**845**	908	910	914	913	916	910	904	920	905	920
4	**1044**	1280	1270	1290	1272	1280	1294	1280	1272	1290	1280
5	**1318**	1530	1532	1544	1540	1520	1540	1530	1540	1544	1522
6	**555**	640	645	663	631	621	610	648	625	632	655
7	**913**	1128	1111	1090	1100	1136	1090	1109	1116	1110	1125
8	**868**	1180	1168	1125	1157	1171	1170	1160	1165	1151	1157
9	**1173**	1323	1389	1366	1356	1360	1340	1370	1375	1350	1365
10	**1404**	1702	1658	1694	1675	1654	1670	1655	1660	1667	1651
11	**1043**	1202	1210	1198	1166	1150	1180	1214	1186	1191	1160
12	**831**	950	910	911	900	900	901	936	915	926	925
13	**1557**	1626	1640	1681	1646	1630	1627	1646	1641	1634	1632
14	**869**	910	921	900	902	896	891	900	896	910	916

The results show the objective values of the solutions obtained by each heuristic. The best values are shown in bold.

Table 9.61. Comparison of the performance of the best perturbative heuristic generated by the GEEA and the human-derived heuristics on Christofides instances

As in Section 9.2.2.1, the non-parametric Friedman test with the post hoc Finner test was used to evaluate the statistical significance of the results presented in Table 9.61. The results for Friedman's rank sum test performed in order to test the first (*null*) hypothesis are presented in Table 9.62. The level of significance α of 0.05 was used for the test.

Friedman's rank sum test	
Friedman's chi-squared	44.603
df	10
p-value	2.564e-06

Table 9.62. Friedman test results for the GEEA and human-derived perturbative heuristics on Christofides instances

From Table 9.62, it can be seen that the p-value of 2.564e-06 is significantly less than the level of significance α of 0.05 adopted for the test which indicates that there is a significant difference in the performance of the heuristics. As explained earlier, the Friedman test does not show which heuristics perform differently and therefore a posthoc pairwise test using the Finner method with the GEEA evolved heuristic as the control method was performed to show which heuristics perform differently. The results of the posthoc test are shown in Table 9.63.

phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
2.03e-5	2.03e-5	8.73e-6	5.80e-4	4.26e-3	7.05e-3	2.03e-5	1.26e-5	7.07e-5	5.22e-4

Table 9.63. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

From Table 9.63, it can be seen that there is a significant difference between the performance of the perturbative heuristic evolved by the GEEA and all the human-derived heuristics as all the p-values are less than the level of significance α of 0.05. In order to confirm that the perturbative heuristic evolved by the GEEA performs better than all the human-derived heuristics (see Table 9.61), a contrast estimation test based on medians was performed. The results for the contrast estimation test are presented in Table 9.64.

	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
GEEA	-123.9	-122.6	-123.7	-114.2	-111.1	-109.9	-123.58	-120.28	-118.8	-115.5

Table 9.64. Results for the contrast estimation test on Christofides instances

From Table 9.64, the value of the difference in the objective value of the solutions obtained by the GEEA generated heuristic and all the human-derived heuristics are negative. This indicates that the perturbative heuristic evolved by the GEEA performs better than the human-derived heuristics on the Christofides instances. This therefore confirms that the statistical significance of the results observed from Table 9.61).

Apart from the Christofides benchmark set, the performance of the perturbative heuristics was also evaluated on the Golden benchmark set. The objective values of the solutions obtained by each heuristic are presented in Table 9.65.

Instance	GEEA	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
1	**5761**	6745	6796	6900	6800	6100	6840	6750	6651	6740	6800
2	**8601**	10500	10600	10570	10750	10450	10420	10640	10860	10710	10800
3	**11307**	13300	13170	13230	13100	13120	13090	13120	13400	13280	13190
4	**14007**	16100	16970	16960	17200	16980	17070	16985	16874	17050	17080
5	**6708**	8400	8390	8450	8320	8520	8290	8321	8450	8420	8460
6	**8700**	9580	9635	9700	9520	9600	9710	9650	9430	9500	9460
7	**10253**	11440	11875	11687	11390	11240	11630	11540	11650	11510	11452
8	**11755**	14870	14900	14960	14350	14520	14980	14784	14965	14785	14650
9	**584**	780	795	790	784	720	730	732	770	780	762
10	**740**	840	846	860	850	855	871	853	850	857	851
11	**921**	1090	1066	1060	1070	1050	1095	1065	1080	1060	1070
12	**1114**	1260	1250	1267	1301	1251	1250	1265	1258	1243	1255
13	**859**	894	890	892	886	895	893	880	884	882	881
14	**1000**	1204	1203	1226	1205	1210	1202	1220	1210	1211	1206
15	**1352**	1520	1542	1522	1530	1520	1510	1512	1508	1520	1510
16	**1642**	1810	1806	1820	1808	1800	1820	1810	1808	1806	1802
17	**708**	810	830	825	801	815	814	818	815	810	812
18	**1002**	1140	1150	1145	1150	1156	1146	1164	1138	1135	1130
19	**1363**	1660	1670	1650	1651	1672	1650	1665	1666	1642	1640
20	**1823**	2020	2010	2022	2016	2020	2010	2012	2015	2020	2022

The results show the objective values of the solutions obtained by each heuristic. The best objective values are shown in bold.

Table 9.65. Comparison of the performance of the best perturbative heuristic generated by the GEEA and the human-derived heuristics on Golden instances

The non-parametric Friedman test with the post hoc Finner test was also used to evaluate the statistical significance of the results presented in Table 9.65. The results for Friedman's rank sum test performed

in order to test the first (*null*) hypothesis are presented in Table 9.66.

Friedman's rank sum test	
Friedman's chi-squared	59.723
df	10
p-value	4.089e-09

Table 9.66. Friedman test results for the GEEA and human-derived perturbative heuristics on Christofides instances

From Table 9.66, it can be seen that the p-value of 4.089e-09 is significantly less than the level of significance α of 0.05 adopted for the test which indicates that there is a significant difference in the performance of the heuristics. A posthoc pairwise test using the Finner method with the GEEA evolved heuristic as the control method was therefore performed to show which heuristics perform differently. The results of the posthoc test are shown in Table 9.67.

phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
4.95e-6	6.7e-7	7.96e-11	3.51e-6	1.44e-5	2.54e-6	1.71e-6	1.9e-6	2.6e-5	2.3e-05

Table 9.67. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

From Table 9.67, it can be seen that there is a significant difference between the performance of the perturbative heuristic evolved by the GEBA and all the human-derived heuristics since all the p-values are less than the level of significance α of 0.05. In order to confirm that the the perturbative heuristics generated by GEEA outperforms all the human-derived perturbative heuristics (see Table 9.65), a contrast estimation test based on medians was performed. The results for the contrast estimation test are presented in Table 9.68.

	phv1	phv2	phv3	phv4	phv5	phv6	phv7	phv8	phv9	phv10
GEBA	-201.6	-206.5	-213.3	-202.3	-200.8	-202.8	-204.5	-203	-200.7	-200.6

Table 9.68. Results for the contrast estimation test on Golden instances

From Table 9.68, the value of the difference in the objective value of the solutions obtained by the GEEA generated heuristic and all the human-derived heuristics are negative. This indicates that the

perturbative heuristic evolved by the GEEA performs better than the human-derived heuristics on the Golden instances. This confirms that the statistical significance of the results observed from Table 9.65) which shows that GEEA obtains the best results and therefore performs better than all the human-derived perturbative heuristics on the Golden instances.

The next section presents the results obtained by the perturbative heuristic evolved by the GEEA and the human-derived perturbative heuristics on benchmark sets for the the boolean satisfiability problem.

9.3.3.3 Boolean satisfiability problem

This section presents the results obtained by applying the perturbative heuristic evolved by the GEEA and the human-derived perturbative heuristics to Gottlieb SAT benchmark sets for the boolean satisfiability problem. The human-derived perturbative heuristics considered for the Boolean satisfiability problem are described in Table 9.69. A comparison of the results obtained by the heuristics is presented in Table 9.70 and the statistical test results are given in Table 9.71.

Heuristic	Description
GWSAT(p)	Flips a random variable with the highest net gain in a randomly selected unsatisfied clause with some probability p. The net gain of a variable is number of clauses that remain unsatisfied in the boolean expression when the variable is flipped. If two or more variables have the same net gain, the ties are randomly broken.
Walksat(p)	Selects a random unsatisfied clause and randomly flips any variable with a net gain of 0 in the clause. Otherwise it uses some probability p to select a random variable to flip from the clause.
Novelty+(p,p_w)	Selects a random unsatisfied clause and with a probability p_w selects a random variable to flip. Otherwise, it flips the variable with the maximal net gain unless the variable has a minimal age. The age of the variable is the number of other variable flips that have occurred since the variable was last flipped. If the variable has minimal age in the unsatisfied clause, then the variable is selected for flipping with probability (1-p).

Table 9.69. Human-derived perturbative heuristics for the SAT problem

Benchmark set		GEEA		GWSAT(0.5)		WalkSAT(0.5)		Novelty+(0.7, 0.01)	
		SR(%)	AF	SR(%)	AF	SR(%)	AF	SR(%)	AF
Suite A	40	**100**	1080	**100**	6062	**100**	2604	**100**	1767
	50	98.20	4320	90.33	31295	99.67	22219	**100**	3061
	100	89.00	32040	71.00	10104	81.00	24855	**90.0**	33148
Suite B	50	**97.30**	10520	89.56	20841	94.4	18609	96.64	13783
	75	**89.40**	26530	71.74	28584	82.48	32939	89.3	25833
	100	**73.50**	28102	56.6	17704	59.16	21824	65.8	24655

SR is the success rate and AF is the average number of flips to solution. The values for the success rate and number of flips as well as the parameter values for GWSAT, WalkSAT and Novelty+ were obtained from [35]. The best results are shown in bold.

Table 9.70. Comparison of the performance of the best perturbative heuristic generated by the GEEA and the human-derived heuristics on Gottlieb SAT instances

The non-parametric Friedman test with the post hoc Finner test was also used to evaluate the statistical significance of the results presented in Table 9.70. The results for Friedman's rank sum test performed in order to test the first (*null*) hypothesis which states that all the heuristics perform equally are presented in Table 9.71. The level of significance α of 0.05 was used for the test.

Friedman's rank sum test	
Friedman's chi-squared	11.88
df	3
p-value	0.007806

Table 9.71. Friedman test results for the GEEA and human-derived perturbative heuristics on Gottlieb SAT instances

From Table 9.71, it can be seen that the p-value of 0.007806 is less than the level of significance α of 0.05 adopted for the test. This indicates that there is at least one heuristic that performs differently. The *null* hypothesis was therefore rejected and the pairwise posthoc finner test with the GEEA as the control method was performed in order to show which heuristics perform differently. The results of the finner test are presented in Table 9.72.

	GWSAT(0.5)	WalkSAT(0.5)	Novelty+(0.7, 0.01)
GEEA	0.04295262	0.3271284	1.00000000

Table 9.72. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

The pairwise finner test results in Table 9.72 shows that there is no significant difference between the GEEA and the human-derived heuristics except for GWSAT(p-value =0.04295262). It is therefore not possible to determine how the performance of the perturbative heuristic evolved by the GEEA compares to that of the human-derived perturbative heuristics using the Finner test. A contrast estimation test based on medians was therefore carried out. The results of the contrast estimation test are presented in Table 9.73.

	GWSAT(0.5)	WalkSAT(0.5)	Novelty+(0.7, 0.01)
GEBA	-11.56375	-4.90125	-0.88000

Table 9.73. Results for the contrast estimation test on Gottlieb SAT instances

From Table 9.73, it can be seen that the GEEA outperforms all the human-derived heuristics since all the p-values are negative.

The next section compares the GEEA to other existing hyper-heuristics.

9.3.4 GEEA heuristics vs Other Hyper-heuristics

This section compares the results obtained by GEEA to existing hyper-heuristic approaches in the literature. The results obtained by each approach are presented in the following sections.

9.3.4.1 Examination timetabling problem

The hyper-heuristics considered for this problem domain are the same as those considered for the GEBA and these are represented in Table 9.74. A comparison of the results obtained by each approach is presented in Table 9.75 while the statistical test results are given in Table 9.76, Table 9.77 and Table 9.78.

Hyper-heuristic	Description
ETP-HH1	A generation perturbative hyper-heuristic proposed by Sabar et al. [40].
ETP-HH2	A selection perturbative hyper-heuristic proposed by Swan et al. [63].
ETP-HH3	A selection perturbative hyper-heuristic proposed by Burke et al. [64].
ETP-HH4	A selection perturbative hyper-heuristic proposed by Anwar et al. [65].

Table 9.74. Selected hyper-heuristics for the examination timetabling problem

Instance	GEEA	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
1	4901	**4362**	5875	6235	11823
2	476	**380**	893	2974	976
3	9064	**8991**	13352	15832	26770
4	16432	**15094**	24174	35106	—
5	3432	**2912**	4734	4873	6772
6	26577	**25735**	27510	31756	30980
7	5434	**4025**	5731	11562	11762
8	7820	**7452**	9507	20994	16286
9	1140	**1111**	—	—	—
10	17958	**14825**	—	—	—
11	32420	**28891**	—	—	—
12	6354	**6181**	—	—	—

The comparison is between the objective values for the best solutions obtained for each approach. The best values are shown in bold. "—" indicates that the values were not provided or a feasible solution was not obtained.

Table 9.75. Performance comparison of the GEEA and other HHs on ITC 2007 instances

The Friedman test was used to evaluate the statistical significance of the results presented in Table 9.75. The results of this test are presented in Table 9.76.

Friedman's rank sum test	
Friedman's chi-squared	26.629
df	4
p-value	2.363e-05

Table 9.76. Friedman test results for the GEEA and other hyper-heuristics on ITC 2007 instances

From the results obtained in Table 9.76, the p-value (2.363e-05) is less than the level of significance α of 0.05. This denotes that there is at least one approach that performs differently from the rest. As a result, the *null* hypothesis was rejected and a posthoc pairwise comparison test using the Finner method with the GEEA as the control method was performed. The results for the posthoc Finner test showing the differences in performance between the GEEA and other hyper-heuristics are shown in Table 9.77.

	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
GEEA	0.2806138750	0.28061387	0.0088495160	0.0282863663

Table 9.77. P-values for the posthoc pairwise Finner test with the GEEA as the control method

The results presented in Table 9.77 show that there is a significant difference in the performance of the GEEA and ETP-HH3 (p-value = 0.0088495160) as well as ETP-HH4 (p-value = 0.0282863663). There is however no significant difference in the performance of the GEEA and the remaining hyper-heuristics, namely ETP-HH1 (p-value = 0.2806138750) and ETP-HH2 (p-value = 0.28061387). To show whether GEEA performs betters than ETP-HH3 and ETP-HH4, a constrast estimation test based on medians was performed. As already mentioned, in this test the performance of the heuristics is reflected by the value of the differences in the objective values of their solutions and a negative value for the heuristic in a given row indicates that the heuristic performs better than the heuristic in a given column. The results for the contrast estimation test are presented in Table 9.78.

	ETP-HH1	ETP-HH2	ETP-HH3	ETP-HH4
GEEA	624.250	-1458.875	-5427.375	-3650.430

Table 9.78. Results for the contrast estimation test

From the results in Table 9.78, it can be concluded that the GEEA performs better than ETP-HH2, ETP-HH3 and ETP-HH4, based on the negative value of the difference in the objective value of their solutions, but is outperformed by ETP-HH1. This performance by the GEEA can be attributed to the fact the perturbative heuristic evolved by the GEEA is applied as it is with no optimization as is the case for perturbative heuristics evolved by ETP-HH1. In addition, ETP-HH1 uses a wider variety of move acceptance criteria which helps in the search for better performing heuristics.

The next section presents the results for the capacitated vehicle routing problem.

9.3.4.2 Capacitated vehicle routing problem

This test also uses the same hyper- heuristics considered for the GEBA which are represented in Table 9.79. A comparison of the results obtained by each approach is given in Table 9.61 and Table 9.65 respectively. The statistical test results are given in Table 9.81 and Table 9.86.

Hyper-heuristic	Description
CVRP-HH1	A generation perturbative hyper-heuristic proposed by Sabar et al. [40].
CVRP-HH2	A selection perturbative hyper-heuristic proposed by Garrido et al. [66].
CVRP-HH3	A selection perturbative hyper-heuristic proposed by Meignan et al. [67].
CVRP-HH4	A selection perturbative hyper-heuristic proposed by Pisinger et al. [76].

Table 9.79. Selected hyper-heuristics for the vehicle routing problem

Instance	GEEA	CVRP-HH1	CVRP-HH2	CVRP-HH3
1	0.00	**0.00**	**0.00**	**0.00**
2	0.51	**0.00**	0.05	0.62
3	2.25	**0.00**	0.21	0.42
4	1.54	**0.11**	0.52	2.05
5	2.08	**1.33**	2.05	5.07
6	0	**0.00**	**0.00**	—
7	0.39	**0.00**	0.09	—
8	0.28	**0.00**	**0.00**	—
9	0.81	**0.20**	0.70	—
10	0.60	**0.53**	1.24	—
11	0.07	**0.00**	0.88	0.00
12	1.43	**0.00**	**0.00**	—
13	1.03	1.90	1.00	**0.00**
14	0.31	**0.00**	**0.00**	—

The values represent the percentage deviation from the best known results in the literature. The best values are shown in bold. "—" indicates that the values were not provided or a feasible solution was not obtained.

Table 9.80. Performance comparison of the GEEA and other HHs on Christofides instances

Friedman's rank sum test	
Friedman's chi-squared	5.4407
df	3
p-value	0.1422

Table 9.81. Friedman test results for the GEEA and other hyper-heuristics on Christofides instances

From the results in Table 9.81, it can be seen that when CVRP-HH3 (which has missing values) is included in the test, the p-value of 0.1422 is greater than the level of significance α of 0.05 adopted for the test which indicates that there is not enough evidence to conclude that there is a significant difference in the performance of the heuristics. In this case, the *null* hypothesis which states that the

approaches perform equally cannot be rejected. However, when the CVRP-HH3 is omitted, the test shows that there is a significant difference in the performance of the approaches as can be seen in Table 9.82.

Friedman's rank sum test	
Friedman's chi-squared	13.644
df	2
p-value	0.001089

Table 9.82. Friedman test results for the GEEA and other hyper-heuristics (without CVRP-HH3) on Christofides instances

For the Christofides instances, only the results obtained by the GEEA, CVRP-HH1 and CVRP-HH2 will therefore be evaluated for statistical significance. And since the Friedman testreveals that there is a significant difference in the performance of the three approaches, the posthoc pairwise finner test with the GEEA as the control method was performed. The results of the posthoc test are presented in Table 9.83.

	CVRP-HH1	CVRP-HH2
GEEA	0.002824454	0.1069002

Table 9.83. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

From the results in Table 9.83, it can be seen that there is a significant difference, based on the p-values, in the performance of the GEEA and CVRP-HH1 but no significant difference in performance of the GEEA and CVRP-HH2. To determine the best approach among the three approaches, a contrast estimation method based on medians was also used. The results for the contrast estimation test are presented in Table 9.84.

	CVRP-HH1	CVRP-HH2
GEBA	0.32166667	0.22333333

Table 9.84. Results for the contrast estimation test on Christofides instances

From the results in Table 9.84, the value of the difference in the objective value of the solutions obtained by the GEEA generated heuristic and other hyper-heuristics are all positive. This indicates

that the two hyper-heuristic approaches, namely CVRP-HH1 and CVRP-HH4 performed better than the GEEA on the Christofides instances. The poor performance by the GEEA can be attributed to the fact that the heuristic evolved by the GEEA was applied with no optimization as is usually done with most perturbative heuristics. In addition, the GEEA, although considered an improvement on the GEBA, did not include a wider variety of move acceptance criteria as is used in CVRP-HH1.

Instance	**GEEA**	CVRP-HH1	CVRP-HH4
1	5761.06	**5626.81**	5650,91
2	8601.05	**8446.19**	8469,32
3	11307.28	11081.60	**11047,01**
4	14007.31	13658.84	**13635,31**
5	6707.84	**6460.98**	6466,68
6	8700.49	8462.10	**8416,13**
7	10253.17	10202.24	**10181,75**
8	11754.68	**11690.82**	11713,62
9	584.30	**583.39**	585,14
10	740.34	**740.91**	748,89
11	921.22	**919.80**	922,7
12	1114.13	**1111.43**	1119,06
13	858.50	**857.19**	864,68
14	1000.19	**1083.59**	1095,4
15	1352.17	**1350.17**	1359,94
16	1641.18	**1631.91**	1639,11
17	708.80	**707.76**	708,9
18	1001.66	**1003.43**	1002,42
19	1369.19	**1368.12**	1374,24
20	1822.65	**1820.09**	1830,8

The comparison is between the best values obtained for each approach. The best values are shown in bold.

Table 9.85. Performance comparison of the GEEA and other HHs on Golden Instances

Friedman's rank sum test	
Friedman's chi-squared	10.8
df	2
p-value	0.004517

Table 9.86. Friedman test results for the GEEA and other hyper-heuristics on Golden instances

From the results in Table 9.86, it can be seen that the p-value of 0.004517 is significantly less than the level of significance α of 0.05 adopted for the test which indicates that there is a significant difference in the performance of the approaches. A posthoc pairwise test using the Finner method with the GEEA as the control method was therefore performed to show CVRP-HH1 and CVRP-HH4 performed against the GEEA. The results of the test are given in Table 9.87.

	CVRP-HH1	CVRP-HH4
GEEA	0.01322088	1.00000000

Table 9.87. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

The results in Table 9.87 show that there is a significant difference, based on the p-values, in the performance of the GEEA and CVRP-HH1, but no significant difference between the GEEA and CVRP-HH4. A contrast estimation method based on medians was therefore used to show which of the approaches performed better than the other. The results for the contrast estimation test are presented in Table 9.88.

	CVRP-HH1	CVRP-HH4
GEEA	3.83	-1.63

Table 9.88. Results for the contrast estimation test on Golden instances

From the results in Table 9.88, it can be concluded that the GEEA performs better than CVRP-HH4, but as is the case in the other domains, is outperformed by CVRP-HH1. This performance by the GEEA can also be attributed to the fact that the heuristic evolved by the GEEA was applied with no optimization as is usually done with most perturbative heuristics.

The next section presents and compares the results obtained by the GEEA and other hyper-heuristic approaches on the boolean satisfiability problem.

9.3.4.3 Boolean satisfiability problem

As in Section 9.2.3.3, the generation perturbative hyper-heuristics considered in this test are presented in Table 9.89. A comparison of the results obtained by each approach is presented in Table 9.90 with the statistical test results given in Table 9.91.

Hyper-heuristic	Description
SAT-HH1	A generation perturbative hyper-heuristic proposed by Bader and Poli [49].
SAT-HH2	A generation perturbative hyper-heuristic proposed by Fukunanga [35].

Table 9.89. Selected hyper-heuristics for the boolean satisfiability problem

Benchmark set		GEEA		SAT-HH1		SAT-HH2	
		SR(%)	AF	SR(%)	AF	SR(%)	AF
Suite A	40	**100**	1080	—	—	**100**	538
	50	98.20	4320	**100**	12872	**100**	3416
	100	89.00	32040	92.00	54257	**100**	20754
Suite B	50	97.30	10520	**100.00**	18936	99.38	6590
	75	89.40	26530	95.00	26571	**98.48**	19791
	100	73.50	28102	74.00	41284	**78.8**	39909

The comparison is between the best values for the success rate for each approach. The best values are shown in bold. "—" indicates that the values were not provided.

Table 9.90. Comparison of the performance of the GEEA and other generation perturbative hyper-heuristics on Gottlieb SAT instances

Friedman's rank sum test	
Friedman's chi-squared	8.3158
df	2
p-value	0.01564

Table 9.91. Friedman test results for the GEEA and human-derived perturbative heuristics on Gottlieb SAT instances

From Table 9.91, the p-value of 0.01564 which is less than the level of significance α of 0.05 indicates that there is at least one approach that performs differently. The *null* hypothesis was therefore rejected and the pairwise posthoc finner test with the GEEA as the control method was performed in order to show which approaches perform differently. The results of the posthoc finner test are presented in Table 9.92.

	SAT-HH1	SAT-HH2
GEEA	0.05914993	0.02141386

Table 9.92. P-values for the posthoc pairwise comparison Finner test with the GEEA as the control method

The pairwise finner test results in Table 9.92 shows that there is a significant difference in performance between the GEEA and the two other hyper-heuristics, namely SAT-HH1 and SAT-HH2. It is however not possible to determine which approach is the best among the three using this test. For this reason, a contrast estimation test based on medians was carried out. The results of the contrast estimation test are presented in Table 9.93.

	SAT-HH1	SAT-HH2
GEBA	13.433333	17.046667

Table 9.93. Results for the constrast estimation test on Gottlieb instances

From Table 9.93, it can be seen that the GEEA is outperformed by both the SAT-HH1 and SAT-HH2 hyper-heuristics. It has to be mentioned however that the results for the SAT-HH1 and SAT-HH2 hyper-heuristics presented here are those obtained by the best perturbative heuristic evolved by the approaches. In their respective studies, both the SAT-HH1 and SAT-HH2 evolved more than one

perturbative heuristic and each of the evolved heuristic when applied to the Gottlieb testing instances obtained different results. In this study, the GEEA generates only one perturbative heuristic and therefore the results presented in this section aim to show how close the GEEA results are to the state of the art results obtained by SAT-HH1 and SAT-HH2.

The next section presents the summary to the chapter.

9.4 SUMMARY

This chapter presented the results obtained by the proposed GE approaches for generating perturbative heuristics to solve combinatorial optimization problems. The results obtained by the two approaches, namely GEBA and GEEA on benchmarks sets from the examination timetabling, vehicle routing and boolean satisfiability problem domains were presented. These results were further compared with those obtained by commonly used human-derived perturbative heuristics and other existing generation perturbative hyper-heuristic approaches in the literature. From the results obtained and presentedin the chapter, it can be concluded that the GEEA is a better approach than the GEBA since the perturbative heuristics generated by the former produced better solutions than those generated by latter. The difference in performance between the two approaches was found to be statistically significant at a confidence level of 0.05. When compared to existing human-derived perturbative heuristics, both GEEA and GEBA were found to generate perturbative heuristics that performed better than the human-derived perturbative heuristics for the examination timetabling and vehicle routing benchmark sets. For the boolean satisfiability problem, the two state-of-the-art human-derived perturbative heuristics, namely Walksat and Novelty+, completely outperformed the GEBA but were equally outperformed by the GEEA. When compared to other hyper-heuristics, GEEA was found to perform better than most of the hyper-heuristic approaches especially on the examination timetabling problem. For the capacitated vehicle routing and Boolean satisfiability problems, the GEEA was outperformed by other hyper-heuristic approaches save for the CVRP-HH4. This performance by the GEEA can be attributed to the fact that the perturbative heuristic evolved by the GEEA was applied with no optimization as is usually done with most perturbative heuristics. The other reason is that hyper-heuristics, specifically the SAT-HH1 and SAT-HH2 applied to the Boolean satisfiability problem instances, generated more than one type of perturbative heuristics which were then applied to the same instances in order to determine which one performed better. The different types of heuristics had varying performance

and even though most of them were actually outperformed by the GEEA on some instances, the best performing heuristic among those generated by the two hyper-heuristics obtained results that were superior to those obtained by the GEEA. In general, the GEEA performed exceptionally well given the circumstances and this good performance is a clear demonstration of the viability of the approach. The next chapter presents the conclusion and potential future extensions to this research work.

CHAPTER 10 Conclusion and Future work

10.1 INTRODUCTION

This chapter presents a summary of the research findings. In particular, it provides a conclusion to each research objective described in Chapter 1 based on the experimental results obtained and presented in Chapter 9. Section 10.2 presents and provides concluding remarks on each research objective while section 10.3 presents the overall conclusion of the research. Some insight into potential future work is provided in section 10.4 and the chapter summary follows in section 10.5.

10.2 RESULTS DISCUSSION FOR OBJECTIVES

The main aim of the research conducted in this dissertation was to investigate automating the process of designing perturbative heuristics for solving combinatorial optimization problems. An analysis of existing literature (see Section 6.2) revealed that this was not a well-researched domain with very few research works conducted. Based on this analysis, it was determined that in order to achieve the aforementioned aim, the following objectives had to be met:

- Develop a GE approach to automatically generate perturbative heuristics for more than one problem domain;
- Test the generality of the proposed approach on three different problem domains, namely examination timetabling, vehicle routing and boolean satisfiability;
- Compare the performance of the perturbative heuristics generated by the proposed approach to that of the human-designed move operators for the three problem domains;

- Compare the performance of the perturbative heuristics generated by the proposed approach to that of the perturbative heuristics generated by other generation perturbative hyper-heuristics in the literature.

Please note that although two new approaches, namely *baseline* and *extended* were developed and implemented in this research, the two approaches were aimed at achieving the same goal, i.e. to develop a general approach for generating perturbative heuristics to solve problems from multiple problem domains. The baseline approach was developed to merely test the basic idea of using GE to generate new perturbative heuristics from heuristic components comprising a set of basic operations and components of the solution. It was the initial approach. The extended approach uses the same basic idea as the baseline approach but the grammar has been significantly changed to include new elements and can therefore be considered as an improvement of the baseline approach. Since the process of determining the viability of the two approaches is the same and with both approaches aiming to achieve the same goal, this chapter will focus only on the extended GE approach as this was shown to be the better approach based on the results presented in Chapter 9.

In this research, the new GE approach is used to generate perturbative heuristics for solving problems in three different problem domains, namely examination timetabling, vehicle routing and boolean satisfiability. The results presented in Chapter 9 show that the approach is able to generate perturbative heuristics that performed well on the benchmark sets from the three problem domains. This is a notable success not only in terms of designing an approach that can be easily applied to more than one discrete combinatorial optimization domain but also in terms of generating heuristics that produce good quality solutions. The next sections provide concluding remarks on how each of the research objectives were met.

10.2.1 Results discussion for Objective One

The first objective was to develop a GE approach to generate perturbative heuristics for combinatorial optimization problems. To meet this objective, a new GE approach was developed and implemented. The approach was based on the basic idea of generating heuristics from two heuristic components, namely basic operations and solution components. The assumption was that by separating the basic operations from the components of the solution, it was possible to automatically generate different

types of heuristics by using an evolutionary algorithm such as GE to determine the best combinations of the heuristic components. A grammar specifying how the heuristic components should be selected and combined was defined with GE being used to search for the best combination of the components. This combination represented a perturbative heuristic. For all the problems investigated, the approach successfully generated valid heuristics.

10.2.2 Results discussion for Objective Two

The second objective was to test the generality of the proposed approach by applying it to problems from three different problem domains. The goal here was to investigate whether the same approach can be successfully used to generate heuristics for more than one problem domain. The GE approach was applied to the problems from three different problem domains, namely the examination, vehicle routing and boolean satisfiability problems. Only minor modifications to the specification of solution components were required to make the approach applicable to each problem domain. This made it very easy to apply the same approach to different types of problems without requiring significant modifications. The approach generated valid heuristics for all the problem instances in the three domains.

10.2.3 Results discussion for Objective Three

The third objective was to compare the performance of the perturbative heuristics generated by the proposed GE approach to the performance of the human-derived perturbative heuristics commonly used in the literature. To meet this objective, some commonly used human-derived heuristics for the three problem domains were selected (see Section 6.6) and applied to the same test sets as the best heuristic generated by the GE approach. The non parametric Friedman test was used to determine the statistical significance of the results. For all the problem instances in the examination timetabling and vehicle routing problems investigated, the GE approach generated heuristics that outperformed the human-derived heuristics. This success can be attributed to a number of factors among which is the ability by GE to discover new perturbative heuristics which human beings may not have been able to think of, and the ability of the approach to automatically combine two or more generated perturbative heuristics into one which has the potential to significantly improve the quality of results.

10.2.4 Results discussion for Objective Four

The fourth objective was to compare the performance of the proposed GE approach with other generation perturbative hyper-heuristics in the literature. As mentioned earlier, the field of generation perturbative hyper-heuristics has generally not been that well researched and very few generation perturbative hyper-heuristic approaches have been applied to the problem instances investigated in this research. The existing few hyper-heuristic approaches were compared with the GE approach proposed in this study. Here, the Friedman test was also used to rank the approaches. The results show that the GE approach proposed in this study performed better than most existing hyper-heuristics on most problem instances from the examination timetabling with the exception of the GEHH proposed by Sabar et al [40]. When compared to the hyper-heuristics for the vehicle routing and boolean satisfiability problems, the GE approach performed better than the approach proposed by by Pisinger et al. [76] on the vehicle routing problem as well as the one proposed by Bader and Poli [49] on the boolean satisfiability problem. It was just slightly worse than the other approaches on the vehicle routing problem and the approach proposed by Fukunaga [35] on the boolean satisfiability problem. This performance by the GE approach can be attributed to a number of factors including the fact that the generated heuristics were applied as they are with no optimization techniques employed. GEHH for example makes use of a wider range of move acceptance criteria and also employs online learning through the use of an adaptive memory which significantly improves the quality of solutions discovered by the approach.

10.3 CONCLUSION

Based on the outcomes of the research objectives and the fact that there is still need for more research in the domain of automated generation of perturbative heuristics, the work presented in this dissertation makes a significant contribution to the field in that it proposes a novel approach that can be used to automatically generate perturbative heuristics for any combinatorial optimization problem. The proposed GE approach produced very promising results for the three problem domains investigated and this success is a clear indication that it is possible to automatically generate good quality heuristics without a human expert. In addition, the approach is very flexible and can be easily customised to generate more problem specific perturbative heuristics if necessary. The approach can therefore

be considered as a stepping stone towards the design of better and more efficient algorithms for automatically generating perturbative heuristics to solve combinatorial optimization problems.

10.4 FUTURE WORK

Future extensions to the work presented in this dissertation will include the following:

- Incorporating optimization techniques to investigate whether the results obtained in this work can be improved further. This will involve applying a selection perturbative hyper-heuristic to the perturbative heuristics generated;
- Extending the grammar to investigate whether better heuristics can be discovered by considering the fitness landscape analysis.

10.5 SUMMARY

The chapter presented a summary of the research findings including a discussion of how each of the objectives was met. Some insights into potential future work were also presented.

REFERENCES

[1] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[2] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2013, vol. 76.

[3] M. Baghel, S. Agrawal, and S. Silakari, "Survey of metaheuristic algorithms for combinatorial optimization," *International Journal of Computer Applications*, vol. 58, no. 19, 2012.

[4] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[5] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization*. Prentice Hall Englewood Cliffs, 1982, vol. 24.

[6] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[7] P. Festa, "A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems," in *2014 16th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2014, pp. 1–20.

[8] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[9] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[10] R. E. Bellman *et al.*, "Dynamic programming, ser," in *Rand Corporation research study*. Princeton University Press, 1957.

[11] D. S. Hochba, "Approximation algorithms for np-hard problems," *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[12] C. P. Gomes and R. Williams, "Approximation algorithms," in *Search Methodologies*. Springer, 2005, pp. 557–585.

[13] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[14] D. Dasgupta and Z. Michalewicz, *Evolutionary algorithms in engineering applications*. Springer Science & Business Media, 2013.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[16] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking," *Control and cybernetics*, vol. 29, no. 3, pp. 653–684, 2000.

[17] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[18] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning: great-deluge hyper-heuristic for examination timetabling," in *Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends*. IGI Global, 2012, pp. 34–55.

[19] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *University of Stirling, Tech. Rep*, 2012.

[20] M. Dorigo and T. Stützle, "Ant colony optimization: overview and recent advances," in *Handbook of metaheuristics*. Springer, 2019, pp. 311–351.

[21] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez, "Variable neighborhood search," in *Handbook of metaheuristics*. Springer, 2019, pp. 57–97.

[22] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of metaheuristics*. Springer, 2019, pp. 129–168.

[23] M. Gendreau and J.-Y. Potvin, "Tabu search," in *Handbook of Metaheuristics*. Springer, 2019, pp. 37–55.

[24] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: Revisited," in *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.

[25] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[26] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *proceedings of an International Conference on Genetic Algorithms and the Applications*, 1985, pp. 183–187.

[27] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*. Springer, 2005, pp. 127–164.

[28] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza, "Genetic programming: An introduction and tutorial, with a survey of techniques and applications," in *Computational intelligence: A compendium*. Springer, 2008, pp. 927–1028.

[29] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[30] R. Poli, "A simple but theoretically-motivated method to control bloat in genetic programming," in *European Conference on Genetic Programming*. Springer, 2003, pp. 204–217.

[31] L. Vanneschi and R. Poli, "Genetic programming—introduction, applications, theory and open issues," *Handbook of natural computing*, pp. 709–739, 2012.

[32] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[33] J. R. Woodward and R. Bai, "Why evolution is not a good paradigm for program induction: a critique of genetic programming," in *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, 2009, pp. 593–600.

[34] S. Nguyen, M. Zhang, and M. Johnston, "A genetic programming based hyper-heuristic approach for combinatorial optimisation," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 1299–1306.

[35] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary computation*, vol. 16, no. 1, pp. 31–61, 2008.

[36] S. Mirjalili, "Genetic algorithm," in *Evolutionary Algorithms and Neural Networks*. Springer, 2019, pp. 43–55.

[37] M. Nicolau, "Understanding grammatical evolution: initialisation," *Genetic Programming and Evolvable Machines*, vol. 18, no. 4, pp. 467–507, 2017.

[38] C. Stone, E. Hart, and B. Paechter, "On the synthesis of perturbative heuristics for multiple combinatorial optimisation domains," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 170–182.

[39] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.

[40] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Grammatical evolution hyper-heuristic for combinatorial optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 6, pp. 840–861, 2013.

[41] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *International Conference on the Practice and Theory of Automated Timetabling*. Springer, 2000, pp. 176–190.

[42] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[43] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[44] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*. Springer, 2018.

[45] R. Qu, E. K. Burke, B. McCollum, L. T. Merlot, and S. Y. Lee, "A survey of search methodologies and automated system development for examination timetabling," *Journal of scheduling*, vol. 12, no. 1, pp. 55–89, 2009.

[46] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and multilevel metaheuristics*. Springer, 2008, pp. 3–29.

[47] R. E. Keller and R. Poli, "Self-adaptive hyperheuristic and greedy search," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 3801–3808.

[48] E. Aarts, E. H. Aarts, and J. K. Lenstra, *Local search in combinatorial optimization*. Princeton University Press, 2003.

[49] M. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer,

2007, pp. 37–49.

[50] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical evolution of local search heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 406–417, 2012.

[51] C. Contreras-Bolton and V. Parada, "Automatic design of algorithms for optimization problems," in *2015 Latin America Congress on Computational Intelligence (LA-CCI)*. IEEE, 2015, pp. 1–5.

[52] B. J. Oates, *Researching information systems and computing*. Sage, 2005.

[53] C. Johnson, "What is research in computing science," *Computer Science Dept., Glasgow University. Electronic resource: http://www.dcs.gla.ac.uk/%7Ejohnson/teaching/research_skills/research.html*, 2006.

[54] C. López-Vázquez and E. Hochsztain, "Extended and updated tables for the friedman rank test," *Communications in Statistics-Theory and Methods*, vol. 48, no. 2, pp. 268–281, 2019.

[55] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, 2010.

[56] B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," *Proceedings of the 4th multidisciplinary international scheduling: Theory and applications 2009 (MISTA 2009)*, pp. 424–434, 2009.

[57] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[58] H. H. Hoos and T. Stützle, "Satlib: An online resource for research on sat," *Sat*, vol. 2000, pp. 283–292, 2000.

[59] J. Gottlieb, E. Marchiori, and C. Rossi, "Evolutionary algorithms for the satisfiability problem," *Evolutionary computation*, vol. 10, no. 1, pp. 35–50, 2002.

[60] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[61] E. K. Burke, A. J. Eckersley, B. McCollum, S. Petrovic, and R. Qu, "Hybrid variable neighbourhood approaches to university exam timetabling," *European Journal of Operational Research*, vol. 206, no. 1, pp. 46–53, 2010.

[62] N. Pillay, "A review of hyper-heuristics for educational timetabling," *Annals of Operations Research*, vol. 239, no. 1, pp. 3–38, 2016.

[63] J. Swan, E. Ozcan, and G. Kendall, "Co-evolving add and delete heuristics," in *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, 2012, pp. 395–399.

[64] E. K. Burke, R. Qu, and A. Soghier, "Adaptive selection of heuristics for improving constructed exam timetables," in *Proc. PATAT*, 2010, pp. 136–151.

[65] K. Anwar, A. T. Khader, M. A. Al-Betar, and M. A. Awadallah, "Harmony search-based hyper-heuristic for examination timetabling," in *2013 IEEE 9th International Colloquium on Signal Processing and its Applications*. IEEE, 2013, pp. 176–181.

[66] P. Garrido and C. Castro, "Stable solving of cvrps using hyperheuristics," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 255–262.

[67] D. Meignan, A. Koukam, and J.-C. Créput, "Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism," *Journal of Heuristics*, vol. 16, no. 6, pp. 859–879, 2010.

[68] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session

on real parameter optimization," *Journal of Heuristics*, vol. 15, no. 6, p. 617, 2009.

[69] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[70] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, J. Bassett, R. Hubley, and A. Chircop, "Ecj: A java-based evolutionary computation research system," *Downloadable versions and documentation can be found at the following url: http://cs. gmu. edu/eclab/projects/ecj*, vol. 880, 2006.

[71] Z. Lü, J.-K. Hao, and F. Glover, "Neighborhood analysis: a case study on curriculum-based course timetabling," *Journal of Heuristics*, vol. 17, no. 2, pp. 97–118, 2011.

[72] D. S. Johnson, "A theoretician's guide to the experimental analysis of algorithms," *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, vol. 59, pp. 215–250, 2002.

[73] L. Di Gaspero and A. Schaerf, "Neighborhood portfolio approach for local search applied to timetabling problems," *Journal of Mathematical Modelling and Algorithms*, vol. 5, no. 1, pp. 65–89, 2006.

[74] A. Goeffon, J.-M. Richer, and J.-K. Hao, "Progressive tree neighborhood applied to the maximum parsimony problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 1, pp. 136–145, 2008.

[75] J. H. Drake, N. Kililis, and E. Özcan, "Generation of vns components with grammatical evolution for vehicle routing," in *European Conference on Genetic Programming*. Springer, 2013, pp. 25–36.

[76] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & operations research*, vol. 34, no. 8, pp. 2403–2435, 2007.