A Study of Genetic Algorithms for Solving the School Timetabling Problem

by

Rushil Raghavjee

Submitted in fulfillment of the academic
requirements for the degree of
Master of Science in the
School of Computer Science,
University of Kwazulu-Natal,
Pietermaritzburg

April 2013

As the candidate's supervisor, I have approved this dissertation for submission

Signed: _____

Name: Prof. Nelishia Pillay

Date: _____

PREFACE

The experimental work described in this dissertation was carried out in the School of Computer Science, University of Natal, Pietermaritzburg, from January 2007 to April 2013, under the supervision of Professor Nelishia Pillay.

These studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of the work of others it is duly acknowledged in the text.

_____

Rushil Raghavjee – Candidate (Student number: 201295456)

_____

Prof. Nelishia Pillay - Supervisor

DECLARATION 1 - PLAGIARISM

I, Rushil Raghavjee (Student number: 201295456) 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 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 re-written but the general information attributed to them has been referenced
   b. Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.

5. 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.

Signed: _____

DECLARATION 2 - PUBLICATIONS

DETAILS OF CONTRIBUTION TO PUBLICATIONS that form part and/or include research presented in this thesis

Publication 1:
R. Raghavjee, N. Pillay. An Application of Genetic Algorithms to the School Timetabling Problem. In C. Cilliers, L. Barnard, R Botha (Eds.) Proceedings of SAICSIT 2008. Eastern Cape, South Africa. Pages 193-199. ACM Press. October 2008. Citations: 11.

Publication 2:
R. Raghavjee, N. Pillay. Evolving Solutions to the School Timetabling Problem. In Proceedings of NABiC 2009. Coimbatore, India. Pages 1524-1527. IEEE Press. December 2009. Citations: 2.

Publication 3:
R. Raghavjee, N. Pillay. An Informed Genetic Algorithm for the High School Timetabling Problem. In Proceedings SAICSIT 2010. Pretoria, South Africa. Pages 408-412. ACM Press. October 2010. Citations: 3.

Publication 4:
R. Raghavjee, N. Pillay. Using Genetic Algorithms to Solve the South African School Timetabling Problem. In Proceedings NABiC 2010. Kitakyushu, Japan. Pages 286-292. IEEE Press. December 2010. Citations: 3.

Publication 5:
R. Raghavjee, N. Pillay. The Effect of Construction Heuristics on the Performance of a Genetic Algorithm for the School Timetabling Problem. In Proceeding of SAICSIT 2011. Cape Town, South Africa. Pages 187-194. ACM Press. October 2011.

Signed:

_____ _____

Rushil Raghavjee Prof. Nelishia Pillay

Abstract

The school timetabling problem is a common optimization problem faced by many primary and secondary schools. Each school has its own set of requirements and constraints that are dependent on various factors such as the number of resources available and rules specified by the department of education for that country. There are two objectives in this study. In previous studies, genetic algorithms have only been used to solve a single type of school timetabling problem. The first objective of this study is to test the effectiveness of a genetic algorithm approach in solving more than one type of school timetabling problem. The second objective is to evaluate a genetic algorithm that uses an indirect representation (IGA) when solving the school timetabling problem. This IGA approach is then compared to the performance of a genetic algorithm that uses a direct representation (DGA). This approach has been covered in other domains such as job shop scheduling but has not been covered for the school timetabling problem.

Both the DGA and IGA were tested on five school timetabling problems. Both the algorithms were initially developed based on findings in the literature. They were then improved iteratively based on their performance when tested on the problems. The processes of the genetic algorithms that were improved were the method of initial population creation, the selection methods and the genetic operators used.

Both the DGA and the IGA were found to produce timetables that were competitive and in some cases superior to that of other methods such as simulated annealing and tabu search. It was found that different processes (i.e. the method of initial population creation, selection methods and genetic operators) were needed for each problem in order to produce the best results. When comparing the performance of the two approaches, the IGA outperformed the DGA for all of the tested school timetabling problems.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1 - Introduction

1.1 Purpose of the study

The school timetabling problem is a common problem faced by many schools. School timetabling problems vary between different schools and different countries in terms of the constraints specified by the educational system of that country. In most schools, timetables are manually designed and teachers usually set aside a week or a weekend for this, thus taking up valuable time in the process. In addition to being a time-consuming process, the manual design of a timetable is subject to human error and may not satisfy all the constraints.

Genetic algorithms attempt to mimic the evolutionary process. They have been previously used successfully to solve combinatorial optimization problems, specifically educational timetabling problems such as examination timetabling and university course timetabling. In previous studies, each genetic algorithm was evaluated on solving a particular type of school timetabling problem with a specific set of constraints i.e. each algorithm was tailored to solve that specific problem. As a result, it is unknown as to how each of these genetic algorithms will fare when applied to more than one type of school timetabling problem, each having a different set of constraints when compared to another [JACO06].

The first objective of this thesis is to study the effectiveness of a genetic algorithm in solving more than one type of school timetabling problem where each type of problem differs in terms of the set of constraints used and the resources of the problem. This genetic algorithm, referred to in this study as DGA, uses a direct representation where each individual represents a timetable.

From the literature, it is clear that a genetic algorithm using an indirect representation has not been studied. This approach has been successfully used in domains such as job shop scheduling and examination timetabling but has not been looked at for the school timetabling problem. The second objective is to develop and evaluate a genetic algorithm that uses an indirect representation (IGA) when solving the school timetabling problem. The IGA will be evaluated by comparing its performance to that of the DGA.

1.2 Objectives

The objectives of this thesis are:

- Based on the analysis of the literature, implement a genetic algorithm approach for solving the school timetabling problem and evaluate it on more than one type of school timetabling problem.
- Develop and evaluate a genetic algorithm approach that uses an indirect representation when solving the school timetabling problem.

1.3 Contributions to the study

This thesis makes the following contributions:

Major contributions

- In investigating genetic algorithms to solve different types of school timetabling problems with different sets of constraints, it was found that different processes (method of initial population creation, selection method and genetic operators used) were needed in order to solve each problem.
- There has been no previous work investigating indirect representations in genetic algorithms when solving the school timetabling problem and it was found that this genetic algorithm (IGA) performed better than a genetic algorithm using a direct representation (DGA).

Minor contributions

- It was clear that the use of a sequential construction method (SCM) produced better results (in terms of both feasibility and timetable quality) when used with the genetic algorithm.
- When solving the school timetabling problems in this study, it was found that the use of a less elitist variant tournament selection was found to be a good alternative to standard tournament selection. In several cases, the genetic algorithm using variant tournament selection performed better than the genetic algorithm that used standard tournament selection (in terms of finding feasible timetables).

1.4 Thesis layout

Following this introduction, Chapter 2 presents an overview of the school timetabling problem. Commonly used terms related to this problem are explained and the most common constraints of the problem are outlined.

Chapter 3 discusses previous work on the school timetabling problem. This chapter describes the school timetabling problems that various authors attempted to solve, the methods that were used to solve the problem and the results obtained.

Chapter 4 firstly covers the standard genetic algorithm described by Goldberg [GOLD89]. This chapter also looks at recent modifications that were made to the processes involved in genetic algorithms in order to solve various problems or improve performance.

Chapter 5 provides a detailed description of previous work that applied genetic algorithms or evolutionary algorithms to solve the school timetabling problem. This includes the processes of the genetic algorithm that were used (initial population generation, selection methods and genetic operators) as well as the results obtained.

Chapter 6 discusses the methodology and experimental setup used to evaluate both the direct representation genetic algorithm approach (DGA) and the indirect representation genetic algorithm approach (IGA).

Chapter 7 presents a genetic algorithm approach that uses a direct representation to solve the selected school timetabling problems. This section describes the representation used, initial population generation, timetable evaluation, the selection methods, and the mutation operators that are considered.

Chapter 8 presents a genetic algorithm approach that uses an indirect representation to solve the selected school timetabling problems. Similar to Chapter 7, this chapter describes the representation used, the creation of the initial population, the fitness function used to evaluate an individual, selection methods used and the genetic operators considered.

Chapter 9 discusses and compares the performance of both approaches (described in chapters 7 and 8) when solving the five school timetabling problems.

Chapter 10 provides a summary of the results in the study and ideas for future research.

Chapter 2 - An overview of the school timetabling problem

This chapter outlines the school timetabling problem. The chapter begins by providing a definition of the school timetabling problem. The chapter then describes the constraints that are commonly faced by schools. These constraints fall into one of two categories, namely hard constraints and soft constraints. Hard constraints are requirements that must be fulfilled. The chapter ends by describing soft constraints, which are preferences that the school would like to implement but are not necessary for a feasible timetable.

2.1 The School timetabling problem

The school timetabling problem is a scheduling problem that involves the allocation of school resources in a particular manner so as to fulfill a set of requirements specified by the school. Common school resources include teachers, venues and classes.

A school timetable consists of a set of periods (timeslots). Each period must be occupied by a group of one or more tuples where each tuple consists of a combination of one or more resources such as classes, teachers, subjects or rooms.

Figure 2.1 shows an example of four tuples consisting of a class, a teacher, a subject and a venue. Table 2.1 shows an incomplete timetable with four tuples allocated to it. A full list of tuples is specified by the school and all tuples must be allocated to the timetable in a manner that will satisfy the requirements set by the school. The feasibility (usability) and quality of a timetable are dependent on whether the timetable satisfies the rules of the school. These rules are referred to as constraints.

TUPLE 1	TUPLE 2	TUPLE 3	TUPLE N
Teacher: SMK Class: 5B Subject: English Venue: Room 23	Teacher: GS Class: 4A Subject: Geography Venue: Room 25	Teacher: SML Class: 5B Subject: Zulu Venue: Room 23	Teacher: ABC Class: 1A Subject: Foundation Venue: Room 1

Figure 2.1: Examples of tuples

Table 2.1: Class timetable example

Day	Period	1A	4A	5B
1	1			Zulu: SML (Room 23)
1	2	Foundation: ABC (Room 1)	Geography: GS (Room 25)	English: SMK (Room 23)
1	3			
1	4			

Any constraint specified by the school will fall into one of two categories. The first category is hard constraints. A hard constraint is a rule that must be met when considering the allocation of tuples to the timetable. When all hard constraints are satisfied, the timetable is said to be feasible. A violation of any of the hard constraints results in a timetable that cannot be used. This is known as an unfeasible timetable.

The second category of constraints, soft constraints, are specifications made by the school, the teacher, or any other resource to improve the quality of the timetable. Violation of soft constraints may still result in a feasible timetable. Soft constraints may also contradict each other, meaning that removing a soft constraint violation may violate another soft constraint. Thus, it may not be possible to satisfy all soft constraints and the aim is to therefore minimize the soft constraint cost of the timetable in order to improve the quality of the timetable.

The ideal timetable is one that has zero hard constraint violations (feasible) and a minimum number of soft constraint violations (high quality). The most common hard and soft constraints are listed in sections 2.2 and 2.3 below.

2.2 Common hard constraints

This section lists the most common hard constraints identified in studies on the school timetabling problem.

2.2.1 Lesson requirements

This constraint specifies that all class-teacher meetings (lessons) specified by the school must be scheduled the required number of times. No class-teacher tuples must be left out of the timetable. This is a common requirement specified by all schools.

2.2.2 Clashes

A clash occurs when a single resource is allocated to two or more periods or other resources at the same time. The three most common clashes are:

- *Class clashes* – When two or more classes are allocated to the same teacher/venue during the same period.
- *Teacher* clashes - When two or more teachers are allocated to the same venue/class during the same period.
- *Venue clashes* – When two or more teachers or classes have been allocated to the same venue.

Table 2.2 shows a *class clash* occurring during period 3 where the teacher SML is required to teach two different classes at the same time. The table also shows a *venue clash* where classes 4A and 5B are scheduled to be taught at the gym at the same time (see period 4). These two situations are regarded as clashes unless the school requirements state that it is acceptable for these two classes to share a venue or teacher at the same time. The clashes constraint is a common requirement of all timetabling problems including Beligiannis et al. [BELI08], Abramson [ABRA91b], Birbas et al. [BIRB97] and Schaerf et al. [SCHA01].

Table 2.2: Clashes during periods 3 (teacher) and 4 (venue)

Day	Period	1A	4A	5B
1	1			Zulu: SML (Room 23)
1	2	Foundation: ABC (Room 1)	Geography: GS (Room 25)	English: SMK (Room 23)
1	3	Zulu: SML (Room 1)	Zulu: SML (Room 25)	
1	4		Physical Education: TM (Gym)	Physical Education: WB (Gym)

2.2.3 Consecutive period requirements

A consecutive period requirement specifies that certain tuples that use the same resources be allocated consecutively. For example, two tuples containing the same class, teacher and venue should be allocated consecutively, where one tuple is placed in period 2 while the next tuple is placed in period 3. Table 2.3 shows an example of this. This requirement often occurs when teachers find that one period is not enough to sufficiently cover the subject matter and require two or more consecutive periods. The placement of two or three consecutive tuples that use the same resources is referred to as double or triple periods respectively. This constraint is considered by Abramson et al. [ABRA91a], Bufe et al. [BUFE01], Di Stefano et al. [DIST01] and Cerdeira-Pena et al. [CERD08]. A consecutive period requirement can also be specified as a soft constraint by a school as proposed by Bello et al. [BELL08].

Table 2.3: Double period allocation during periods 2 and 3 (Class 5B)

Day	Period	1A	4A	5B
1	1			Zulu: SML (Room 23)
1	2	Foundation: ABC (Room 1)	Geography: GS (Room 25)	**English: SMK (Room 23)**
1	3	Zulu: SML (Room 1)	Zulu: HHH (Room 25)	**English: SMK (Room 23)**
1	4		Physical Education: TM (Gym)	Physical Education: WB (Gym)

2.2.4 Co-teaching and subclasses

According to Beligiannis et al. [BELI08], a co-teaching requirement occurs when two or more classes are divided into two or more temporary classes. Each temporary class contains a subset of students from different classes and each temporary class is allocated to a particular teacher and is taught a specific subject. For example, all grade 10 classes in a school divide and form two temporary language classes. One temporary class is taught French while the other is taught Spanish. The teachers are allocated to both classes with no clash being recorded.

According to Beligiannis et al. [BELI08], a subclass requirement occurs when a single class is split into two or more groups and each group is taught a subject by a different teacher. For example, in physical education, a class is divided into males and females with the boys being taught by a male teacher and the girls being taught by a female teacher. In this case, two or more teachers are allocated to the same class and no clash is recorded.

2.2.5 Specialized venues

Certain venues contain specialized equipment for specific subjects. For example, physical education is held on the school grounds or the gym allowing the class and teacher to use sports and gym equipment. Computer lessons or biology lessons should be allocated to venues where there are computers and microscopes respectively. A hard constraint would be for particular lessons to be allocated to specialized venues. Jacobsen et al. [JACO06] address this problem for the German school timetabling problem. Another requirement may also specify that lessons must be taught in venues that are able to accommodate the size of the class. For example, a class of 40 students must only be allocated to venues that are able to hold 40 or more students.

2.2.6 Teacher availability

Beligiannis et al. [BELI08] and Valouxis et al. [VALO03], when dealing with the Greek school timetabling problem, had to address the problem of teacher availability. Certain teachers are only available to teach during certain periods of the day or certain days of the week. This may be due to teaching duties in other schools or various other duties (such as meetings or administration duties) that make them unavailable to teach. For example, a headmaster that teaches may be unavailable to teach in the late afternoons due to administrative duties and/or parent meetings. Thus to allocate the headmaster to the last few periods of the day would be a violation. Another example would be a teacher that specializes in a subject and is required to teach in two different schools. He/she would therefore only be available to teach on certain days for one school and the remaining days for the other. Allocation of this resource to a day when that teacher is unavailable would be a violation. This constraint can also be defined as a soft constraint.

2.3 Common soft constraints

2.3.1 Daily teacher/subject limits

A common requirement is the limitation of the number of lessons a teacher is allocated per day. This could also be applied to subjects where there is a limit on the number of times the subject is taught to a particular class in a day. For example, a class should only be taught English twice in a day. An advantage of this constraint is that the subjects are spread equally throughout the week and a public holiday will have a minimum impact on the number of lessons missed. Setting a daily limit on the number of lessons taught by a particular teacher would prevent the teacher from being overworked on one day and underutilized on another. This constraint is addressed by Beligiannis et al. [BELI08] and Valouxis et al. [VALO03] when solving different Greek school timetabling problems.

2.3.2 Compact timetables

A compact timetable is a timetable where all lessons occur consecutively with no free (idle) periods between lessons. This requirement could be specified for both teachers and/or classes.

For example, the timetable below (Table 2.4) for teacher A would be regarded as a compact timetable as the teacher has no free periods between his/her first and last lesson. The timetable for teacher B has two free periods so this timetable is not as compact as the timetable produced for teacher A. The objective of this constraint would be to minimize the number of free periods for each teacher or class.

Table 2.4: Compact timetable (Teacher A) and a timetables with gaps (Teacher B)

Teacher A		1A:PE	2C:Maths	7A:Geog	7B:Bio				
Teacher B	2A:Eng	3A:Eng		4B:Math		5C:Math			

Compact timetable constraints are addressed by Beligiannis et al. [BELI08], Desef et al. [DESE06], Schaerf et al. [SCHA01] and De Haan et al. [DEHA07] among others. This constraint has been known to be specified as a hard constraint.

2.3.3 Replacement teachers

This requirement differs from the teacher availability constraint in that with the teacher availability constraint, it is known from the beginning of the school term that the teacher will be unavailable on a particular day or at a particular time (due to other commitments such as teaching in another school). With the replacement teacher constraint, a substitute teacher replaces the originally allocated teacher only if he/she cannot attend on a particular day due to unforeseen circumstances. The most common example is that a teacher is ill and that another teacher must replace him/her.

2.3.4 Resource preferences

It may be necessary to allocate a resource to a specific period on the timetable. For example, a requirement may state that mathematics should be taught in the morning sessions (periods before noon). There may be a request for physical education lessons to be placed later in the day due to weather (cold mornings) or field conditions (wet fields). A teacher may also request that their lessons be allocated on specific days, for example from Monday to Thursday, allowing them to perform other duties or activities on a Friday. Some teachers may request that their lessons be allocated to the morning periods, while others may request periods that are in the afternoon [VALO08]. This constraint could also be categorized as a hard constraint in some schools. Jacobsen et al. [JACO06] and Di Stefano et al. [DIST01] address this constraint for the German school timetabling problem.

2.4 The school timetabling problem as a multi-objective problem

For the school timetabling problem, the satisfaction of hard constraints cannot be seen as a multi-objective problem since all these constraints must be met ([BURK08], [SAGI06], and [PAIS09]).

In their study of examination scheduling, Burke et al. [BURK08], Sagir [SAGI06] and Pais et al. [PAIS09] found that fulfilling of soft constraints could be seen as a multi-objective problem. Burke et al. [BURK08] found that grouping constraints according to the

stakeholders involved (i.e. the students and the administrators) reduced the complexity of the problem. In the three studies, the soft constraints were considered as sub-objectives and the number of violations for each of the sub-objectives had to be minimized. A similar approach can be applied for the school timetabling problem when dealing with timetable quality (soft constraints). In this case, each soft constraint will be regarded as a sub-objective.

2.5 Chapter summary

This chapter covers the main aspects of the school timetabling problem. The chapter begins by providing a definition of the problem and describes the common constraints that must be fulfilled in order to make a timetable feasible and of a high quality. Schools usually define these constraints based on the needs of the school, the resources available and the rules specified by the department of education for that country.

Chapter 3 - Previous work on the school timetabling problem

This chapter describes previous research conducted to solve the school timetabling problem. Some of the techniques reported below include tabu search, simulated annealing, constraint programming, neural networks and particle swarm optimization. Hybrid techniques incorporating multiple methods or approaches are also covered here. Research regarding genetic or evolutionary algorithms is discussed in Chapter 5.

3.1 The tabu search

3.1.1 Tabu search description

The tabu search is a metaheuristic originally introduced by Glover [GLOV89]. Starting from a candidate solution (also known as a potential solution), the tabu search is a local search algorithm that moves from one candidate solution to another candidate solution (referred to as a neighbour) until some problem dependent termination criteria has been met. Moving from a candidate solution to a neighbour is accomplished using a move operator, where a single change is made to the candidate solution resulting in its neighbour. A neighbourhood is defined as a set of neighbours that occur as a result of implementing a single move to a candidate solution.

In order to prevent the search from cycling (returning to a previously encountered area of the search space), a tabu list is kept. This tabu list stores a set of *k* recent candidate solutions. Alternatively, previous moves that have been applied to candidate solutions can also be stored in the tabu list [SCHA96]. A move is rejected if it results in a candidate solution that is in the tabu list. Another move must then be made.

As described by Glover [GLOV97], an advantage of the tabu search is its use of the tabu list. This list resembles a form of memory, preventing the search algorithm from returning to previous candidate solutions. A disadvantage of the tabu search is that the focus is always on a single candidate solution. Thus, the possibility arises that a large area of the search space is not covered [LUKE12], [ZDAN02].

3.1.2 Applications of tabu search to the school timetabling problem

Schaerf [SCHA96] presented a tabu search algorithm to solve an Italian school timetabling problem. The problem has the following hard and soft constraints with each constraint being given a specific weighting:

- Timetables must be compact.
- Lessons of a particular subject must not be repeated in a day.
- Daily teaching limits must be met.
- Requirements involving specialized rooms must be met.
- Teacher availability requirements must be met.
- Movement from one venue to another between consecutive periods should be minimized.

A timetable was represented using a matrix structure (two-dimensional array). In this representation, the rows represented the periods of the week and each column represented a teacher. The cells contained the class that will be taught by the teacher during that period. The author did not make mention of how a venue is represented. An initial solution was randomly generated. Neighbours were formed by applying one of two types of moves. The first type was a move that randomly chooses a tuple and moves the tuple from one part of the timetable to another. The second type of move, a double move, was composed of a single move and an optional second move if the first move resulted in a constraint violation. A tabu list was kept which stored a predefined number of unacceptable moves. The tabu search terminated after a set number of iterations or when a feasible solution was found. The tabu search was applied to three Italian school timetabling problems, with each school differing in the number of teachers and lessons. The tabu search produced feasible timetables for all three of the schools. The timetable structure adopted was appropriate as the matrix structure best represents a timetable (discussed further in section 5.4.1). Besides using only single moves to swap tuples in the timetable, the authors also introduced the concept of double moves in order to repair any constraint violations caused by the first move. Hill climbing, where moves resulting in constraint violations are rejected, could have been used instead. The tabu search employed, however, did produce acceptable results for all three schools as feasible timetables with only a few soft constraint violations were generated.

Desef et al. [DESE06] used a tabu search to solve the German school timetabling problem that was subject to the following hard constraints:

- No class clashes, teacher clashes and venue clashes.
- Teacher and room availability constraints must be met.
- Co-teaching and subclass requirements must be met.
- Double period requirements must be met.

The soft constraints for the problem are the following:

- Lessons of a particular subject should be evenly distributed over the week for each class.
- There should be no free periods for the first four periods of the day.

In this paper, a tuple was defined as a complex lesson unit (CLU). In the event of a double period requirement, the two tuples involved in the double period were combined into a single CLU with a double period attribute, thus fulfilling the double period requirement. Furthermore, a set of CLUs were combined to form the lessons of the week (i.e. the class timetable) for a particular class. Each timetable was represented as a set of period vectors. The period vectors contain the list of CLUs for each class for the week. An initial solution was created where the CLUs with the fewest number of feasible periods on the timetable were allocated first. The selected CLU was then allocated to the earliest possible period that resulted in a minimized hard constraint cost. The tabu search was applied in two ways. The first application allowed the tabu search to move from one neighbour to another using single moves (making a single change to the timetable). The second application of the tabu search only allowed moves that did not violate constraints. Two tabu lists were kept. The first list was a tabu list that stored previous moves (similar to that described in section 3.1.1) and the second was a frequency list that kept track of how often CLUs moved around the timetable. A move was rejected if it was in the first tabu list or if the frequency of the related CLU exceeded a particular frequency value. The timetables produced using this tabu search was better or equivalent in terms of quality when compared to timetables produced manually by the school staff. Creating an initial solution by first allocating CLUs with the fewest feasible periods was an important factor that contributed towards the good performance of the tabu search as the number of violations would have been fewer than if a random allocation method had been used.

3.2 Integer or linear programming

3.2.1 Integer or linear programming description

According to Wolsey [WOLS98], integer programming is a method that attempts to solve optimization problems using discrete or integer variables. Each variable represents a constraint of a problem. Binary variables can also be used to symbolize decision events such as when to buy or sell or to represent whether a switch is on or off. In terms of optimization problems, the objective of the integer programming method is to minimize the cost of a function while ensuring that various constraints related to the problem are met. The integer programming method determines the best possible combination of resources to allocate. A disadvantage of integer programming is that the method can only be applied to situations where mathematical equations are formulated [WOLS98]. Thus the method cannot be applied to many real world problems. Another problem with integer programming is that the element of risk or uncertainty cannot be considered in the formula.

3.2.2 Application of integer programming to the school timetabling problem

Birbas et al. [BIRB97] constructed Greek school timetables using an integer programming approach. The hard constraints for the problem are:

- There must be no clashes.
- Each teacher should teach at least one lesson per day.

The soft constraints for the problem are:

- Even distribution of subjects taught per week.
- Certain subjects should be taught at the beginning of the day.
- Certain subjects should not be taught during the last period of a day.
- All class timetables should be compact.
- Certain periods should be left empty for teachers to conduct other duties.

Each period was represented using a variable x_{abcde} where a, b, c, d and e represent the day, period, class, teacher and subject respectively. Each variable was binary in nature and was allocated a value of 1 if it is included in the final solution i.e. a value of 1 is given to the variable if it has been allocated a subject, class and teacher to a specific day or period. The objective of the model was to find a violation free timetable from all the possible combinations. The integer programming method was applied to a Greek school timetabling problem with six classes and 17 subjects. The timetables created using this method were

described as optimal. The approach attempts to find a feasible, high quality timetable by determining the best possible combination of tuples such that the timetable cost was minimized. While this technique does allow for a thorough exploration of all possible values, no time element was included in the results and it is not known how long the system would have taken to find the best timetables.

3.3 Tiling algorithms

Kingston [KING05] proposed a tiling algorithm to solve the Australian school timetabling problem. The hard constraints for the problem are:

- No class clashes or teacher clashes.
- There must be an even distribution of lessons taught by teachers during the week.
- Teacher availability requirements must be met.
- Certain subjects must be allocated to specific venues.

The soft constraints are:

- There should be an even distribution of lessons for classes throughout the week.
- Co-teaching and subclass requirements should be met.
- The number of daily lessons allocated to each teacher should be within a specified range.

A tile is a group of lessons placed together. Each tile has a height and a width. The height specifies the number of resources required while the width indicates the number of times that the lessons have been requested. To create a timetable, all tuples were placed onto tiles. These tiles were then placed in major columns and each major column was given an index number. The column indices were then allocated to the timetable where each column was spread evenly throughout the week. These tiles were then allocated to the timetable. By adopting this approach, several subjects were taught as double or triple periods. An unspecified tree search was used to allocate tuples to columns such that no lessons are split. Each lesson was classified as vertical, runaround or easy. A vertical classification indicated that the lessons must run simultaneously. A runaround classification indicated that each meeting should be placed in different columns and an easy classification symbolized either a vertical or runaround strategy could be used. Tiles were allocated to the timetable using three methods. The first method involved a worst case scenario where tiles with the least number of resources available were allocated. The second method involved adding

weightings to tiles. Tiles that had a greater resource usage took priority in terms of allocation. The third method allocated tiles that involved co-teaching and subclass combinations. High quality timetables were obtained in an acceptable time. The study singled out tiling algorithms and resource allocation as the key innovations.

3.4 Simulated Annealing

3.4.1 Introduction to simulated annealing

Simulated annealing was originally established by Kirkpatrick et al. [KIRK83]. This metaheuristic is inspired from the process of annealing in metallurgy. Annealing describes the way in which the heating and cooling of molten metal takes place in order to remove defects. As metal is cooled, its atoms freeze into the positions that they are in, giving the metal its brittle shape that a person may be aiming for. Heating of the metal allows the atoms to unfreeze and move more freely. With simulated annealing, the atoms represent elements of a given problem. For example, in the school timetabling problem, atoms represent tuples on the timetable. These atoms are cooled into position as the algorithm pushes the candidate solution closer to a problem specific ideal state. In the event that the candidate solution moves further away from the ideal state, the elements are heated allowing them to move through the solution. In simulated annealing, a candidate solution is changed using the move concept described earlier in the tabu search. A single move results in the forming of another candidate solution (called a neighbour) and a group of neighbours is referred to as a neighbourhood. Simulated annealing is seen as a stochastic search and optimization technique [KIRK83] and thus the moves made are usually random in nature.

One of the advantages of simulated annealing, according to Elmohamed et al. [ELMO98], is the relative ease of implementation. The simulated annealing technique also has the ability to approach a global optimum [BUSE03]. However there is often a trade-off between attaining an optimal solution and the time taken to find a solution. A common disadvantage found with simulated annealing is the difficulty in defining an optimal cooling schedule [COEL07]. This is an important factor that could prevent premature convergence.

3.4.2 Applications of simulated annealing to the school timetabling problem

Abramson [ABRA91b] developed a system to solve the Australian school timetabling problem. The list below describes the hard and soft constraints of the problem.

- No class clashes, teacher clashes and venue clashes.

- Classes must be allocated to rooms that can accommodate them.
- Double period requirements must be met.

The soft constraints for the problem are:

- Teacher preferences should be met.
- Certain lessons should only be taught a maximum number of times per day.

Abramson used the simulated annealing algorithm in which the atoms represented tuples on a timetable. The paper does not make mention of how a timetable is represented, but a three-dimensional array representation is illustrated with each dimension representing a period, a day of the week and an array of tuples (indicating the tuples that will be scheduled during a particular period on a certain day). The cooling and heating process was dependent on the timetable cost, which was determined by counting the number of constraint violations. As the cost was lowered, the temperature was lowered and an increasing number of tuples were fixed ("frozen") to the periods that they had been scheduled in. If the cost increased, the tuples were "heated", resulting in an increase in the number of tuples that could have been swapped. The system was applied to generated data sets as well as an Australian high school timetabling problem. Ten data sets were created and solutions were found for all but one data set. The results were described as promising and future work would look at improving this method of solving optimization problems. The cube representation, while appropriate, could have been difficult to implement due to the structure having three dimensions.

Liu et al. [LIU09] developed a system to solve the Greek school timetabling problem. The hard constraints of the problem are:

- No clashes.
- Teacher availability requirements must be met.
- All class timetables must be compact.
- All free periods must be during the last period of the day.

The soft constraints for the problem are:

- Compact timetables for teachers.
- A daily teacher balance in terms of the number of lessons taught.

- Avoidance of repetition of subjects during a day.
- Teacher preferences should be met.

A timetable was represented using a two-dimensional matrix where the rows represent the class and the columns represent the periods of the week. A cell in the matrix stored the teacher that would be meeting that particular class during the specified period. Changes to the timetable were made by swapping of tuples. The authors defined a neighbourhood as all the candidate solutions that could be reached by performing a set of one or more swaps on a candidate solution. The swapping process involved searching for and swapping of constraint violating tuples. The algorithm was applied to the HDTT data sets provided by Smith et al. [SMIT03] and a Greek school timetabling problem. Based on the results presented, it was found that the simulated annealing approach managed to produce high quality timetables when compared to the simulated annealing and neural network approaches developed by Smith et al. [SMIT03]. The authors stated that the idea of combining their neighbourhood strategy with other metaheuristics required further research. The use of multiple swaps rather than only single swaps was instrumental in assisting the simulated annealing technique in finding high quality, feasible timetables. The concept of using moves that focused on constraint violations (rather than just swapping of randomly chosen tuples) also contributed to the success of the approach.

3.5 Constraint programming

3.5.1 Constraint programming description

Bartak [BART99] describes constraint programming as the development of computational systems that are based on constraints. The main objective of the constraint programming technique is to solve a problem by identifying the constraints within the problem area and then finding a solution that satisfies all these constraints. A set of variables are defined where each variable represents an element of the problem (for example, variables could represent the tuples or lessons in the school timetabling problem). Each variable holds a single value which must be chosen from a specified range of values. These values could vary and be anything from integers, strings or a range of situations depending on the type of problem [BART99]. The constraints restrict the values that each variable can hold at one particular time. The two most common branches of constraint programming are *constraint satisfaction* and *constraint solving*. *Constraint satisfaction* is the more common approach and deals with problems involving a finite domain such as resource allocation while *constraint solving* deals with problems involving an infinite or more complex domain such as production planning.

Some applications of constraint programming include resource allocation, scheduling problems, software configuration and production planning.

3.5.2 Applications of constraint programming to the school timetabling problem

Valouxis et al. [VALO03] used a constraint programming method to solve the Greek school timetabling problem. The hard constraints for the problem are as follows:

- No class clashes and teacher clashes.
- Class timetables must be compact. Any free periods must be at the end of the day.
- Teacher allocations should be equally balanced throughout the week.
- The number of times a teacher meets a class in a day to teach a particular subject should be balanced throughout the week.

The soft constraints are as follows:

- Teacher preferences should be met.
- Teacher timetables should be compact.

The system was applied to the school timetabling problem faced by Greek high schools. The problems consist of varying sizes of classes (ranging from five to nine) and teachers (ranging from 11 to 23). The system found optimal timetables for two of the four problems. These two solutions were found in approximately 15 to 20 minutes. For the two larger data sets, the author concluded that the timetables produced, while not optimal, were satisfactory. The time taken by the constraint satisfaction approach to find solutions to these problems was approximately one hour. Similar to integer programming, the constraint programming technique needs to evaluate several combinations of variable values and determine whether or not these values violate the constraints specified. Due to the large number of permutations that could exist, finding the ideal solution may be a time consuming process.

The school timetabling problem that Marte [MART07] attempted to solve had the following hard constraints. The problem contained no soft constraints.

- No clashes.
- Teacher availability requirements must be met.
- Co-teaching and subclass requirements must be met.

- Teacher lessons must be evenly distributed throughout the week.
- The number of days that a teacher works for must be within a specified range.
- The number of days that a class has lessons must be within a specified range.

A timetable was represented as a matrix consisting of rows (days) and columns (periods). The approach used included the following techniques:

- *Backtracking*, where a candidate solution is built and once this candidate solution can no longer be improved (referred to as a dead end), the process returns to a previous candidate solution and continues to make changes to the candidate solution until some termination criteria has been met [ROSS06].
- *Constraint propagation*, which prevents tuples or combinations of tuples from being allocated to specific periods since they would violate a constraint [ROSS06].
- *Dead-end driven learning* and *restarting* strategies, where the system keeps track of moves that could no longer improve the timetable by storing this information in memory. The algorithm learns from this by searching a different area of the search space [FROS94].

Information regarding constraints that were not fulfilled was stored and the search strategy was then changed accordingly. The approach was applied to six German school timetabling problems. Good results were found and further research would address other constraints and improve the quality of the timetables produced. The use of a matrix representation was justified as this structure bests represents an actual school timetable. Good results were found for large problems in less than a minute, indicating that the constraint programming approach is more than capable of solving the German school timetabling problem.

3.6 Particle Swarm Optimization

3.6.1 Introduction to particle swam optimization

Particle swarm optimization, introduced by Kennedy et al. [KENN95], is a technique that was originally used to simulate social behaviour. Particle swarm optimization begins by developing and evaluating an initial population of candidate solutions (known as particles) and attempts to find an optimal solution by evolving these candidate solutions over a predefined number of iterations.

During every iteration, the particles move from one position in the search space to another. The movements of these particles are guided by two factors; the position of the best particle

in the search space *(gbest)* and each candidate's own best position *(pbest)*. The velocity of each particle is calculated in order to determine the new location of that particle. The calculation of the velocity involves the use of g*best, pbest*, the particles' current position and a random value. The above process continues until either an optimal solution is found or until some problem specific termination criteria stops the iteration process.

Particle swarm optimization has been applied to many problems including optimization problems, security solutions and in the medical field amongst others. Advantages of the approach include ease of implementation and that a minimal number of parameters require tuning.

3.6.2 Application of particle swarm optimization to the school timetabling problem

Beligiannis et al. [BELI12] attempted to solve the Greek school timetabling problem. This problem had the following set of hard constraints:

- Teacher availability requirements must be met.
- No class clashes or teacher clashes.
- All free periods for classes must be allocated to the last period of the day.
- Co-teaching and subclass requirements must be met.

The soft constraints for the problem are:

- The number of teaching periods for each teacher should be evenly distributed over the days that he/she is available at the school.
- Teacher timetables should be compact.
- The number of free periods should be uniformly distributed amongst all teachers while free periods for each teacher should be uniformly distributed amongst all days that he/she is available at the school.
- Subjects taught to each class should be evenly distributed throughout the week

A timetable was represented using a two-dimensional matrix where each column represented a period while each row represented a class. Each cell in the matrix represented the teacher (or teachers) that would engage with a class during a particular period. The subject being taught was not represented as this was found to increase both the search space and the complexity of the problem. Teachers were assumed to know the rules of the school and the subjects that they were required to teach. The cost of the timetable

was calculated by finding the weighted sum of the constraint violations. For the PSO algorithm, 150 particles were used and were evolved over 8000 generations (iterations). Particles that were considered weak were deactivated but no explanation was given as to how this was done. Timetables were evolved through the swapping of tuples as well as a column move approach. All swaps that reduced the timetable cost were accepted and swaps that increased the cost were accepted with a given probability. In the column move approach, the tuples from a randomly chosen period in a timetable are copied and moved to the corresponding period of another randomly chosen timetable. The tuples that were originally in that period are reallocated to other periods in the timetable. The performance of the PSO metaheuristic was compared to the performance of a genetic algorithm, a constraint programming approach, a column generation approach and against the actual timetable used by the school. The PSO algorithm was found to be very efficient and produced better quality timetables than the other techniques. The removal of subjects as part of the problem was effective in reducing the complexity of the problem. The column move approach was a novel approach not used in other studies and contributed towards maintaining diversity among the population of particles by moving columns of tuples and reallocating replaced tuples.

3.7 Hybrid algorithms

This section describes papers that have solved various school timetabling problems by using combinations of different techniques.

Avella et al. [AVEL07] addressed an Italian high school timetabling problem by using a combination of local search algorithms, namely simulated annealing and a VLSN (Very Large Scale Neighbourhood) search. The hard constraints for this problem are as follows:

- No clashes.
- Certain lessons must be allocated to specific periods.
- Class timetables must be compact.
- Some teachers have one day off a week (full-time) while others have more than one day free (part time).

In addition to hard constraints, the soft constraints are:

- Teacher preferences should be met.
- Teacher timetables should be compact.

- Teacher lessons should be evenly distributed throughout the week.
- No teacher should be allocated more than one period of teaching during afternoon sessions.

A two phased approach was used. The first phase used simulated annealing to find feasible timetables and the second phase used a VLSN search (integer programming) to improve the quality of the feasible timetables found. In the first phase, timetables were created by randomly allocating tuples to periods. As the cost of the timetable was reduced, the tuples placed in the violation free periods were fixed to their allocated periods. As the cost increased, the tuples were "heated", allowing these tuples to be included in the swapping process if required. The system was firstly applied to the benchmark data sets provided by Abramson [ABRA93]. Feasible timetables were produced for all data sets and these results were compared to the results obtained by Smith et al. [SMIT03]. The simulated annealing part of the system performed better (in reducing the hard constraint cost) than the neural network and simulated annealing methods presented by Smith et al. [SMIT03]. The authors also applied their approach to two Italian high school timetabling problems. Their results were compared to the timetables generated using a commercial software package. Feasible solutions were found for all problems and the VLSN search was found to reduce the number of soft constraint violations. The VLSN search managed to reduce the number of soft constraint violations by approximately 20% when compared to the commercial software alternative. This VLSN search also managed to reduce the soft constraint cost of four randomly generated timetables. The authors concluded that simulated annealing found feasible solutions and the VLSN search improved the quality of the timetables.

A three phase approach was adopted by Alvarez-Valdez et al. [ALVA96] to solve the Spanish school timetabling problem. This approach used two separate tabu search methods to respectively address the hard constraints and the soft constraints. The hard constraints for this problem are the following:

- No clashes.
- Teacher availability requirements must be met.
- Each subject is taught to a particular class at most once in a day.
- Double period requirements must be met.

The only soft constraint is that all class timetables should be compact. The first phase involved creating a candidate solution. Tuples were allocated in order of urgency i.e. tuples

with the fewest violation free periods were allocated first. In the event of two or more tuples having the same urgency, then teacher urgency is used as a tie-breaker i.e. teachers with the fewest number of violation-free periods are given priority. A tabu search was applied in phase two and was able to find feasible solutions very quickly as the initial solution did not contain many constraint violations. The tabu list had a variable length instead of a fixed length as the tabu search with a variable length list produced better results. Phase three involved resolving the only soft constraint of compactness. A tabu search was once again used and the results produced were described as good. The authors concluded that they had developed a program to obtain good solutions that satisfied all of the hard constraints. The quality of the timetables was also found to be better than the solutions that were produced manually. The multiphase approach was found to be successful in finding feasible, high quality timetables. The first phase of creating timetables by allocating tuples with the fewest feasible periods first contributed a great deal towards finding a feasible solution. The tabu search could then remove the remaining hard constraint violations. A separate tabu search used to reduce the soft constraint cost also proved to be successful. Another contributing factor was the changing of the length of the tabu list whenever a predefined number of moves were performed. This was found to have improved the results since a changing list size would reduce the probability of cycling when increased and increased the exploration of the search space when decreased.

De Haan et al. [DEHA07] solved the school timetabling problem using a combination of a graph colouring problem and a tabu search. The hard constraints of the problem are:

- Each lesson of a particular subject must be taught on different days.
- Double period requirements must be met.
- Teacher availability requirements must be met.
- Timetables for lower grade classes must be compact.

The only soft constraint was that teacher timetables and higher grade class timetables should be compact. A four phase approach was used. The first phase dealt with optional subjects taken by classes in the upper grades. A branch and bound algorithm was used to place students into groups such that each group contained a set of students doing the same optional subjects. The second and third phases involved the construction of a feasible timetable. In the second phase, the tuples involving the upper grades were allocated in order of tuples with the fewest feasible periods on the timetable. In phase three, a graph colouring heuristic was used to allocate the remaining tuples (involving the lower grades). In

a graph colouring problem, the vertices represent the lessons and these vertices are coloured according to the period to which they have been allocated. When an edge joins two nodes of the same colour, a clash occurs. The fourth phase was used to improve timetable quality (including allocation of rooms and resources) by using a tabu search. The authors stated that not all constraints were incorporated into their system. An empirical comparison was performed with the actual timetable used by the school and it was found that there was a significant reduction in the number of free periods for teachers (reduced from 128 to 48).

Bello et al. [BELL08] used a combination of both a graph colouring algorithm and a tabu search to solve a school timetabling problem that is subject to the following hard constraints:

- No class clashes and teacher clashes.
- Teacher availability requirements must be met.
- Each class must have a maximum of two lessons with the same teacher per day.

The soft constraints for the problem are:

- Teachers should be allocated to teach in the least number of days possible.
- Double period requirements should be met.
- Teacher timetables should be compact.

A timetable was represented using a two-dimensional matrix. Initial solutions were created using a greedy algorithm. No details were provided as to how the greedy algorithm chooses and allocates tuples. The timetable fitness in terms of both feasibility and quality was determined by finding the weighted sum of all the constraint violations. The authors used a graph colouring algorithm to find a feasible timetable. In this algorithm, a vertex of the graph represents a lesson. Two nodes of the same colour that are joined by an edge represent a clash. The graph was coloured using a tabu search method. The system was applied to three Brazilian school timetabling problems as well as two artificial school timetabling problems. The authors found that this hybrid approach produced competitive results when compared to two other tabu search approaches from two unpublished studies.

Schaerf et al. [SCHA01] solved the school timetabling problem by alternating between two different local search techniques, namely hill climbing and a tabu search. These two

techniques are alternated until a solution can no longer be improved. The hard constraints for the problem are:

- No clashes.
- Consecutive period requirements must be met.

The soft constraints are:

- Class timetables should be compact.
- If lessons are not scheduled as doubles or quadruples and they are repeated on a day, then these lessons should be separated.
- Class-teacher lessons should be evenly distributed throughout the week.
- Teacher preferences should be met.
- The same lesson should not be taught to a class more than once in a day.
- Movement between venues should be minimized.

A candidate solution was created by randomly allocating tuples to the timetable. Hill climbing was then applied where moves were accepted only when the fitness of the candidate solution had improved or had not changed. The search terminated when the solution could no longer be improved or when a fixed number of iterations had been performed. The second phase was the application of the tabu search, which continuously made moves until the timetable could no longer be improved. A variable length tabu list was used that decreased when several improvements were made and increased when moves resulted in an increase in timetable cost. The alternation of the tabu search and hill climbing continued for a given number of iterations. If a local optimum is reached, a shifting penalty strategy was employed. This strategy involved changing the weightings of each constraint in the cost function, allowing the tabu search to continuously explore a new area of the search space. While no results were formally provided, the authors found that the alternating method worked well for the school timetabling problem.

3.8 Comparative studies

This section describes research that compared different methods used to solve the school timetabling problem.

A Hopfield neural network [ROJA96] was developed by Smith et al. [SMIT03] to solve the school timetabling problem. This technique was compared to many others techniques used

for scheduling, namely tabu search, greedy search and simulated annealing. These approaches were tested using the benchmark data sets provided by Abramson et al. [ABRA93]. The only constraint for this problem is that there must be no teacher clashes, class clashes or venue clashes. The results showed that the neural networks could perform just as well as the simulated annealing technique. When comparing the neural network to the greedy search, it was found that the neural network produced more feasible timetables, while the greedy search was able to find feasible timetables in a much faster time.

Jacobsen et al. [JACO06] compared a hybrid tabu search and the constraint programming approach in solving the school timetabling problem. The following are the hard constraints for the problem:

- No clashes.
- Teacher availability requirements must be met.
- Double period requirements must be met.
- Co-teaching requirements must be met.
- Certain lessons must be allocated to specific periods.

The following are the soft constraints for the problem:

- Class timetables should be compact.
- Any free periods should be scheduled at the end of the day.

A solution was created using two vectors, a room vector and a period vector. Periods were allocated to double period tuples and co-teaching tuples using the period vector while venues were allocated to tuples using the room vector. An initial solution was created by firstly sorting lesson requirements in order of difficulty (tuples that have the least number of feasible periods were allocated first) and allocating them to periods using a graph colouring algorithm. The tabu search was then applied in order to reduce the hard constraint cost and soft constraint cost of the timetable. In this tabu search, two types of neighbourhoods were explored, namely a period neighbourhood and a room neighbourhood. This tabu search incorporated hill climbing, meaning that a new candidate solution was only accepted if the constraint cost was reduced. If the hard constraint cost was the same, then the solution was kept only if the quality improved. Two tabu lists were kept, namely a standard tabu list (as described in section 3.1) and a frequency tabu list that was used to avoid the movement of tuples that made little difference to the number of violations of a timetable. A comparison of

the hybrid approach and constraint programming was performed and no difference was found in terms of feasibility or timetable quality.

3.9 Chapter Summary

This chapter described several techniques previously used to solve the school timetabling problem. Some studies used a single approach to solve the school timetabling problem while a few used hybrid approaches that incorporated two or more techniques. Each problem differed with respect to the constraints that had to be fulfilled as well as the number of resources that needed to be allocated. In addition, other observations are listed below.

- Most studies made use of a two-dimensional array in order to represent a timetable. The advantages of this approach are discussed in further detail in section 5.4.1.
- Timetables were created either randomly or using heuristics. In this case, heuristics refer to rules that must be followed when choosing the order of tuple allocation or when choosing which period to allocate the tuple to. The use of heuristics, especially allocating tuples based on the number of feasible periods available, proved to be effective and any remaining violations were easily removed using the approach implemented.
- In order to reduce the cost of the timetable, a tuple swapping strategy was commonly used. Most swaps involved randomly choosing tuples with some studies incorporating hill climbing i.e. only accepting swaps that reduce the constraint cost of the timetable. One approach that also assisted in finding good solutions is to conduct swaps by searching for violation causing tuples. This increased the probability of at least removing a constraint violation.
- Another observation was that while some studies attempted to reduce hard and soft constraints simultaneously, other studies such as Alvarez-Valdez et al. [ALVA96] chose a phased approach where attempts were made to first find feasible timetables and then reduce the soft constraint cost. Avella et al. [AVEL07] chose to use two different techniques to respectively reduce each of the hard constraint cost and soft constraint cost of the timetable.

Chapter 4 - Introduction to Genetic Algorithms

4.1 Introduction

Genetic algorithms fall under the category of evolutionary computing, a rapidly growing area of artificial intelligence where problems are solved based on theories of biological evolution such as natural selection and genetic inheritance [OBIT98]. Holland [HOLL92] describes a genetic algorithm as an algorithm that can emulate the evolutionary process. Research shows that genetic algorithms are able to solve complex problems that humans find difficult to interpret and solve. Genetic algorithms are often used to solve combinatorial optimization problems. An advantage of genetic algorithms, when applied to problems such as the school timetabling problem, is that while many other algorithms produce only one solution at a particular time, a genetic algorithm has the capability of producing more than one solution by using mechanisms such as niching [MAHF95]. This allows for the possibility of providing the user with a choice of solutions [ABRA91a].

Section 4.2 describes the biological inspiration for genetic algorithms. Section 4.3 provides an overview of the algorithmic structure of the standard genetic algorithm used by Goldberg [GOLD89]. Section 4.4 describes some of the advances made in genetic algorithms to solve various problems and section 4.5 summarizes key points discussed in this chapter.

4.2 Biological background

All living organisms consist of cells. Each cell consists of a common set of chromosomes, which are strings of DNA. DNA is the hereditary material found in all organisms. Parts of the DNA may vary from one organism to another. These variations produce different characteristics such as different eye colour, skin tone, personality, etc. During reproduction, new organisms (called offspring) are produced. Often, the DNA of an offspring is different from that of the parent(s). The DNA contributes towards the organism's probability of survival in its environment. The fitness of an organism is a measure of its chances of survival in that environment [GOLD89].

4.3 Genetic Algorithm Overview

According to Goldberg [GOLD89], a genetic algorithm begins by creating a population of individuals. This population is referred to as the initial population. Each individual in the

population is then evaluated using a fitness function. A selection process is then applied where fitter individuals from the population become parents. Fitness proportionate selection is used to select two individuals as parents. Two offspring are produced by applying a crossover genetic operator (with probability p_c) to the selected parents. A mutation genetic operator (with probability p_m) is then applied to each of the offspring. The resultant offspring are added to a new population and are evaluated using the fitness function. This process of evaluation, selection and creation of a new generation of offspring continues until some termination criteria have been met. The termination criteria are problem dependent and may involve reaching a generation limit or if an individual in the population fulfils all the criteria specified by the problem. The genetic algorithm presented by Goldberg [GOLD89] is shown in Figure 4.1 below.

```
Gen = 0
Create and evaluate initial population
Repeat
        J = 1
        Repeat
                Select two individuals from population (A and B)
                If crossover occurs with probability pc
                        Offspring C and D = Application of crossover operator to A and B
                Else
                        Offspring C and D = Copy of A and B
                If mutation occurs with probability pm
                        Apply mutation to offspring C and D
                End If
                Evaluate C and D
                Add C and D to new population
                J = J + 2
        Until J > Population Size
        Old Population replaced by New Population
Until Termination criteria have been met
```

Figure 4.1: Genetic algorithm [GOLD89]

Sections 4.3.1 to 4.3.5 discuss the genetic algorithm process in greater detail.

4.3.1 Initial population generation

The population size *N* is a parameter value that specifies the population size. In the genetic algorithm presented by Goldberg [GOLD89], all individuals in the initial population are created randomly. Each individual is represented as a binary string with each bit in the string being a 1 or a 0.

4.3.2 Fitness function

A fitness function is an objective function that is used to assess each individual in the population [GOLD89]. A fitness measure is calculated using the fitness function and this

fitness measure indicates how close an individual is to satisfying the problem specific criteria. The fitness function plays an important role in guiding the evolution of the population.

4.3.3 Selection method

In the standard genetic algorithm [GOLD89], fitness proportionate selection is used to choose parents. The first step is to find the sum of all the fitness values of each individual in the population. Each individual has a probability (p^i) of being selected based on that individual's fitness. In order to calculate the probability of each individual (p^i), each individual's fitness (f^i) is divided by the total fitness of all the individuals. The equation is shown below.

$$p^i = \frac{f^i}{\sum_j f^j}$$

The number of times each individual will be placed in a mating pool is then determined by multiplying the probability p^i by the number of individuals in the population (n).

Table 4.1 shows a population of five individuals and their fitness values. The sum of the fitness values of all the individuals adds up to 200. The fourth column shows each individual's probability of selection followed by the number of times they will appear in the mating pool (fifth column). The example shown in Table 4.1 assumes that a higher fitness value indicates a fitter (better) individual.

Table 4.1: Fitness values of individuals and total fitness

ID no	Individual	Fitness value (f^i)	% of total (p^i)	Number of occurrences in mating pool (p^i * n)
1	101110001	45	45 / 200 = 0.225	0.225 * 5 = 1.125 ≈ 1
2	111000111	10	10 / 200 = 0.050	0.050 * 5 = 0.250 ≈ 0
3	101010101	60	60 / 200 = 0.300	0.300 * 5 = 1.500 ≈ 2
4	000000001	70	70 / 200 = 0.350	0.350 * 5 = 1.750 ≈ 2
5	100010001	15	15 / 200 = 0.075	0.075 * 5 = 0.375 ≈ 0
TOTALS		**200**	**100%**	

Based on the calculations performed in Table 4.1, the individuals in the mating pool are individual 1 (once), individual 3 (twice) and individual 4 (twice). Parents are then selected by

randomly choosing individuals from the mating pool. In the example above, individual 2 and individual 5, the least fit individuals, will not be selected as parents.

A common issue found when using fitness proportionate selection is that the fittest individual can be selected as a parent more often than other individuals due to a high selection pressure (all individuals in the population are included in the selection method). This reduces genetic diversity; especially if the fitness variance is high (a low fitness variance indicates that the fitness values of the individuals in the population are very close while a high fitness variance indicates that there are large differences between the fitness values of the individuals). As a result the genetic algorithm may converge prematurely (offspring produced can no longer be better than the parents) [BLIC95].

4.3.4 Genetic operators

This section describes the mutation and crossover genetic operators.

4.3.4.1 Mutation

The purpose of the mutation operator is to maintain genetic diversity and prevent premature convergence by making random changes to an individual. Figure 4.2 shows an example of the application of the standard mutation operator described by Goldberg [GOLD89] (also known as bit-flip mutation). The first and fifth positions have been randomly selected and their values changed. In the example below, the assumption is made that each bit is independent of each other. In the event that there are bits that are dependent on other bits, then the implementation of the mutation operator will need to be modified.

Parent X **(Mutation points)**	**1**1100**0**110010011
Offspring of Parent X	**0**1101**1**110010011

Figure 4.2: Mutation Example

4.3.4.2 Crossover

The crossover operator is a local search operator that combines two parents to produce two offspring. This operator explores a specific area of the search space with the objective being to produce better offspring from good parents.

In the genetic algorithm used by [GOLD89], offspring are created by combining string fragments (portions of the string) which are obtained from each of the parents. The fragments are determined by using a randomly chosen crossover point. This method is referred to as *one point crossover* and an example is shown in Figure 4.3.

In the example, the randomly chosen crossover point is selected and divides the string into two fragments. The first fragment (all characters before the selected crossover point) from parent X and the second fragment from parent Y (all characters after the crossover point) are combined to form offspring Z1. Offspring Z2 is produced using the second fragment of parent X and the first fragment of parent Y.

Parent X	**01100**111010101
Parent Y	11111**000011110**
Resultant offspring Z1	01100000011110
Resultant offspring Z2	11111111010101

Figure 4.3: One point Crossover example and resultant offspring

4.3.5 Control parameters

There are two main control parameters that have to be set. These values will affect the performance of the algorithm.

- Population size – A larger population size increases computational effort but also allows the algorithm to explore a larger area of the search space during the initial generations. Smaller population sizes will reduce the time taken per generation but may mean that large areas of the search space are not explored during the initial generations (depending on the complexity of the problem).
- Genetic operator probability – This parameter specifies the probability of the genetic operator being applied to an individual. In the case of mutation, if the mutation probability is too high, then the genetic algorithm becomes more of a random search and will take longer to converge. A low mutation probability reduces genetic diversity and the algorithm may converge prematurely. In the case of crossover, a high crossover probability may result in the algorithm converging quickly but prematurely while a low crossover probability will reduce the local search and as a result reduce the convergence rate. The ideal genetic operator probabilities will vary depending on the problem [SRIN94].

4.4 Advances in Genetic Algorithms

The genetic algorithm detailed in sections 4.3.1 to 4.3.5 describes the standard genetic algorithm used by Goldberg [GOLD89]. However, genetic algorithms have been modified to solve various problems. Some of these modifications have been influenced by the development of other evolutionary computing techniques such as genetic programming.

Sections 4.4.1 to 4.4.5 describe some of the common changes made to the genetic algorithm.

4.4.1 Control models

A control model describes the way in which a genetic algorithm is executed. These models control the way in which the newly created offspring are introduced into the population. Sastry and Goldberg [SAST05] refer to these control models as replacement techniques. Bruce [BRUC95] describes three different control models, namely the generational, steady-state and varying population size control models.

G*enerational control model* - For every generation, a new population consists entirely of offspring created by parents from the previous generation (offspring could be duplicates of parents). This model is one of the most common and popular models that have been used as it is very easy to implement but does require an extra control parameter (the number of generations). This control model is also known as the *replace all* model [SAST05]. An advantage of the generational control model over the steady-state model is that since all individuals in the population are replaced, the probability of maintaining genetic diversity increases (how offspring are created will also affect genetic diversity).

In the *steady-state control model*, only a single population is used and offspring are immediately added to the population by replacing the least fit individuals. The individuals chosen for replacement are selected using an inverse selection method that favours individuals with poor fitness values. In terms of computational resources, this model uses only a single population and therefore uses less memory than the generational control model [BRUC95]. An extra control parameter indicating the number of individuals to replace is required. If the number of individuals that are replaced is small, then the algorithm may converge quickly but prematurely (this will also depend on the design of the genetic operators). If the number of individuals to be replaced is increased, then genetic diversity increases and the convergence rate is reduced.

Research has shown that, depending on the problem, different population sizes may be preferred at different stages of evolution [HINT96]. The *varying population size model*, which could be implemented as a variation of either the generational control model or the steady-state control model, changes the size of the population from one generation (or iteration) to another depending on the fitness of the individuals in the population. Poorer fitness among individuals results in an increase in the population size for the next generation in order to diversify the search and possibly improve the overall fitness. If good fitness

values are found in the population, then the population size decreases for the next generation (or iteration) in order for the algorithm to converge to a single solution [BRUC95].

4.4.2 Individual representation

Rather than using binary strings, many other representations have been used depending on the type of problem that must be solved. Other representations used include single or multi-dimensional arrays, strings, vectors, and tree structures ([AFFE09], [BEAS93]).

4.4.3 Initial population creation

The initial population should represent and cover as much of the search space as possible (known as uniform coverage). This reduces the possibility of the algorithm converging prematurely. Having no duplicate individuals in the initial population increases the diversity of a population and allows for a greater coverage of the search space.

In the standard genetic algorithm, individuals in the initial population are created randomly. Some studies such as [CART96], [CALD97] and [RAMS93] have created individuals in the initial population by using a set of rules (referred to as heuristics). These heuristics assist in the creation of a fitter initial population which could improve the probability of finding acceptable solutions ([BEAS93], [RAMS93] and [DIAZ07]). In the case of the school timetabling problem, an example of a heuristic would be to give priority to teachers that teach the most number of lessons. In scheduling problems, an initial population may also consist of previous solutions with the objective being to improve the quality of these solutions. In the case of the school timetabling problem, a previous solution could be the timetable that was used in the previous year or a timetable that was manually created by the staff.

4.4.4 Tournament Selection

One of the most commonly used selection methods, other than fitness proportionate selection, is *tournament selection*. A group of individuals are randomly chosen from the population. The fittest individual of the group (called a tournament) is selected as a parent. The tournament size is problem dependant and must be specified. A small tournament size (low selection pressure) results in a more random based selection. A large tournament size (greater selection pressure) results in the selection method becoming more elitist i.e. stronger individuals have a better chance of being selected. Tournament selection is easy to implement and is faster (in runtime) than fitness proportionate selection.

```
Best = Randomly chosen individual
For Loop = 2 to T
{
        Ind = Randomly select participant as Contender
        If Fitness(Ind) is better than Fitness(Best)
                Best = Ind
}
Parent = Best
```

Figure 4.4: Standard tournament selection algorithm

4.4.5 Genetic operators

In the standard genetic algorithm, crossover and mutation are used as genetic operators. In recent work, genetic algorithms have been implemented using only the mutation operator [BEAS93]. The motivation for this type of genetic algorithm is based on organisms in nature that reproduce asexually. In his review of previous work, Beasley [BEAS93] reports that a genetic algorithm using only mutation resembles a primitive form of evolution. Spears [SPEA93] found that a suitably modified mutation operator may perform just as well as the crossover operator.

As genetic algorithms were used to solve various problems, standard genetic operators were found to be insufficient when applied to different problem domains. These operators were then varied or changed based on the problem domain. Section 4.4.5.1 covers different types of mutation operators while section 4.4.5.2 covers the different types of crossover operators that were cited in the literature. Section 4.4.5.3 describes the reproduction operator. Finally, section 4.4.5.4 discusses the difference between application rates and operator probabilities.

4.4.5.1 Mutation

Goldberg initially described mutation as a secondary operator [GOLD89]. However, the use of different representations for solving problems and the development of evolutionary algorithms have resulted in a greater reliance on the mutation operator to explore the search space.

Boundary mutation is used for individuals that are represented using integers or float values [OBIT98]. With this mutation, a gene changes to an upper or lower bound value. For example, consider an individual 6-3-4-7. A boundary mutation will result in changing the 4 to a lower or upper bound value such as 0 or 10. The resultant offspring will be either 6-3-0-7 or 6-3-10-7.

Uniform mutation [OBIT98] changes the gene to a random value within a specified range. In the previous example, the first gene (6) will change to a random value where the range is

specified by the user. If the range is set between 0 and 4, then the resultant offspring will be 1-3-4-7, 2-3-4-7 or 3-3-4-7.

4.4.5.2 Crossover

Besides the one point crossover described in section 4.3.4.2, several other variants of the crossover operator exist.

Two point crossover [SPEA91a] selects two crossover points and the bits between the crossover points are exchanged between individuals. In the example in Figure 4.5, two crossover points are selected. The fragments between these two crossover points are 1110 and 0000 for each parent respectively. These fragments are then swapped between the parents resulting in offspring Z1 and Z2.

Parent X	01100**1110**10101
Parent Y	11111**0000**11110
Resultant offspring Z1	01100000010101
Resultant offspring Z2	11111111011110

Figure 4.5: Two point crossover

In *uniform crossover* [SPEA91a], bit positions of the two parents are randomly selected. The bits in these positions are then swapped between the parents resulting in offspring Z1 and Z2. In the example in Figure 4.6, five bit positions are randomly selected. The values in these bit positions are swapped between parents resulting in offspring Z1 and Z2. The uniform crossover operator is used to provide a greater exploration of the search space and is controlled by the number of bits that will be exchanged between parents [SPEA91b].

Parent X (crossover points in bold)	0**1**100**11**101**0**10**1**
Parent Y (crossover points in bold)	1**1**111**00**001**1**11**0**
Resultant offspring Z1	01100001011100
Resultant offspring Z2	11111110010111

Figure 4.6: Uniform crossover

Goldberg et al. [GOLE89] implemented the *"Cut and Splice" crossover* operator. This operator is similar to the one point crossover operator with the exception that each parent may have a different crossover point as depicted in Figure 4.7. This results in two or more offspring having different string lengths. According to Mitchell [MICH98], producing individuals with varying string lengths is advantageous as the probability of a string containing the necessary information to produce an acceptable solution is greater.

Parent X (crossover points in bold)	**011**00111010101
Parent Y (crossover points in bold)	11111**000011110**
Resultant offspring Z1	011000011110
Resultant offspring Z2	1111100111010101

Figure 4.7: "Cut and Splice" crossover operator

Eiben et al. [EIBE94] considered a *multiparent crossover* operator. In this case, crossover is applied to three or more parents resulting in three or more offspring. According to Eiben [EIBE95], the use of more than two parents allows for greater diversity, thus reducing the possibility of premature convergence.

Figure 4.8 illustrates a crossover operator involving three parents W, X and Y resulting in an offspring Z1, Z2 and Z3. The first fragment from parent W, the second fragment from parent X and the third fragment from parent Y result in offspring Z1. This is an example of diagonal crossover and involves two crossover points. Alternative implementations of this operator include choosing multiple crossover points as well as recombination based on occurrence (the bits that are included the most in the parents are included in the offspring [EIBE94]) or fitness (the number of bits from each parent is proportional to the fitness of the parent).

Parent W	**1111**0000***1111***
Parent X	**0011**0011***0011***
Parent Y	**1010**1010***1010***
Resultant offspring Z1	111100111010
Resultant offspring Z2	001110101111
Resultant offspring Z3	101000000011

Figure 4.8: Diagonal crossover operator

4.4.5.3 Reproduction

According to Banzhaf [BANZ98], the reproduction operator creates an offspring by copying the parent (the offspring is an exact duplicate of the parent).

4.4.5.4 Genetic operator probability versus genetic operator application rate

In the standard genetic algorithm [GOLD89], each genetic operator is applied with a given probability. Genetic operator application rates can also be used and differ from operator probabilities in that it specifies how many individuals in the new population are created using each genetic operator. For example, having crossover and mutation application rates of 70% and 30% respectively indicate that 70% of the offspring are created using the crossover

operator while 30% of the offspring are created using only the mutation operator. A high mutation application rate results in a more random search while the local search is reduced. As a result, the genetic algorithm will take longer to converge. A high crossover application rate will result in a more local search which may increase the possibility of premature convergence.

4.5 Chapter Summary

This chapter provided a description of the genetic algorithm. Initially, the standard genetic algorithm implemented by Goldberg [GOLD89] was described. However, in order to use genetic algorithms to solve various problems, it may be necessary to modify the genetic algorithm in terms of the representation used, initial population generation, selection methods and genetic operator design. The advancements described in section 4.4 are some of the more common variations made to the standard genetic algorithm.

Chapter 5 – Genetic algorithms and the School Timetabling Problem

This chapter describes previous work on the application of evolutionary algorithms to the school timetabling problem with a specific focus on genetic algorithms. The following is described for each study: the school timetabling problem and its constraints, the representation used, initial population generation, selection methods, genetic operators used and the results obtained. Section 5.1 deals with genetic and evolutionary algorithms used to solve the school timetabling problem. The problem has also been solved by combining genetic algorithms with other techniques and section 5.2 describes this type of research. Section 5.3 reports on comparative studies involving genetic algorithms when solving the school timetabling problem. Section 5.4 provides an overview of the lessons learnt based on the experiences reported in these studies.

5.1 Evolutionary or genetic algorithms

Bedoya et al. [BEDO04] developed a genetic algorithm to solve the school timetabling problem. The hard constraints for the problem are:

- No class clashes or teacher clashes.
- A class must be taught a particular subject a maximum of once in a day.

This problem did not have any soft constraints. A matrix representation was used with each row representing a class and each column representing a period. All initial timetables were created by randomly allocating the tuples to the timetable. Each constraint was allocated a weighting based on its priority and the fitness value (measure) was calculated by finding the weighted sum of the constraint violations. Clashes were given the highest priority as a clash free timetable is at least usable. Fitness proportionate selection was used. A generational control model was adopted. The crossover operator was not implemented as recombination of two timetables would result in duplicate or missing tuples and would require repair operators to deal with this problem. This repair operator will need to find all duplicate or missing tuples and reallocate them to the timetable, adding to the processing overhead. The only genetic operator used was the mutation operator which searched for and swapped constraint violating tuples. Repair operators were not needed for this operator since tuples are swapped within the timetable and thus no tuples would be duplicated or lost during

mutation. The genetic algorithm was applied to a small problem involving four classes, 10 subjects and 19 teachers. The approach induced timetables that satisfied all the constraints.

Di Steffano et al. [DIST01] developed a genetic algorithm to solve the Italian school timetabling problem. The hard constraints for the problem are as follows:

- Room constraints - The size of the room must accommodate the size of the class.
- No class clashes, teacher clashes or venue clashes.
- Teacher availability requirements must be met.
- Certain lessons must be allocated to specific periods.
- Lunch breaks must be at different hours for each class.

The soft constraints for the problem are:

- Subject preferences should be met.
- Lessons should be evenly distributed throughout the week.
- Timetables should be compact.
- Distances between venues must be as short as possible.

Each individual in the population was represented as an integer array with the length being the total number of tuples. The array index represented the tuple reference number and the integer value stored in each cell was the period allocated to that particular tuple. When creating a timetable, certain tuples (specified by the school) had to be allocated to specific periods so these tuples were allocated first. The remaining tuples were then randomly allocated to the timetable. Tournament selection was used. The generational control model was used with elitism incorporated whereby the best individual in the population is copied across to the next generation. Crossover was applied followed by an intelligent mutation operator and an improvement mutation operator. Intelligent mutation reduced the number of hard constraint violations by incorporating hill climbing (only swaps that improved the cost of the timetable were accepted). Improvement mutation searched for and removed free periods in the timetable. As a result, all tuples after the free period were then moved one period forward. The changes resulting from the improvement mutation were only accepted if the timetable remained feasible. The algorithm followed a phased approach where soft constraints were only dealt with once the hard constraints of the timetables were satisfied. The authors describe the results as remarkable due to the positive feedback provided by the school administrators. The timetables created did not require any manual adjustment from

the school administrators. The concept of using an array with the tuple number as an index was a good approach that removed the possibility of having duplicate or missing tuples in a timetable. The authors also found that genetic operators could be easily designed using this representation.

Caldeira et al. [CALD97] used a genetic algorithm to solve the school timetabling problem for a small school. The hard constraints dealt with are:

- No teacher clashes, venue clashes or class clashes.
- Teacher availability requirements must be met.

The soft constraints are:

- Class and teacher timetables should be compact.
- A one hour lunch break should be scheduled for each class between 12:00 and 15:00.
- The number of lessons should not exceed a specified daily limit.
- Lessons for each subject should be evenly distributed throughout the week.
- Any free periods should be allocated to the end of the day or the beginning of the day.
- Classes and teachers preferences should be met.

The authors used an array to represent the timetable and the array size is determined by the number of tuples. Each array index value is the tuple number and each element in the array stores a period of the week. When creating a timetable, tuples with the fewest feasible periods on the timetables are given priority. The authors found feasible timetables in the initial population so the main objective of the genetic algorithm was to optimize the quality of these feasible timetables. The fitness of an individual was the weighted sum of the constraint violations. Fitness proportionate selection was used. The steady-state control model was employed. The genetic operators used were uniform crossover and mutation. To further reduce the constraint cost, a second mutation operator was also introduced that looked for constraint violating tuples and moved it to the closest unallocated period. The algorithm was applied to a school timetabling problem with four teachers and four classes for which it was able to produce feasible solutions of an acceptable quality.

Fernandez et al. [FERN99] solved a large school timetabling problem involving 41 classes, 109 teachers and 37 venues. The hard constraints of the problem are:

- No class clashes, teacher clashes or venue clashes.
- No repetition of subjects in a day.
- Consecutive period requirements must be met.
- Teacher availability requirements must be met.
- An allocated venue must have the capacity to hold the number of students in the class.
- Each class must be allocated a free period during the day for a break.

The soft constraints are:

- Teacher and class timetables should be compact.
- Class-teacher meetings should be evenly distributed throughout the week.
- Subjects should be evenly distributed throughout the week.
- Free periods for class timetables should be moved to either the beginning or the end of the day.
- Teacher preferences should be met.

The timetable was represented as a two-dimensional matrix with the rows representing the periods and the columns representing the classes. Timetables were initially created by ordering tuples according to priority. Lessons with consecutive periods were prioritized first, followed by lessons with specific rooms and lastly, teacher priority. Tuples were allocated to the timetable such that the hard constraint cost was minimized. Each constraint was weighted depending on its importance and the fitness value of an individual was calculated by the weighted sum of the constraint violations. Fitness proportionate selection was used. The steady-state control model was adopted. One of three crossover operators and a mutation operator were applied. The three crossover operators were one-point crossover, three point crossover and uniform crossover. A constraint focused mutation operator was used that searched for constraint violating tuples and attempted to move them to a feasible and empty period on the timetable. If there are no feasible, empty periods available, then the mutation is cancelled. The timetable induced by the genetic algorithm was compared to a manually created timetable and was found to be better as it was feasible and satisfied more soft constraint violations. However, the timetables produced using the manual system

were more compact and contained half the number of free periods when compared to the timetables found by the genetic algorithm.

Abramson et al. [ABRA91a] developed a genetic algorithm to solve the school timetabling problems for a group of generated data sets. The only constraint for this problem was that there must be no class clashes, teacher clashes or venue clashes. The timetable was represented as an array of periods with each period containing a set of tuples. The timetables in the initial population were randomly created. The cost of the timetable was determined by counting the number of clashes. Individuals from the population were randomly selected as parents. The genetic operators used were one point crossover and mutation, both of which incorporated hill climbing. In most cases, the implementation of the crossover and mutation operators resulted in a loss of tuples. This problem was solved using a repair operator which removed duplicate tuples and included missing tuples. This repair operator added to the processing overhead of the overall algorithm but was a necessary process. The genetic algorithm was applied to nine data sets. The number of classes, teachers and venues vary between three and 15 with a constant of thirty periods per week. Feasible timetables were found for each of the data sets. The study also investigated the concept of parallel processing and this was found to substantially reduce the algorithm runtime.

Beligiannis et al. [BELI08] solved the Greek school timetabling problem. The hard constraints for the problem are:

- No class clashes or teacher clashes.
- Teacher availability requirements must be met.
- Any free periods must be allocated to the last period of the day.
- All subclass and co-teaching requirements must be met.

The soft constraints are:

- Teacher timetables should be compact.
- Free periods should be uniformly distributed amongst all teachers and free periods for each teacher should be uniformly distributed amongst all days he/she is available at school.
- The number of teaching periods for each teacher should be evenly distributed over the days that he/she is available at the school.

- Repetition of subjects in a day should be minimized.

A timetable was represented by a two-dimensional array with a row representing a class, a column indicating a specific period and the intersection of each row and column specifying the teacher that is required to meet the class at that time. The subjects taught were not considered as it was assumed (by the authors) that teachers knew which subjects they were required to teach as well as the number of periods required to teach that subject. A randomly generated initial population was created. The fitness of each individual was calculated by the weighted sum of the constraint violations with clashes given the highest weighting so that the timetables were at least usable. A linear rank selection method was used and the steady-state control model was employed. The crossover operator was not applied because trial runs found that it did not improve the performance of the algorithm. Mutation was the only genetic operator used and involved the swapping of either two randomly chosen tuples or two constraint violating tuples. Swaps were designed such that teacher availability and co-teaching requirements were not violated. The algorithm was applied to seven data sets, with the number of teachers ranging from 18 to 35 and the number of classes ranging from six to 13. Feasible timetables that were of a good quality were induced and these timetables were found to be better than that of two other effective, unnamed algorithms.

Filho et al. [FILH01] implemented a genetic algorithm to assist administrative staff at a Brazilian school. The hard constraints are that there must be no class clashes or teacher clashes. The only soft constraint was teacher period preferences. Tuples were ordered according to higher teacher seniority (primary heuristic) followed by teachers with the largest number of preferences (secondary heuristic). These tuples were allocated to randomly chosen periods in the timetable. The algorithm used a generational control model with a varying population size. Three mutation operators were implemented with two of these mutation operators aimed at removing class clashes and teacher clashes respectively. The third mutation operator (teacher preference operator) attempted to reduce the soft constraint cost. The authors concluded that the results were promising and would aid teachers in a task that normally took a very long time.

Tongchim [TONG99] proposed solving the school timetabling problem by using three parallel processing models of genetic algorithms. The models used in this case were the coarse-grained genetic algorithm, the fine-grained model and the master-slave model. The hard constraints for this problem are listed below.

- No class clashes, teacher clashes and venue clashes for specialized subjects.
- Lesson preferences must be met.

The problem did not have any soft constraints. A 2*n* array representation was used where *n* is the number of subjects (each subject is taught by a different teacher). Each cell stored a tuple consisting of the venue and the period. The fitness of an individual was the sum of all the constraint violations. The selection method used was tournament selection with a tournament size of three. The genetic operators applied were mutation and crossover. The first model, a master-slave model used a single population. The processes of fitness evaluation and application of genetic operators were each performed in parallel (simultaneously). The second model, a coarse grained model, divided the population into multiple subpopulations, which evolved independently with little or no interaction with other subpopulations. Each subpopulation was evolved on a different processor. The third model, a fine grained model, used a larger number of parallel processors and divided the population into a larger number of subpopulations. The coarse grained genetic algorithm was the best performing genetic algorithm and was also the easiest to implement.

Bufe et al. [BUFE01] used a hybrid algorithm to solve the German school timetabling problem. The hard constraints are:

- No teacher clashes, class clashes or venue clashes.
- Co-teaching requirements must be met.
- Teacher availability requirements must be met.
- Certain subjects must be held in specific venues.

The soft constraints are:

- Any free periods for classes should be placed at the end of the day.
- Certain teachers should have a minimum number of free days per week.
- Some lessons that require double periods and fortnightly allocation should be met.
- Subjects for each class should be evenly distributed throughout the week.
- Class and teacher timetables should be compact.

A two-dimensional matrix was used to represent a timetable. Initial timetables are created by randomly selected tuples and allocating them to feasible periods such that the hard and soft constraint cost is minimized. The genetic algorithm follows the steady-state control

model, where 40 percent of the worst individuals in the population are replaced by newly created offspring. Three mutation operators were used and involved removing allocated tuples, placement of unallocated tuples and swapping of two tuples respectively. The algorithm was applied to a problem that involved 61 teachers, 23 classes, 49 venues and 351 lessons. No feasible timetables had been induced. Conclusions were made that future research regarding the use of intelligent operators that minimize the cost of initial timetables and mutation operators that search for constraint violations was necessary.

Wilke et al. [WILK02] attempted to solve the German school timetabling problem with the following hard constraints:

- Teacher and class timetables must be compact.
- Free periods for teachers for breaks.
- Any free periods must be allocated towards the end of the day.
- Certain subjects must be evenly distributed throughout the week.
- One free period to be allocated to classes for a lunch break.
- Some lessons must be taught in dedicated rooms.
- All co-teaching and subclass requirements must be met.

The only soft constraint is that teacher preferences should be met. An individual in the population was represented as a two-dimensional matrix with the rows representing the days of the week and the columns representing the periods. Each cell in the timetable was allocated a tuple consisting of an assigned room, a subject and a class-teacher combination. Individuals in the initial population were randomly created. The cost of the timetable was determined by the weighted sum of the constraint violations. The best two individuals of each generation are chosen as parents and the remaining parents are chosen using fitness proportionate selection. One of two crossover operators (one point and two point crossover) were randomly chosen and applied followed by three mutation operators. The two mutation operators swap lessons and randomly reallocate rooms respectively. The third mutation operator makes a copy of the best timetable in the population and then makes one swap between two tuples. Mutation operators that focus on soft constraints are applied if the genetic algorithm converges prematurely. The approach was applied to a large German high school timetabling problem. While most hard constraints were satisfied, feasible timetables were not induced.

5.2 Genetic algorithms with other techniques

Zuters [ZUTE07] implemented a neural network as part of a genetic algorithm to solve the school timetabling problem. This paper was a continuation of a previous study and attempted to use neural networks as a fitness function. These neural networks were trained using previous timetables as well as randomly generated timetables. This study focused only on timetable quality. The soft constraints for the problem are:

- Teacher lessons should be evenly distributed throughout the week.
- Class timetables should be compact.
- Lessons should start as early in the day as possible.
- Subjects for each class should be evenly distributed throughout the week.

A matrix representation was used with each row representing a class and the columns representing the periods. Each cell contained a subject that would be taught to the class during that particular period. This representation was chosen as room and teacher details were not available. Details of initial population generation as well as the selection method used were not provided. Mutation was the only genetic operator. Four separate fitness functions were used to assess each soft constraint. The main objective of this study was to determine whether any of the fitness functions could be replaced by a neural network. Zuters found that the neural network could replace only one of the four fitness functions. The function that could be replaced assessed the constraint dealing with starting lessons as early as possible.

Nurmi et al. [NURM07] used a combination of a genetic algorithm and tabu search to solve the school timetabling problem for schools in Finland. The hard constraints of the problem are the following:

- No class clashes, teacher clashes or venue clashes.
- Lesson preferences must be met.
- Teacher, class and venue availability requirements must be met.

The soft constraints of the problem are the following:

- Class and teacher timetables should be compact.
- The lessons for some classes should be placed as late in the day as possible i.e. initial periods for the day should be free periods.

- The lessons for some classes should end as early as possible on particular days.
- Each teacher should teach a set number of lessons in a day within a specified range.
- Teacher day and period preferences should be met.
- A class should only be taught a specific subject once in a day.

Information regarding representation, timetable creation and selection method used was not provided. Mutation was the only genetic operator and involved making a number of moves (swaps). Mutation incorporated both hill climbing and a tabu list. Hill climbing ensured that moves did not result in an increase in the constraint cost of the timetable. The tabu list stored moves that would result in a return to a previous candidate solution. The genetic algorithm was applied to the HDTT benchmark problem [ABRA91a] and real world primary and secondary school timetabling problems. The algorithm was only able to find solutions to some of the data sets in the benchmark problem but did find feasible timetables for the real world problems. The use of a tabu search was identified as an important factor in improving the performance of the mutation operator.

Cerdeira-Pena et al. [CERD08] implemented a hill climbing approach, a genetic algorithm and two hybrid algorithms to solve the school timetabling problem.

The hard constraints for the problem are:

- No teacher clashes or class clashes.
- Co-teaching and subclass requirements must be met.
- Teacher availability requirements must be met.
- Double period requirements must be met.

The soft constraints for this problem are:

- Maximum daily teaching limits for teachers should not be exceeded.
- Each teacher should have his/her lessons evenly distributed throughout the week.
- Teacher timetables should be compact.
- Subjects should be evenly distributed throughout the week.
- Teacher preferences should be met.

Each timetable was represented as a two-dimensional matrix with the rows of the table representing periods while each column represented a teacher. A cell of the matrix indicated

the class and subject. The cost of a timetable was determined by counting the number of constraint violations. The initial population consisted of previous solutions and randomly generated timetables.

The hill climbing approach (called RNA) explored the search space using double moves (a single swap and an optional second swap if a constraint violation occurs from the first swap). This process continued until no further improvements could be made to the timetable after a specified number of moves.

Two genetic algorithms were implemented with the algorithms differing only in the selection method used. In the first selection method (GAT), two parents are chosen, each from two pairs of randomly selected individuals. The second selection method (GAT4C) differs from the first in that four parents (rather than two) are selected and selection is without replacement. Genetic operators used were one point crossover and a mutation operator that made random swaps.

The final two approaches combined the hill climbing (RNA) algorithm with each of the genetic algorithms. The hybrid approach alternated the genetic algorithm and the RNA approach with the genetic algorithm applied first. The RNA approach was applied to each individual in the population and produced the offspring for a new generation. The alternation between the RNA approach and the genetic algorithm continued until a generation limit was reached.

The algorithms were applied to three sets of school data. The first school had six co-teaching and subclass groups, 70 classes and approximately 16 teachers. The best performing algorithm was the hybrid algorithm using the GAT4C selection method. The second problem contained 27 co-teaching and subclass groups, 333 classes and 71 teachers and best results were found with the hybrid algorithm using the GAT selection method. The third problem involved a Spanish high school with 11 teachers and 33 classes. The hybrid algorithm using the GAT4C selection method produced the best results. The authors concluded that, overall, the best performing algorithms were the hybrid approaches.

Rahoual and Saad [RAHO06] solved the school timetabling problem by hybridizing a genetic algorithm and tabu search. The hard constraints are:

- No class clashes or teacher clashes.
- Certain subjects must be allocated to specific periods.

The only soft constraint for the problem is that teacher preferences should be met. Each timetable was represented as an array, with the array index indicating the tuple number. Each tuple in the array is allocated to a period and a teacher. Timetables were created by allocating tuples to randomly chosen periods but no details were provided as to how the tuples were selected. The fitness of a timetable was determined by calculating the weighted sum of the constraint violations. A steady-state control model was used. Crossover and three mutation operators were used to create offspring. Each mutation operator focused on a particular constraint and incorporated both hill climbing (swapping of tuples was only accepted if the constraint cost was reduced) and a tabu list (to prevent cycling in terms of performing the same swap and returning to previous solutions).

The approach was tested on benchmark data sets provided by Abramson [ABRA91a] and the performance was found to be equivalent to that of the simulated annealing technique and the tabu search used by Abramson [ABRA93]. The approach was also tested using four generated data sets. The first set had 64 subjects, 12 teachers and 16 venues. The second set involved 100 subjects, 21 teachers and 25 venues. The third data set involved 150 subjects, 26 teachers and 31 venues. Data set four had 200 subjects, 33 teachers and 37 venues. The performance of the genetic algorithm using the constraint focused mutation operators performed better when compared to a genetic algorithm using a standard mutation operator that randomly swaps tuples. The hybrid algorithm found good results and the authors found that the tabu search minimized the possibility of premature convergence. This approach was also tested against a real world case study involving 500 groups, 3000 teachers, 5000 subjects and 200 venues. The manual creation of a timetable for this problem took approximately four weeks. In comparison to the manual approach, the genetic algorithm approach solved the problem in less than an hour.

The authors concluded that the hybrid algorithm provided great flexibility and efficiency. A contributing factor towards the success of the algorithm was the use of three separate mutation operators that were each dedicated to reducing the cost of one specific constraint. Another key contributing factor was the use of the tabu search incorporated with the mutation operator.

5.3 Comparative studies

Colorni et al. [COLO98] implemented simulated annealing, a genetic algorithm and tabu search to solve the school timetabling problem. The approaches were applied to benchmark

problems as well as an Italian high school timetabling problem. The hard constraints for the school are:

- No class clashes or teacher clashes.
- All class timetables must be compact.

The soft constraints for the problem are:

- Each teacher should have between two and four teaching periods a day.
- A different teacher should be allocated to the last period for each day.
- Lessons should be evenly distributed throughout the week.
- Double period requirements should be met.
- Teacher timetables should be compact.

A two-dimensional matrix was used to represent a teacher timetable where the rows represent the teachers and the columns represent the periods of each day. Each cell in the matrix represented a teacher activity such as teaching and development lessons. The initial population was randomly created. The selection method used was fitness proportionate selection. The genetic operators used were crossover and mutation. Repair operators were used to remove duplicate tuples and to reallocate missing tuples caused by the genetic operators. The authors found that all three approaches performed better than the manual system. The tabu search produced better timetables than the genetic algorithm, with simulated annealing producing timetables with the most constraint violations. While the tabu search performed the best, the genetic algorithm produced a population of solutions providing users with a choice of timetables.

The school timetabling problem in the study conducted by *Wilke et al.* [WILK08] involved a German secondary school timetabling problem with 113 teachers, 100 rooms and 43 classes. The hard constraints for the problem are as follows:

- No class clashes, teacher clashes and venue clashes.
- Classes must be allocated to rooms that can accommodate the number of students in the class.

This problem did not involve any soft constraints. The four algorithms used were tabu search, simulated annealing, a genetic algorithm and the branch and bound algorithm.

Specifics regarding each of the algorithms were not provided. The tabu search performed 200 moves per iteration. The simulated annealing technique was given a time limit of two hours to find a feasible solution. The genetic algorithm used a population size of 30 and ran for a time of two hours. The fittest individual and 5% of the individuals were automatically selected as parents. No selection method was mentioned and the genetic operators used were mutation and crossover. For the branch and bound algorithm, the hard constraint regarding the minimum number of lessons per day was not considered. The simulated annealing technique performed the best, producing feasible solutions for all runs. The other techniques did not induce feasible solutions.

5.4 Chapter Summary

Based on the literature reviewed in this chapter, it is clear that the standard genetic algorithm used by Goldberg [GOLD89] has not been used as is and variations had to be made to the majority of the genetic algorithms in order to induce satisfactory solutions to the school timetabling problems. The following sections describe the most common changes reported in the literature as well as how the genetic algorithm approach in this study will be implemented (discussed in further detail in Chapter 7 and Chapter 8).

5.4.1 Representation

In the literature, either an array representation ([CALD97], [FERN99], [DIST01], [ABRA91a], [TONG99], and [RAHO06]) or a matrix representation ([BEDO04], [BELI08], [BUFE01], [NURM07], [CERD08], [WILK02] and [COLO98]) was used. The array representation is easily implemented and no repair operators are needed when using the crossover operator. The matrix representation directly represents a timetable, immediately satisfies certain constraint violations such as avoidance of class or teacher clashes (since a row or column could represent a teacher or a class) and allows for easy implementation of genetic operators e.g. swapping tuples within rows or columns. This study will use a matrix representation because of the advantages stated above.

5.4.2 Control model

Most studies opted for the generational ([BEDO04], [DIST01], [ABRA91a], [BELI08], [NURM07], [FILH08] and [CERD08]) or steady-state models ([CALD97], [FERN99], [BUFE01] and [COLO98]). For this study, a generational model will be adopted as the majority of the studies had successfully implemented this model.

5.4.3 Initial population generation

When creating a timetable, tuples are either randomly selected ([BEDO04], [FERN99], [ABRA91a], [BELI08], [CERD08], [WILK02]) or selected using heuristics ([FILH01] and

[CALD97]) such as teacher priority and the least number of feasible periods (this has been called saturation degree in a study by Carter [CART96]). Once a tuple is selected, it is then randomly allocated to a period ([ABRA91a], [BELI08], [FILH01], [CERD08], [WILK02]) or allocated to a period that minimized the hard (or soft) constraint cost ([FERN99], [DIST01]). This study will assess the effect of randomly choosing tuples for allocation against choosing tuples using heuristics. Once selected, a tuple will be allocated to the period that minimizes the hard and soft constraint cost. This strategy will minimize the overall cost of the initial population and increase the probability of finding a feasible timetable of a high quality.

5.4.4 Selection method

Most of the studies used either fitness proportionate selection ([BEDO04], [CALD97], [FERN99] and [WILK02]) or tournament selection ([DIST01], [TONG99] and [CERD08]) as these were the most popular selection methods. For this study, the tournament selection method will be used due to ease of implementation (individuals in the tournament are randomly chosen and the best individual is selected as a parent) and reduction in computational time [BLIC95] (fewer individuals are evaluated and used in tournament selection than in fitness proportionate selection).

5.4.5 Genetic operators

Most studies implemented the mutation and crossover operators. Studies that used a matrix representation ([ABRA91a], [BUFE01], [WILKE02] and [COLO98]) required the use of repair operators after crossover was applied in order to remove duplicate tuples and to reallocate missing tuples. Since this study will use a matrix representation, crossover will not be used in order to avoid the extra processing overhead that occurs when using repair operators.

In all the studies, mutation involved the swapping of tuples between periods. Some studies randomly swapped tuples while others ([DIST01], [FILH01] and [BUFE01]) searched for and swapped tuples that caused constraint violations to successfully solve their respective problems. Studies by [DIST01], [ABRA91a], [CERD08] and [RAHO06] incorporated hill climbing as part of the genetic operators to solve their respective problems. This study will use constraint focused mutation operators (similar to those used by [DIST01], [FILH01] and [BUFE01]) and will assess the performance of the genetic algorithm when using these operators with and without hill climbing.

5.4.6 Single phase versus multiphase

Most of the studies presented addressed hard and soft constraints in a single phase with varying degrees of success. Three studies ([AVEL07], [ALVA96] and [DIST01]) successfully used a multiphase approach that addressed timetable quality only if hard constraints had

been satisfied. This study will test the performance of a genetic algorithm when using a single phase approach and a multiphase approach.

5.4.7 School Timetabling Problems

From the previous work, it is evident that each of the genetic algorithms in the literature has only been evaluated for one specific type of school timetabling problem and has not been tested generally for different types of problems. This study will investigate the use of a genetic algorithm approach to solve different types of school timetabling problems i.e. school timetabling problems with different sets of constraints. It was also found that school timetabling problems differed between countries and while problems from Australia, Brazil, Italy, Spain and Greece were addressed, no research was done on the South African school timetabling problem which also differs from the other problems in terms of constraints and resources.

Chapter 6 - Methodology

6.1 Introduction

This chapter outlines the methodology used in order to attain the objectives described in Chapter 1. Section 6.2 re-emphasizes the two objectives as well as how each objective will be met. Section 6.3 describes the hypothesis testing that will be conducted to test the significance of results. Section 6.4 covers the selected school timetabling problems that will be used. Section 6.5 provides a brief overview of the hardware and software used in this study.

6.2 Fulfilling the objectives of the study

This section outlines the methodology used to fulfill the objectives listed in Chapter 1. Section 6.2.1 outlines the methodology for the first objective of using a genetic algorithm approach to solve different types of school timetabling problems. Section 6.2.2 describes the methodology for the second objective of evaluating a genetic algorithm that uses an indirect representation to solve the school timetabling problem.

Genetic algorithms start from random points in the search space and hence a single run may not produce a solution. Thus for each objective, thirty random number generator seed values will be tested for each process and each parameter value (the same set of seed values will be used when testing each process and each parameter). The significance of any results will be tested using hypothesis tests (described in section 6.3).

6.2.1 Objective One

The first objective (as outlined in Chapter 1) is to test the effectiveness of genetic algorithms in solving different types of school timetabling problems. An initial genetic algorithm approach based on the literature review (outlined in section 5.4 of Chapter 5) will be implemented. This will be referred to as a direct genetic algorithm (DGA) approach as the algorithm will evolve the timetables directly. The performance of the DGA will be improved iteratively by changing the processes of the genetic algorithm (if required) i.e. the representation, fitness calculation, the method of initial population generation, selection method and genetic operators. The control parameter values will also be fine-tuned in order to improve performance. These changes will be made based on the evaluation of the system when applied to the problems. This type of methodology is referred to as implementation driven research where the emphasis is on iteratively refining a system until it accomplishes the objectives that are required [JOHN11]. In the event that the DGA

performs poorly (i.e. feasible solutions cannot be found), then an analysis will be performed to find reasons for poor performance. The DGA will then be refined based on the reasons identified. The DGA will be tested on five school timetabling problems (described in section 6.4).

6.2.2 Objective Two

The second objective is to evaluate the performance of a genetic algorithm that uses an indirect representation when solving the school timetabling problem. All studies on the school timetabling problem (see Chapter 3 and Chapter 5) successfully used a direct representation that searched a solution space for an optimal timetable. Previous studies in the domain of job shop scheduling ([ABDE10]) and exam timetabling ([TERA95]) used a genetic algorithm with an indirect representation consisting of a string of instructions capable of building a schedule. The genetic algorithm using this representation was found to be very effective and according to Gutierrez et al. [GUTI02], reduced the search space. Research regarding genetic algorithms that use an indirect representation has not been conducted for the school timetabling problem and this study will evaluate this approach. A genetic algorithm using an indirect representation (IGA) will be applied to the five school timetabling problems described in section 6.4. Iterative improvements will be made to the IGA if necessary. The performance of the IGA will be compared to the performance of the DGA as well as other methods from the literature that solved the same school timetabling problems.

6.3 Hypothesis testing

Hypothesis tests are conducted to test for the significance of the results where applicable. Z-tests are used and the levels of significance, critical values and decision rules used for these tests are listed in Table 6.1.

Table 6.1: Levels of significance, critical values and decision rules

P	Critical Value	Decision Rule
0.01	2.33	Reject H_0 if Z > 2.33
0.05	1.64	Reject H_0 if Z > 1.64
0.1	1.28	Reject H_0 if Z > 1.28

6.4 The School Timetabling Problems

This section describes five types of school timetabling problems that will be used to assess the performance of the IGA and DGA (in terms of feasibility and quality). The first problem, the Abramson (or HDTT) problem, was introduced by Abramson et al. [ABRA93] and was the most commonly cited problem in the literature. This problem was chosen as most studies provided results which can be used for comparison purposes. Although there were several school timetabling problems discussed in the literature, only two of these problems were made publicly available by the authors at the time of this research. These two problems were two Greek school timetabling problems introduced by Beligiannis et al. [BELI08] and Valouxis et al. [VALO03] respectively and will also be used. Finally, two South African school timetabling problems are used as problems from the South African educational system, which differs from the other school timetabling problems in terms of constraints and timetable structure, have not been covered in the literature. For the two Greek problems and the two South African problems, sample timetables (timetables provided by the authors) were made available. An empirical comparison will be made between these sample solutions and the timetables induced by the DGA and IGA.

Section 6.4.1 covers the Abramson problem (also known as the HDTT problem). Section 6.4.2 and section 6.4.3 respectively describe the two Greek school timetabling problems. Finally, section 6.4.4 and 6.4.5 describes the two South African school timetabling problems (Lewitt and Woodlands).

6.4.1 The Hard Defined TimeTable (HDTT) school timetabling problem

The HDTT problem is a generated (non-real) school timetabling problem that was studied by [ABRA93], [RAHO06], [SMIT03] and [LIU09]. Each data set in the problem has the following characteristics:

Table 6.2: Characteristics of HDTT data sets

Data set	No. of teachers	No. of Venues	No. of Classes	Total No. of Periods
HDTT4	4	4	4	30
HDTT5	5	5	5	30
HDTT6	6	6	6	30
HDTT7	7	7	7	30
HDTT8	8	8	8	30

The only hard constraint for the problem is that there must be no class clashes, teacher clashes and venue clashes (HC_1).

When compared to other problems in the literature, the Abramson problem appears to be simple with a small number of classes, teachers and venues and only one constraint (no clashes). This, however, does not imply that the problem is easy to solve. Three studies ([RAHO06], [LIU09] and [SMIT12]) have described this problem as difficult since all the timetables produced (teacher, class and venue) must be dense i.e. each class, teacher and venue must be assigned to every period of the week with no free periods. No other problem found in the literature required timetables with this degree of density.

The mathematical formulation of the fitness function for this problem is simply the sum of all the clashes of a given timetables. The formula for the fitness function for this problem is shown below:

$$Min\ f(obj) = (class\ clashes + teacher\ clashes + venue\ clashes) \geq 0$$

In the formula above, the objective is to minimize the number of clashes. A fitness value of 0 indicates that a feasible timetable has been found.

6.4.2 The Valouxis Greek school timetabling problem

This problem was introduced by Valouxis et al. [VALO03]. The problem involves 15 teachers and six classes. A school week is comprised of five days and each day has seven periods. The hard constraints for this problem are:

- No class clashes or teacher clashes (HC_1).
- Class timetables must be compact. Any free periods must be allocated to the end of the day (HC_2).
- A teacher's workload must be evenly distributed throughout the week (HC_3).

- Lessons must be evenly distributed throughout the week (HC_4).

The soft constraints for the problem are as follows:

- Teacher preferences should be met (SC_1).
- Teacher timetables should be compact (SC_2).

Unlike the HDTT problem described in section 6.4.1, this problem contains a greater number of constraints that must be satisfied. This study has four hard constraints and two soft constraints and is equivalent to other studies in the literature in terms of average number of constraints. In a study of school timetable difficulty by Van Heuvan van Staerling [VANH12], it was found that one of these constraints, namely the teacher availability constraint, is classified as a difficult constraint to solve. The two distribution constraints have normally been specified as soft constraints in other studies ([CALD97], [LIU09], [BELI08] and [AVEL07]) but are specified as hard constraints in this study. This increases the difficulty of the problem as these constraints must be met rather than minimized.

Based on the constraints listed above, the formulation of the fitness function is shown below and follows the format of a similar formulation used by Beligiannis et al. [BELI08]:

$$Min\ f(obj) = \sum_{x=1}^{4} (HCV_x)\ \geq 0$$

In the above formula, HCV_X is the number of violations in the timetable for hard constraint HC_X. For this function, the calculated value must be greater than or equal to 0 and the objective is to minimize *f(obj)*. A feasible timetable is found if *f(obj)* = 0.

The formulation of the fitness function to calculate the soft constraint cost is a summation of the number of violations for each of the listed soft constraints.

$$Min\ f(obj) = \sum_{x=1}^{2} (SCV_x)\ \geq 0$$

In the formula above, SCV_X represents the number of soft constraint violations in the timetable for soft constraint SC_X. The calculated fitness value must be greater than or equal

to 0. A lower soft constraint cost indicates a better quality timetable. Therefore the objective is to minimize the soft constraint cost.

6.4.3 The Beligiannis Greek school timetabling problem

Beligiannis et al. [BELI08] introduced the Greek high school timetabling problem. This problem has seven data sets, each with a varied number of classes and teachers. Only six of the seven data sets were used because one of the data sets (high_school_06) has an error, where the total number of lessons for certain classes is more than the number of periods available. The six data sets used have the following characteristics:

Table 6.3: Characteristics of the Beligiannis data set

Problem code	No. of teachers	No. of classes	No. of co-teaching and subclass requirements	Total no. of periods
High_school_01 (HS1)	11	34	18	35
High_school_02 (HS2)	11	35	24	35
High_school_03 (HS3)	6	19	0	35
High_school_04 (HS4)	7	19	12	35
High_school_05 (HS5)	6	18	0	35
High_school_07 (HS7)	13	35	20	35

The hard constraints for this timetabling problem are:

- No class clashes or teacher clashes (HC_1).
- Teacher availability requirements must be met (HC_2).
- Any free periods must be allocated to the last period of the day (HC_3).
- All subclass and co-teaching requirements must be met (HC_4).

The soft constraints are:

- Teacher timetables should be compact. If this is not possible, then the number of free periods should be minimized (SC_1).
- Free periods should be uniformly distributed amongst all teachers while free periods for each teacher should be uniformly distributed amongst all days he/she is available at school (SC_2).
- The number of teaching periods for each teacher should be evenly distributed over the days that he/she is available at the school (SC_3).

- Classes should not have the same lesson in successive periods and, if possible, not even on the same day (SC_4).

There are eight constraints in total that need to be addressed which is more than most problems found in the literature. According to Van Heuvan Van Staerling [VANH12], the subclass and co-teaching requirements as well as the teacher availability requirements are difficult constraints to solve. There are also four soft constraints that need to be minimized which is more than most of the problems discussed in the literature. Thus, this problem is one of the more challenging problems when compared to the other problems in the literature.

Based on the above constraints, the formulation of the fitness function to calculate the hard constraint cost is shown below. The formulation follows a similar format to that used by Beligiannis et al. [BELI08].

$$Min\ f(obj) = \sum_{x=1}^{4} (HCV_x) \geq 0$$

In the above formula, HCV_X is the number of violations for hard constraint HC_X. For this function, the calculated value must be greater than or equal to 0 and the objective is to minimize *f(obj)*. A fitness value of 0 is desired as it indicates that the timetable is feasible.

The formulation of the fitness function to calculate the soft constraint cost is similar to the above formula:

$$Min\ f(obj) = \sum_{x=1}^{4} (SCV_x) \geq 0$$

SCV_X represents the number of soft constraint violations for soft constraint SC_X. The calculated fitness value must be greater than or equal to zero. A lower soft constraint cost indicates a better quality timetable thus the objective is to minimize the value of the above function.

6.4.4 The Woodlands Secondary school timetabling problem

The Woodlands secondary school data set has 30 classes, 40 teachers and 44 subjects. A school week is comprised of seven periods in a day and there are six days in a school week. The hard constraints for the problem are the following:

- No teacher clashes or class clashes (HC_1).
- All subclass and co-teaching requirements must be met (HC_2).

The soft constraints are the following:

- Class-period allocation preferences should be met (SC_1).
- Teacher preferences should be met (SC_2).
- Co-teaching and subclass allocation preferences should be met (SC_3).

The Woodlands problem has the largest number of teachers, classes and subjects when compared to the other problems in this study. While the total number of constraints is slightly less than the average number of constraints in the literature, the problem does have subclass and co-teaching requirements, a constraint that is difficult to solve according to [VANH12]. Unlike the Beligiannis problem, the number of classes and teachers involved in the subclass and co-teaching requirements are more than two thus increasing the difficulty of the problem. In addition, all class timetables must be dense (no free periods for classes).

The formulation of the fitness function that calculates the hard constraint cost is shown below:

$$Min\ f(obj) = \sum_{x=1}^{2} (HCV_x) \geq 0$$

HCV_X represents the number of hard constraint violations for each hard constraint HC_X. The objective is to minimize the cost of the timetable and a feasible timetable is one that has a hard constraint cost of 0. The formulation for the fitness function used to determine the quality of the timetable is shown below:

$$Min\ f(obj) = \sum_{x=1}^{3} (SCV_x) \geq 0$$

The objective is to minimize *f(obj)*. SCV_X represents the number of soft constraint violations in the timetable for soft constraint SC_X.

6.4.5 The W.A. Lewitt primary school timetabling problem

W.A. Lewitt Primary school has 19 teachers, 16 classes and 14 subjects. Each school day consists of a maximum of 11 periods and each school week is comprised of five days. The timetable structure differs based on grade. Some grades are not allocated the full 11 periods in the day. The hard constraints for this problem are the following:

- No teacher clashes, class clashes and in some cases venue clashes (HC_1).
- For all classes, mathematics must only be taught in the mornings (HC_2).
- Co-teaching requirements must be met (HC_3).
- Double period requirements must be met (HC_4).

The only soft constraint is that there must be a balance in the number of lessons a class is taught per day (SC_1).

This problem has four hard constraints and one soft constraint. This results in a total of five constraints which is one less than the average number of constraints for the problems in the literature. Two of the constraints, double periods and co-teaching requirements, are categorized as difficult according to [VANH12] and [SMIT12]. The majority of the lessons in this problem are specified as double periods, thus increasing the difficulty of the problem.

The mathematical formulation of the fitness function to calculate the hard constraint cost of a timetable for this problem is shown below:

$$Min\ f(obj) = \sum_{x=1}^{4} (HCV_x) \geq 0$$

In the above formulation, HCV_X represents the number of hard constraint violations for each hard constraint HC_X. Thus the fitness value is determined by the sum of the number of hard constraint violations for each hard constraint HC_X. The objective is to minimize *f(obj)* and a feasible timetable is one that has no hard constraint violations.

The mathematical formulation for the single soft constraint cost of lesson balance throughout the week is shown below:

$$f(obj) = SCV_1 \geq 0$$

SCV_1 is the number of violations for soft constraint SC_1. *f(obj)* must be minimized and a lower value indicates a better quality timetable.

6.5 System implementation details

The GA system was developed using Visual C++ 2008 and 2010. The random number generator function available in C++ is used to generate random numbers. The programs were developed on a computer with the following specifications: Intel Core 2 Duo CPU P8600 @ 2.40GHz, 2.00 GB RAM, Windows XP/Windows 7 Enterprise OS.

Simulations (trial and final) were run on several machines, namely:

- Intel Core 2 Duo CPU @ 2.40 GHz, 2.00 GB RAM, Windows XP/Windows 7 Enterprise OS.
- Intel Core I7 870 CPU @ 2.93 GHz, 4.00 GB RAM, Windows 7 64-bit OS.
- Intel Core I7 860 CPU @ 2.80 GHZ, 4.00 GB RAM (3.49 Usable), Windows 7 32-bit OS.
- Pentium Dual Core @ 2GHZ, 2.00 GB RAM, Windows Vista.
- Center for High Performance Computing. See www.chpc.ac.za/sun for cluster specifications

Results of simulations as well as analysis and calculations for hypothesis testing were performed using Microsoft Excel 2007 and 2010.

6.6 Chapter Summary

This chapter provides an overview of the methodologies used to achieve the objectives described in Chapter 1. The hypothesis test that will be used to test for statistical significance is then covered. The two genetic algorithm approaches will be applied to five school timetabling problems which are also described in this chapter. This chapter concludes by providing the technical specifications for the study.

Chapter 7 - A Genetic Algorithm Approach using a Direct Representation

This chapter describes a genetic algorithm approach that uses a direct representation (DGA) to solve the school timetabling problem. The chapter includes the overall genetic algorithm, the representation used, heuristics used during initial population creation, selection methods, genetic operators and control parameters.

7.1 Overall algorithm

The studies in the literature incorporated either a single phase approach that addressed hard and soft constraints simultaneously or a two phased approach that addressed hard and soft constraints separately (see section 5.4 in Chapter 5). It was intended that a comparison of both these approaches would be conducted as part of the requirements for objective one (section 6.2.1 of Chapter 6). While conducting initial runs, it was observed that the single phase approach performed poorly when applied to the five school timetabling problems. While feasible timetables were found, the quality of these timetables was poor and had twice as many soft constraint violations as the timetables induced using a two phased approach. The reason for this was that most of the swaps performed to reduce the soft constraint cost were rejected due to an increase in the hard constraint cost of the timetable. The average cost and best cost for the trial runs (single phase) are shown in Table 7.1.

Table 7.1: Trial runs for single phase approach

	Beligiannis (HS7)	Beligiannis (HS4)	Valouxis	Lewitt	Woodlands
Average HC Cost	0	0	0	1.44	1.5
Average SC Cost	210.78	118.11	82.89	27	11.5
Best HC Cost	0	0	0	0	0
Best SC Cost	201	106	73	23	11

As a result of the above observations, a two phased approach will be adopted. Phase 1 implements a genetic algorithm to find feasible timetables. During Phase 2, a genetic algorithm is once again used, but is aimed at improving the quality of the timetables (minimizing the soft constraint cost) while maintaining feasibility. Phase 2 is only performed if feasible timetables are found during Phase 1. Both Phase 1 and Phase 2 will follow the generational control model with a fixed population size for each generation. The

generational control model was chosen as most studies in the literature adopted this approach.

The genetic algorithms used in Phase 1 and Phase 2 differ from the genetic algorithm described in Chapter 4 in that the crossover operator is not implemented. As discussed in section 5.4 (Chapter 5), the use of a crossover operator would require repair operators to remove duplicate tuples and add missing tuples. The operator is therefore not used in order to avoid the extra processing overhead of the repair operators.

Section 7.2 describes how the initial population is created. Section 7.3 discusses timetable evaluation with respect to feasibility and quality. Section 7.4 describes the selection methods that are used while section 7.5 discusses the application of the proposed mutation operators. Finally, section 7.6 covers the control parameters for the DGA approach in this study.

7.2 Initial population creation

This section describes how the initial population is created. Section 7.2.1 describes the direct representation used. Section 7.2.2 discusses the algorithm for creating the initial population. Section 7.2.3 covers how the requirements of class-teacher meetings are converted to tuples. Section 7.2.4 describes the sequential construction method as a means to create individuals of the initial population. This section also outlines the different heuristics that will be used when selecting which tuple gets priority in terms of allocation. Finally, section 7.2.5 covers how the initial population is created for Phase 2.

7.2.1 Representation

Each individual in the population is a timetable that is represented using a two-dimensional matrix. As discussed in section 5.4 (Chapter 5), this was the most commonly used representation in previous studies. An illustration of this representation is shown in Figure 7.1. The rows of the timetable represent the periods for the week. Each column represents a class. The intersection (cell) of a row (period) and column (class) stores the remaining information in the tuple, usually the teacher, venue and subject. This representation removes the possibility of class clashes while teacher or venue clashes can be easily calculated by counting the number of duplicates in each of the rows. Other constraint violations can also be calculated by counting these violations when they occur based on the day/period.

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
1	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
2	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
3	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
4	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
5	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
6	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue
7	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue	Teacher/ Venue

Figure 7.1: Sample timetable structure

In Figure 7.1, the first column and first row indicating the period and class label respectively have been included for reference purposes and are **not** part of the structure. Each cell in the matrix contains the necessary information related to the lesson i.e. the teacher, venue, subject and other resources if necessary. The timetabling problem related to Figure 7.1 involves allocating a teacher and a venue to a class for each period.

7.2.2 Initial population creation process

A population of *N* individuals must be created to form the initial population (*N* is a control parameter). Each individual in the initial population is created using a sequential construction method (SCM) which is described in more detail in section 7.2.4.

7.2.3 Converting the class-teacher lessons list into a list of tuples

Each school specifies a requirements list consisting of all the class-teacher-venue meetings that must be allocated to the timetable. Table 7.2 shows a sample of the requirements list. Each of the meetings is then converted into individual tuples (as seen in Table 7.3).

Table 7.2: List of class-teacher lessons

Meeting No	Teacher	Class	Venue	Lessons	Double?
1	A	10b	12	4	1
2	B	12c	13	3	0
3	C	5d	4	7	0

The three meetings in Table 7.2 are converted into 14 rows of tuples, 4 rows for meeting number 1, 3 rows for meeting number 2 and 7 rows for meeting number 3. A portion of these tuples are shown in Table 7.3. As meeting number 1 has a double period

specification, only two occurrences (rather than four) of this meeting are reflected in Table 7.3. The "Double/Single" column value for these tuples is set to 1. This indicates that two double period tuples must be allocated to the timetable.

Table 7.3: Tuple table

Meeting No	Teacher	Class	Venue	Double/Single
1	A	10b	12	1
1	A	10b	12	1
2	B	12c	13	0
2	B	12c	13	0
2	B	12c	13	0
3	C	5d	4	0
3	C	5d	4	0
…	…	…	…	…

7.2.4 The sequential construction method (SCM)

The algorithm for the sequential construction method to create an individual for the initial population is shown in Algorithm 7.1 below. This method has not been discussed in previous studies and it is anticipated that this method would improve the overall fitness of the initial population and thus improve the possibility of finding feasible timetables. The effect of the SCM on the performance of the DGA approach will be covered in Chapter 9.

Algorithm 7.1: Sequential Construction Method

```
For LoopSCM = 1 to X
        While all tuples NOT allocated
                Use heuristic(s) to find tuple to allocate
                Allocate tuple to timetable (avoid violations)
        End While
        Calculate fitness of timetable
End Loop
BESTSCM = Best timetable (lowest violation count) created from LoopSCM
Add timetable BESTSCM to initial population
```

The sequential construction method creates *X* timetables (individuals) where *X* is a control parameter (SCM size). Each timetable in the SCM is created as follows:

- Sort the list of tuples in order of how difficult the tuple is to schedule. Heuristics have been successfully used in previous studies to assess difficulty (see section 5.4 in Chapter 5). The low level heuristics used in this study to assess difficulty are

explained in further detail in section 7.2.4.1. Secondary heuristics are used in the event that two or more tuples have the same difficulty. If secondary heuristics cannot find a difference in the difficulty between the tuples, then the tuple is randomly selected.

- Allocate each tuple to a feasible period (if possible) in the timetable. Allocation of tuples should avoid hard constraint violations if possible. If a feasible period cannot be found, then allocate the tuple to a randomly chosen period in the timetable. In the event that the timetabling problem includes soft constraints and there is more than one feasible period available, then the tuple is allocated to a minimum penalty timeslot i.e. a period that will result in a minimum soft constraint cost if the tuple is allocated to it. If several periods are both feasible and result in the minimum soft constraint cost, then the tuple is randomly allocated to one of these minimum penalty timeslots. The minimum penalty timeslot was also implemented by many studies in the literature (see section 5.4 in Chapter 5).
- Once all tuples have been allocated to the timetable, then evaluate the timetable by counting the number of hard and soft constraint violations.

The timetable with the best fitness (i.e. the lowest number of hard constraint violations) is added to the initial population. If two or more timetables have an equal best fitness, then the timetable with the better quality is added to the initial population. The SCM is implemented whenever an individual needs to be added to the initial population i.e. as many times as the initial population size.

7.2.4.1 Low-level Construction heuristics

The following sections describe the low-level construction heuristics that will be used to select tuples to allocate to the timetable. These construction heuristics can be used as primary or secondary heuristics. A primary heuristic is the first heuristic that is used to choose between tuples. Secondary heuristics are used as tie breakers.

Random Allocation

When using this heuristic, a tuple is randomly chosen from the list and is allocated to the timetable. In Table 7.3, any of the tuples could be chosen first. The random heuristic is applied continuously until all tuples have been placed in the timetable.

Largest degree

This heuristic gives priority to the tuple with the most number of lessons. In Table 7.1 above, meeting number 3 has the most occurrences (7 lessons). Any one of the seven tuples from meeting number 3 will then be allocated to a period in the timetable. As a result of the allocation, the number of lessons left to allocate for meeting number 3 is reduced by one

(from 7 to 6). The remaining list of tuples still to be allocated is then resorted and the next tuple with the most lessons is allocated to the timetable. This continues until all tuples have been allocated.

Consecutive periods

Priority in terms of allocation is given to the tuples that must be allocated as double periods. In Table 7.1, meeting number 1 has a double period requirement so tuples corresponding to this meeting are given priority and two of the four tuples are allocated to the timetable. One of these tuples is allocated to a period in the timetable and the other is placed in an adjacent position either above or below the first allocation. If a double period tuple cannot be accommodated, then it must be split into two single period tuples (if specified as a constraint, then this will adversely affect the cost of the timetable).

Co-teaching and subclasses

In a co-teaching or subclass heuristic, priority is given to tuples that include multiple classes that form several subgroups or a single class that is split into two or more groups. These tuples are added to the timetable first to ensure that all subclass or co-teaching requirements are met.

Session priority

Tuples containing resources that must be allocated to specific periods of the day are given priority when being scheduled. This is done to allow these tuples to be allocated to their required periods before any other tuples (with no session priority) can be placed in those periods.

Teacher degree

When using this heuristic, priority is given to tuples with teachers that have to teach the most number of lessons. All tuples related to this teacher then get preference and one of these tuples is then allocated to the timetable.

Class degree

This is similar to the teacher degree where priority is given to the tuples involving the class with the most lessons. For example, higher grades that have 45 lessons in the week are given priority over lower grades that have 33 lessons in a week. Therefore the tuples that involve the higher grades are allocated first (if given priority), followed by the tuples involving the lower grades.

Number of available days

The tuples involving the teacher with the least number of days get priority and one of these tuples is then allocated to the timetable. For example, Teacher A, who is only available for two days in the week, is given a higher priority than teacher B who is available for five days

in a week. Therefore, the tuples involving teacher A are allocated before the tuples involving teacher B.

Saturation degree

The saturation degree heuristic, used by Carter [CART96] in examination timetabling, involves calculating the number of feasible periods available for each tuple. A feasible period is one that does not result in any hard constraint violations upon allocating the tuple to it. Priority is given to the tuple with the least number of feasible periods available. The tuples are allocated to periods in the timetable in this order, potentially reducing the number of violations when the timetable is populated. An example is shown in Table 7.4 (this table is similar but not related to Table 7.3).

Table 7.4: Tuple list with saturation degree

Meeting No	Teacher	Class	Venue	Double (1) /Single (0)	Saturation Degree
1	A	10b	12	0	30
1	A	10b	12	0	30
1	A	10b	12	0	30
1	A	10b	12	0	30
2	B	12c	13	0	30
2	B	12c	13	0	30
2	B	12c	13	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
3	A	5d	4	0	30
4	D	10b	1	0	30
4	D	10b	1	0	30
4	D	10b	1	0	30
4	D	10b	1	0	30
4	D	10b	1	0	30
4	D	10b	1	0	30
5	C	12c	12	0	30
5	C	12c	12	0	30
5	C	12c	12	0	30
5	C	12c	12	0	30

Initially, all tuples have the same saturation degree (i.e. the number of periods) as the timetable is empty. The first tuple found is placed in the minimum penalty timeslot in the timetable as illustrated in Table 7.5 below, and removed from the tuple list.

Table 7.5: Timetable with tuple added

	5d	…	10b	…	12c
1					
2					
3			A: 12		
4					
5					
6					

The saturation degree for each tuple is revised, leaving any tuples involving class 10b or teacher A or venue 12 with one less available space in the timetable. The revised table with the new saturation degree values is shown in Table 7.6.

Table 7.6: Tuple list with updated saturation degree

Meeting No	Teacher	Class	Venue	Double/Single	Saturation Degree
1	A	10b	12	1	**29**
1	A	10b	12	1	**29**
1	A	10b	12	1	**29**
2	B	12c	13	1	30
2	B	12c	13	1	30
2	B	12c	13	1	30
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
3	A	5d	4	1	**29**
4	D	10b	1	1	**29**
4	D	10b	1	1	**29**
4	D	10b	1	1	**29**
4	D	10b	1	1	**29**
4	D	10b	1	1	**29**
4	D	10b	1	1	**29**
5	C	12c	12	1	**29**
5	C	12c	12	1	**29**
5	C	12c	12	1	**29**
5	C	12c	12	1	**29**

Once a tuple has been allocated, the saturation degree is updated. Priority is given to tuples with the lowest saturation degree. As shown in the example in Table 7.6 above, after allocation all tuples containing class 10b, teacher A or venue 12 now have 29 available periods that will not result in hard constraint violations. The tuples containing other teachers and venues are not affected. During the next iteration of the allocation process, all tuples with a saturation degree of 29 are given priority and one of these tuples (chosen using secondary heuristics) will be allocated to the timetable. This process continues until all tuples have been allocated to the timetable.

7.2.5 Initial population creation during Phase 2

Phase 2 focuses on improving the quality of the timetables i.e. reducing the soft constraint cost. Phase 2 only occurs if at least one feasible timetable is found during Phase 1. The population size (*N*) for both phases is the same. Three scenarios are possible with regard to the initial population of Phase 2:

1. Phase 1 produces more than *N* feasible timetables (over several generations) before the last generation is reached - In this case, the initial population for Phase 2 contains the first *N* feasible timetables (*N* is the population size used in Phase 1). As soon as *N* feasible individuals are added to the initial population of Phase 2, Phase 1 ends. In this event, Phase 1 may not reach the final generation.
2. The number of feasible timetables produced by Phase 1 is less than *N* - The initial population for Phase 2 will contain all the feasible timetables found in Phase 1. The remaining population members are randomly chosen from the feasible timetables in the last generation of Phase 1.
3. Phase 1 does not produce any feasible timetables - In this case, Phase 2 is not needed as no feasible timetables are produced and the run will terminate and be considered unsuccessful in finding a feasible timetable.

7.3 Evaluating a timetable for feasibility and quality

Two separate evaluations are performed to respectively determine the feasibility and quality of a timetable. In Phase 1, a fitness function is used to evaluate the feasibility of an individual. In Phase 2, an individual is evaluated based on the soft constraint cost (quality). The fitness functions described in sections 7.3.1 and 7.3.2 are used to calculate the hard constraint cost and soft constraint cost respectively.

7.3.1 Evaluating the feasibility of a timetable (Phase 1)

A timetable is evaluated in terms of feasibility by counting the number of hard constraint violations. A feasible timetable is one with a fitness value of zero, indicating no hard constraint violations. At least one hard constraint violation results in a timetable not being feasible. The greater the number of hard constraint violations, the further the timetable is from being feasible.

7.3.2 Evaluating the quality of a timetable (Phase 2)

The quality of a timetable is determined by counting the number of soft constraint violations. A lower soft constraint cost indicates that a better quality timetable has been produced. Soft constraints often conflict with each other and thus it may not be possible to remove all soft

constraint violations. The objective, therefore, is to minimize the number of soft constraint violations.

7.4 Selecting a parent

In Phase 1, individuals are selected as parents based on their hard constraint cost. During Phase 2, selection of parents is based on an individual's soft constraint cost. The following sections describe standard tournament selection and a modified version of it named variant tournament selection.

7.4.1 Standard tournament selection

Standard tournament selection was covered in section 4.4.4 of Chapter 4. A group of *T* individuals (known as a tournament) are randomly chosen from the population. The fittest of the *T* individuals is selected as a parent.

7.4.2 Variant tournament selection (VTS)

Variant tournament selection (VTS) is a less elitist variation of standard tournament selection. VTS differs from standard tournament selection in that the selected parent is not always the best individual but weaker individuals in the tournament also have a chance to be selected as parents. The first individual included in the tournament is given the status of champion. The current champion is then placed in a match against other individuals (contenders) in the tournament. The winner is either the fitter individual or is randomly chosen from the two individuals regardless of fitness. Algorithm 7.2 outlines variant tournament selection.

Algorithm 7.2: Variant tournament selection

```
Current Champion = Randomly selected individual from population
For Loop = 2 to Tournament Size
{
        Contender = Randomly selected individual from population
        Choose a random number from 1 to 3
        If Random number is 1
                Current Champion remains the same
        Else If Random number is 2
                Current Champion = Contender
        Else If Random number is 3
                If Fitness(Contender) is better than Fitness(Champion)
                        Current Champion = Contender
}
Parent = Current Champion
```

7.5 Genetic operators for Phase 1

The only genetic operator used is mutation (reasons for not implementing the crossover operator has been discussed in section 5.4.5 in Chapter 5). This study will follow a similar approach to that of studies in the literature (see section 5.4.5 in Chapter 5) where the mutation operators involved swapping teachers (or venues) between positions in the timetable. In the literature, some studies (such as [ABRA91a] and [DIST01]) incorporated hill climbing with their mutation operators. This study will test the mutation operators with (lines 2 to 20 of Algorithm 7.3) and without (lines 21 to 28 of Algorithm 7.3) hill climbing. In hill climbing, an offspring is rejected if its fitness is worse than the fitness of the parent. If the constraint cost is reduced or remains the same, then the swap is deemed successful. Lines 12 to 15 for Algorithm 7.3 reflect this evaluation.

Algorithm 7.3: Mutation operator

```
1.  Obtain parent using selection method
2.  If Hill climbing is used
3.  {
4.      HillClimbingSteps = 0
5.      Do
6.      {
7.          Offspring = Copy Parent
8.          For MutLoop = 1 to Swaps
9.          {
10.             Mutate Offspring
11.             Evaluate Fitness of Offspring
12.             If (OffspringFitness < PreviousOffspringFitness)
13.                 Accept swap
14.             Else
15.                 Reject Swap
16.         }
17.         HillClimbingSteps++
18.     }
19.     While (ParentFitness < OffspringFitness) AND (HillClimbingSteps < 500)
20. }
21. Else If Hill Climbing is not used
22. {
23.     For MutLoop = 1 to Swaps
24.     {
25.         Mutate Offspring
26.         Evaluate Fitness of Offspring
27.     }
28. }
29. Add Offspring to new generation
```

Mutation of the offspring is performed in lines 10 and 25. Sections 7.5.1 to 7.5.3 describe the Phase 1 mutation operators implemented by the genetic algorithm. Section 7.5.4 describes the concept of hill climbing.

7.5.1 Two violation mutation (2V)

This mutation operator searches for two cells that contain hard constraint violations. The two teachers (or venues) are then swapped within the timetable possibly resulting in the removal of hard constraint violations. This implementation was based on the mutation operators used by Di Stefano et al. [DIST01] and Filho et al. [FILH01].

An example of an application of this mutation operator is illustrated in Figure 7.2 and explained below. T1 represents teacher 1 being allocated to class 1 during period 1. While a venue could be involved, it is not represented as it is not part of the example below.

	Class			
Period	1	2	3	4
1	T1	T2	T3	T4
2	T1	T1	T2	T4
3	T4	T3	T2	T1
4	T3	T4	T1	T2
5	T3	T3	T2	T4
6	T2	T4	T4	T3

Clash here as T1 allocated to 2 classes

Clash here as T3 allocated to 2 classes

Clash here as T4 allocated to 2 classes

Figure 7.2: Finding 2 violations

The timetable segment in Figure 7.2 shows three clashes. In period 2, T1 has been allocated to both classes 1 and 2. In period 5, T3 has been allocated to two classes and in period 6, T4 has been allocated to 2 classes. The mutation operator firstly locates a clash, so it finds any one of the three teacher clashes. In this example, the clash found is in period 2, class 2. The two remaining possible clashes are during periods 5 and 6 where T3 and T4 clash respectively. Any one of the two clashes can be chosen. If T4 in period 6 is chosen as the second clashing tuple, then the contents of this cell and the contents of the first cell with a clash are swapped. This resultant mutation means that T4 moves to period 2 and T1 moves to period 6 (for class 2). This change is shown in Figure 7.3. The above example only reflects clashes but this mutation operator applies to any hard constraint violation. For example, if the availability constraints specify that T2 is not available to teach during period 6, then this is regarded as an availability violation. The constraint violation, once identified, is then swapped with another cell with a hard constraint violation.

	Class			
Period	1	2	3	4
1	T1	T2	T3	T4
2	T1	T4	T2	T4
3	T4	T3	T2	T1
4	T3	T4	T1	T2
5	T3	T3	T2	T4
6	T2	T1	T4	T3

Clash here as T4 allocated to 2 classes

Clash here as T3 allocated to 2 classes

Figure 7.3: Resultant timetable after 2 violation swap – Example 1

The swap shown in Figure 7.3 will fix the clashing situation in row 6, but the swap will result in a new clash occurring in period 2 where T4 is being allocated to two classes. If T3 in period 5 were to be swapped instead of T4, then the following (Figure 7.4) occurs:

	Class			
Period	1	2	3	4
1	T1	T2	T3	T4
2	T1	T3	T2	T4
3	T4	T3	T2	T1
4	T3	T4	T1	T2
5	T3	T1	T2	T4
6	T2	T4	T4	T3

Clash here as T4 allocated to 2 classes

Figure 7.4: Resultant timetable after 2 violation swap – Example 2

This is the ideal scenario as the swap has fixed two clashes. The worst case scenario will be for a swap to result in another clash. Note that the swaps only occur within a selected column (class) in order to avoid the movement of teachers to classes that they are not required to teach.

7.5.2 One violation mutation (1V)

This mutation operator, also based on the operators used by [DIST01] and [FILH01], follows the same procedure as 2V mutation with the difference being that only one constraint violation is located rather than two. Once a constraint violation is found, a second timeslot is then randomly chosen. The contents in the second timeslot may or may not result in a hard constraint violation but must be randomly chosen from the same class (column) as the first teacher in order to avoid the allocation of teachers to classes that they are not required to teach. Once chosen, the contents of the two cells are swapped. This results in three possible scenarios:

- Worst Case: A swap results in two constraint violations.
- Middle Case: A constraint violation is removed but causes a new constraint violation in another location.
- Best Case: The second cell chosen has a constraint violation and both violations are removed.

7.5.3 Hill climbing versus non-hill climbing operators

Genetic operators are potentially destructive in that they could continuously result in offspring that are worse than the parents [BANZ98]. Hill climbing will prevent this by only accepting swaps that improve the fitness of the individual. The disadvantage of hill climbing is that it is susceptible to local minima i.e. the algorithm will reach an area of the search space that appears to be promising (heading towards convergence) by ensuring that offspring always improve but instead the algorithm convergences prematurely [BEAS93]. As mentioned in section 5.4, this study will evaluate the use of mutation operators with and without hill climbing.

Similar to the studies by [CERD08] and [RAHO06], when hill climbing is used, it is implemented within the mutation process. While the timetable is being mutated, the currently mutated version is compared to either the parent or the timetable from the previous swap (if this swap has improved fitness). If the fitness of the new version is better than the current version, then the swap is accepted. If the older version has a better fitness then the swap is rejected. A limit is used to prevent the problem of premature convergence (to a local minimum) and to prevent long runtimes when the offspring cannot be further improved (lines 4, 17 and 19 in Algorithm 7.3). The mutation operators with hill climbing are referred to as two violation hill climbing (2VH) and one violation hill climbing (1VH) while the non-hill climbing operators are referred to as two violation non-hill climbing (2VNH) and one violation non-hill climbing (1VNH).

7.6 Genetic operators for Phase 2

The main objective of Phase 2 is to improve the quality of feasible timetables that have been produced during Phase 1. This section describes the mutation operators that are aimed at reducing the soft constraints cost of the individual. Three of the four operators discussed in this section are based on the mutation operators used in the literature. Hill climbing is used for all the operators in this phase to ensure that timetable feasibility is maintained and that timetable quality improves. Trial runs listed below (Table 7.7) also found that operators used

with hill climbing produce better timetables than operators without hill climbing. Sections 7.6.1 to 7.6.4 describe the mutation operators used for Phase 2 of the DGA approach.

Table 7.7: Trial runs for non-hill climbing mutation operators (Phase 2)

	Beligiannis (HS7)	Beligiannis (HS4)	Valouxis	Lewitt
Average HC Cost	0	0	0	0.67
Best HC Cost	0	0	0	0
Average SC Cost	186.55	94	74.22	26
Best SC Cost	174	88	68	22

7.6.1 Random Swap

The random swap was used in most studies in the literature to address both hard and soft constraint violations ([BELI08], [WILK02], and [TONG99]). This mutation operator randomly selects two cells from the same class and swaps the contents of these cells. Two randomly selected cells are shown in Figure 7.5. The chosen cells must be in the same class in order to prevent the movement of teachers to classes that they are not required to teach.

	Class			
Period	1	2	3	4
1	T1	T2	T3	T4
2	T3	T1	T2	T7
3	T4	T3	T2	T1
4	T3	T4	T5	T2
5	T3	T6	T2	T4
6	T2	T4	T8	T3

1st Randomly selected cell

2nd Randomly selected cell

Figure 7.5: Selecting cells in a random swap

The contents allocated to the two randomly chosen cells are then swapped. In Figure 7.5, if the randomly selected cells contain T1 (class 2, period 2) and T6 (class 2, period 5), then these two teachers are swapped where T6 moves to period 2 and T1 moves to period 5 as is shown below (Figure 7.6).

	Class			
Period	1	2	3	4
1	T1	T2	T3	T4
2	T3	T6	T2	T7
3	T4	T3	T2	T1
4	T3	T4	T5	T2
5	T3	T1	T2	T4
6	T2	T4	T8	T3

T6 now in this position

T1 now in this position

Figure 7.6: Resultant random swap

7.6.2 Row swap

The row swap begins by randomly selecting two different rows (which represent periods). The tuples in these rows are then swapped between the periods. The reason for using this mutation operator is that since all resources in the row are being swapped with resources in another row, new clashes cannot occur due to the resultant swap. However, this mutation can adversely affect the cost of other constraints such as the teacher availability constraint (a row swap may result in a teacher being moved to a day or period when they are unavailable to teach) and double period constraints (a double period is split due to a row swap). An example of a row swap resulting in a double period constraint violation is shown below:

	Row swap conflicting with hard constraint violation							
Period	1A	1B	1C	4A	4B	5C		
1	Found	Found	Found	Eng	Afk	Math		
2	Found	Found	Found	Eng	NSc	Math		SWAP ROW 2 with ROW 3
3	Found	Found	Afk	Acc	PE	Afk		

Period	1A	1B	1C	4A	4B	5C		
1	Found	Found	Found	Eng	Afk	Math		
2	Found	Found	Afk	Acc	PE	Afk		2 doubles are split
3	Found	Found	Found	Eng	NSc	Math		HC violation occurs

Figure 7.7: Consequence of a row swap results in double period split violation

As can be seen in Figure 7.7, a row swap may not cause any clashes, but the two double periods involving English and Mathematics have been separated resulting in two double period constraint violations.

7.6.3 One violation mutation (1V)

This operator is similar to the one described in section 7.5.2, however, this operator searches for soft constraint violations. The procedure begins by firstly locating a cell where a soft constraint violation occurs. Another cell is randomly chosen and the contents of the two cells are swapped resulting in the possibility that the soft constraint violation would be removed.

7.6.4 Two violation mutation (2V)

This operator, similar to the hard constraint version described in section 7.5.1, searches for two cells with soft constraint violations. Once found, the contents within the two cells are swapped. This could result in at least two soft constraint violations being removed.

7.7 Control parameters

The different control parameters used for the algorithm are outlined below and will affect the performance of the algorithm.

- SCM size – This parameter value indicates the number of timetables to create, of which the best timetable is added to the initial population. For example, an SCM size of one indicates that the first timetable created is added to the initial population. An SCM size of 20 indicates that 20 timetables will be created with the best timetable being added to the initial population. A larger SCM value will result in a fitter initial population but an increase in runtime.
- Population size – This was defined previously in section 4.3.5 (Chapter 4).
- Tournament size – This was defined previously in section 4.4.4 (Chapter 4).
- Number of swaps – This parameter value indicates the number of swaps that are performed when applying the mutation operator. It is anticipated that a low number of swaps will improve the individual but not at a sufficient rate. By increasing the number of swaps, the individual is improved at a faster rate. If this parameter value is set too high, then improvements are made until a point where the remaining swaps do not affect the cost of the individual. The effect of using different swap parameter values is discussed in greater detail in Chapter 9.

- Number of generations – This parameter value specifies the maximum number of generations for both phases. If the maximum number of generations is too low, the genetic algorithm may not be given enough generations to converge. If the generation limit is too high, then the algorithm will converge before the generation limit is reached and the extra generations will unnecessarily increase the runtime of the program.

7.8 Chapter Summary

This chapter presents a genetic algorithm approach that uses a direct representation. The chapter discusses the representation, initial population generation and the evaluation of each timetable in terms of feasibility and quality. The chapter then goes on to describe the selection methods and the mutation operators that will be available for use. Each of the heuristics used to create the initial population, the selection methods and the genetic operators listed will be compared in order to determine an ideal combination that will solve the five school timetabling problems listed in Chapter 6.

Chapter 8: A Genetic Algorithm Approach using an Indirect Representation

Chapter 7 discussed a genetic algorithm that uses a direct representation where each individual was a timetable. This chapter describes a genetic algorithm that uses an indirect representation (referred to as IGA) where each individual is a string of instructions that are implemented to build a timetable. Section 8.1 describes the overall algorithm for the IGA. Section 8.2 covers the initial population generation. Section 8.3 describes how each individual is evaluated. Section 8.4 and section 8.5 respectively cover the selection method and genetic operators used. Section 8.6 lists the control parameters used for the IGA.

8.1 Overall Algorithm

Similar to the DGA described in Chapter 7, the IGA approach also uses two phases. Phase 1 implements a genetic algorithm that evolves a population of strings where each string contains instructions that are used to build a timetable. The first phase of the IGA ends when a feasible timetable is built using the instructions of an individual string. Phase 2 then implements a genetic algorithm that evolves a population of strings containing instructions that improve the quality of the timetable found in Phase 1. The genetic algorithms in both phases use standard tournament selection and the genetic operators used are mutation and crossover. The algorithms for Phase 1 and Phase 2 differ in terms of the fitness function and the set of instructions used. The aim of Phase 1 is to produce a string that builds a feasible timetable while the aim of Phase 2 is to produce a string that best improves the quality of the feasible timetable found in Phase 1.

8.2 Initial Population Creation

This section describes the creation of the initial population. The representation used for the IGA is described in section 8.2.1. Section 8.2.2 outlines the algorithm for creating the initial population. Section 8.2.3 describes the sequential construction method (SCM) which is used to create an individual in the initial population. Finally, section 8.2.4 covers the creation of the initial population during Phase 2.

8.2.1 Instruction String Representation

Each individual in the population is a string consisting of a specified set of characters, each representing an instruction used to build or change a timetable. These characters are

randomly chosen and allocated to the string. The set of characters and its associated instruction are listed in the table below:

Table 8.1: Instructions used to build a timetable

Character	Instruction
A	Allocation – Add an unallocated tuple to the timetable.
D	De-allocation – Remove an allocated tuple from the timetable.
1	Mutation operator 1 (1VNH described in chapter 7).
2	Mutation operator 2 (2VNH described in chapter 7).
3	Mutation operator 3 (1VH described in chapter 7).
4	Mutation operator 4 (2VH described in chapter 7).
5	Phase 2 Mutation operator 1 – Random swap with hill climbing (Described in chapter 7).
6	Phase 2 Mutation operator 2 – Random row swap with hill climbing (described in chapter 7).
7	Phase 2 Mutation operator 3 – 1VH that searches for soft constraint violations.
8	Phase 2 Mutation operator 4 – Varies depending on problem. Discussed in results chapter.

In Table 8.1 above, the instructions labelled “1” to “8” are the mutation operators used by the DGA. In addition, the instruction labelled “A” is used to allocate a tuple to the timetable and the instruction labelled “D” is used to remove (de-allocate) a tuple from the timetable. The “A” and “D” instructions are based on the timetable builders used by Bufe et al. [BUFE01]. These timetable builders randomly selected tuples and allocated them to the timetable such that the hard constraint cost was minimized.

The length of the initial string is equal to the number of tuples that need to be allocated. Additionally, some characters have a better probability of being chosen than others. For example, it was found through observation that the allocation instruction (A) must have a greater probability of being selected than the de-allocation instruction (D) in order to produce a greater number of complete timetables.

An example of an instruction string is *ADA3DA4AA34DDDA*. In this string, the first instruction, the allocation instruction, allocates a tuple to a period in the timetable.

Thereafter, the de-allocation instruction will remove the tuple from the timetable (resulting in an empty timetable). The allocation instruction is then applied and an unallocated tuple is placed in the timetable. The next instruction to be called is mutation operator 3. This indicates the application of the 1VH operator described earlier in Chapter 7. The de-allocation instruction is then applied. The allocation instruction then allocates another tuple to the timetable. This is followed by the application of mutation operator 4 (2VH), the allocation of two more tuples (*AA*), the application of two more mutation operators (*34*), three de-allocation instructions (*DDD*) and finally, the placement of an unallocated tuple (*A*).

8.2.2 Algorithm for Initial Population Creation

Algorithm 8.1 lists the steps that are involved when creating the initial population for Phase 1. The variable *N* represents the population size and is a control parameter. The variable *M* indicates the SCM size and is also a control parameter. As can be seen in the algorithm below, Phase 1 ends if an instruction string is able to produce a feasible timetable in the initial population.

Algorithm 8.1: Initial Population Creation

```
Begin
    For LoopIP = 1 to N
        For LoopSCM = 1 to M
            Create HeuristicString
            Create timetable using HeuristicString
            Evaluate HeuristicString by evaluating timetable
            BestHeuristicString = HeuristicString that induces most fit timetable
            If Feasible Timetable Found
            {
                End Phase 1
                Feasible Timetable carried over to Phase 2
            }
        End Loop
        Add BestHeuristicString to initial population
    End Loop
End
```

8.2.3 The Sequential Construction Method (SCM)

The SCM is similar to the SCM incorporated by the DGA in Chapter 7. This method creates *M* instruction strings (*M* is a control parameter) and the string that produces the best timetable (evaluation discussed in section 8.3) is added to the population.

8.2.4 Phase 2 Initial Population Creation

The first feasible solution found is carried over to Phase 2. A population of strings are created with each string containing a combination of instructions that will be used to improve the quality of the timetable found in Phase 1. Similar to the initial population of Phase 2 in the DGA, the SCM was not implemented.

8.3 Evaluating an individual in the population

Each instruction string is evaluated by building a timetable. The fitness value of the individual is calculated by counting the number of hard constraint violations (Phase 1) and soft constraints violations (Phase 2) of the timetable produced.

The process of creating a timetable using the instruction string is shown in Algorithm 8.2. If an allocation instruction ("A") is executed, then the tuple with the lowest saturation degree (i.e. the tuple with the fewest number of violation free periods) is allocated to a period that does not violate any hard constraints (if possible). The saturation degree heuristic (described in Chapter 7) was chosen as it was found to be the best performing heuristic from the three that were tested using the DGA (see Chapter 9). If a de-allocation ("D") instruction is executed, then a randomly chosen tuple is removed from the timetable. If a mutation instruction is executed, then that specific mutation operator is executed on the timetable (see Table 8.1). If there are no hard constraint violations, then these instructions are not applied.

Algorithm 8.2: Creating a timetable

```
Begin
        For each character in heuristic string
        {
                If character = "A"
                        If all tuples are allocated
                                Do nothing
                        Else
                                Find tuple that is most difficult to allocate (Saturation degree)
                                Allocate tuple to timetable
                                Resort unallocated tuple list in order of difficulty
                Else If character = "D"
                        If timetable is empty
                                Do nothing
                        Else
                                Randomly select tuple from timetable
                                Remove tuple and return to unallocated tuple list
                Else If character = "1" or "2" or "3" or "4" (or "5" or "6" or "7" or "8")
                        If the hard (or soft) constraint cost = 0
                                Do nothing
                        Else
                                Apply associated mutation operator
        }
End
```

8.4 Selecting a parent

Parents are selected using standard tournament selection. This selection method was described in Chapter 4 (section 4.4.4).

8.5 Genetic Operators

This section describes the crossover and mutation genetic operators used in both phases of the IGA.

8.5.1 Mutation

Mutation is applied by randomly choosing a character from the string and replacing it with another randomly selected character from the instruction set. An example of the application of the mutation operator is shown below (Figure 8.1):

Parent (randomly selected character underlined):
ADADAAA341<u>2</u>12AD1A34AAAD

Randomly chosen character = "A"

Resultant offspring (Single character replaced)
ADADAAA341<u>A</u>12AD1A34AAAD

Figure 8.1: Mutation of heuristics string

In the example above, the 11th character of the string (2) has been randomly chosen. This character is replaced by another randomly selected character in the instruction set, in this case the "A" character which represents an allocation operator.

8.5.2 Crossover

The "Cut and Splice" crossover operator [GOLE89] is used and creates offspring with different string lengths. The use of variable string lengths increases the probability that the individual will have all the information necessary to produce an optimal solution ([MICH98]).

For this operator, a randomly chosen crossover point is selected for each of the two individuals. The points may differ between the selected individuals. Figure 8.2 below shows an example of two strings. The crossover points and affected characters (instructions) are shown in bold and are underlined.

Parent 1 (randomly selected character sets underlined)
ADADAAA341212AD**<u>1A34A</u>**
Parent 2 (randomly selected character sets underlined)
<u>DA31AAA43</u>21444ADADAD

Figure 8.2: Selection of crossover points

Offspring 1 is produced by taking the fragment before the crossover point of Parent 1 and exchanging it with the fragment before the crossover point of Parent 2. The second offspring is formed using the remaining instructions. The resultant crossover is shown below (Figure 8.3).

Resultant Individual 1 after crossover
ADADAAA341212AD21444ADADAD
Resultant Individual 2 after crossover
DA31AAA431A34A

Figure 8.3: Resultant offspring after crossover

The two offspring form part of a new generation of individuals. As can be seen in Figure 8.3, the crossover operator has also resulted in the length of the strings being changed.

8.6 Control Parameters

The parameters used for the IGA are the following:

- SCM Size.
- Population Size.
- Tournament Size.
- Number of swaps.
- Mutation Application Rate – Defined in Chapter 4 (section 4.4.5.4).
- Crossover Application Rate – Defined in Chapter 4 (section 4.4.5.4).
- Maximum number of generations.

8.7 Summary

This chapter describes a genetic algorithm that uses an indirect representation. The chapter begins by describing the overall IGA. This is followed by the description of how an individual is represented, how the initial population is generated for both phases of the IGA and how each individual is evaluated. The chapter then goes on to describe how selection occurs and the genetic operators used to create offspring. Finally the chapter covers the control parameters of the IGA.

Chapter 9 – Results and discussion

9.1 Introduction

This chapter discusses the results obtained when applying both the DGA and IGA approaches to the five school timetabling problems. Both approaches were able to induce feasible timetables and the quality of the timetables were competitive if not better than timetables produced using other methods. For the DGA, it was found that different methods of initial population generation, selection methods and mutation operators were needed in order to solve each school timetabling problem. When comparing the two approaches, the IGA performed better than the DGA.

Section 9.2 provides an analysis of the performance of the DGA when modifying the processes of the algorithm. The processes that will be changed include the method for initial population creation, the selection methods and the mutation operators used. Section 9.3 reports on the fine-tuning of the DGA parameters. Section 9.4 and section 9.5 covers the performance of the IGA when solving the five school timetabling problems. In section 9.6, timetables induced by the DGA and IGA are compared against timetables produced using other methods.

9.2 DGA process evaluation

This section reports on the effects of different processes used with the DGA when applied to each problem. The DGA is run using sets of different genetic algorithm processes, each of which have already been discussed in Chapter 7. Based on the performance of the DGA, a decision is taken as to which process to use.

9.2.1 The Abramson benchmark school timetabling problem (HDTT)

This section describes the performance of the DGA approach when applied to the HDTT school timetabling problem. Section 9.2.1.1 describes the performance of the DGA with different low-level construction heuristics for each data set. Section 9.2.1.2 discusses the best selection method for the different data sets and section 9.2.1.3 discusses the best mutation operator to use for each of the data sets. As this problem does not consider any soft constraints, Phase 2 is not implemented.

9.2.1.1 Comparison of low-level construction heuristics

One of three primary construction heuristics is used to construct the initial population. These heuristics are random allocation, largest degree heuristic and the saturation degree. If two

or more tuples have the same largest degree or saturation degree, then the random allocation heuristic is used as a tie breaker. Table 9.1 lists the other processes and parameter values that were used when testing the three primary construction heuristics.

Table 9.1: Processes and parameter values to test best low-level construction heuristic (HDTT problem)

Constant Methods and Operators	
Selection	VTS
Mutation	2 Violation Non Hill Climbing (2VNH)
Constant Parameter Values	
SCM	100
Population Size	1000
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates and average number of generations taken to find a solution are shown in Table 9.2 below. The average number of constraint violations and standard deviations are listed in Table 9.3. Based on the tables below, the DGA performed best when using the saturation degree heuristic.

Table 9.2: Performance comparison with different construction heuristics

	Success Rates				Average number of generations taken to find solution		
	Random	Largest degree	Saturation degree		Random	Largest degree	Saturation Degree
HDTT4	100.00%	100.00%	100.00%	HDTT4	3	2	1
HDTT5	100.00%	96.67%	100.00%	HDTT5	9	7	2
HDTT6	43.33%	53.33%	56.67%	HDTT6	21	18	7
HDTT7	20.00%	20.00%	20.00%	HDTT7	39	30	15
HDTT8	0.00%	3.33%	6.67%	HDTT8	50	47	20

In Table 9.2, the success rates for data sets HDTT4 and HDTT5 are very high with feasible timetables being found for almost every run. In addition, the feasible timetables are found within ten generations. The DGA using the saturation degree finds feasible timetables in at most two generations for data sets HDTT4 and HDTT5. These data sets have a relatively small number of resources and are not as constrained as the HDTT6, HDTT7 and HDTT8 data sets. Results using other methods (described in section 9.6.1) also show that feasible timetables were easily induced for these two data sets.

The success rate for the DGA drops as the number of resources increases for the problem (HDTT6, HDTT7 and HDTT8). The averages in Table 9.3 indicate that the DGA using the saturation degree produces timetables with very few constraint violations when compared to the DGA using the other construction heuristics. The saturation degree heuristic contributes to the low averages by producing a fitter initial population than when using the other two construction heuristics. The results listed below are similar to that of other methods used in the literature, thus indicating the difficulty of these data sets.

Table 9.3: Average constraint violations (and standard deviations) found for different construction heuristics

	Random	Largest Degree	Saturation Degree
HDTT4	0 (0)	0 (0)	0 (0)
HDTT5	0 (0)	0.07 (0.37)	0 (0)
HDTT6	1.13 (1.01)	0.93 (1.01)	0.87 (1.01)
HDTT7	1.60 (0.81)	1.60 (0.81)	1.63 (0.85)
HDTT8	15.07 (7.44)	4.07 (3.26)	2.00 (0.64)

A bar chart showing a comparison of the three heuristics is given in Figure 9.1 below. The x-axis represents the data sets that were used and the y-axis represents the success rates. The success rate is the percentage of runs that have produced feasible timetables. For example in data set HDTT5, the GA approach using the largest degree heuristic produces feasible timetables for 29 of the thirty runs performed. This results in a 96.67% success rate.

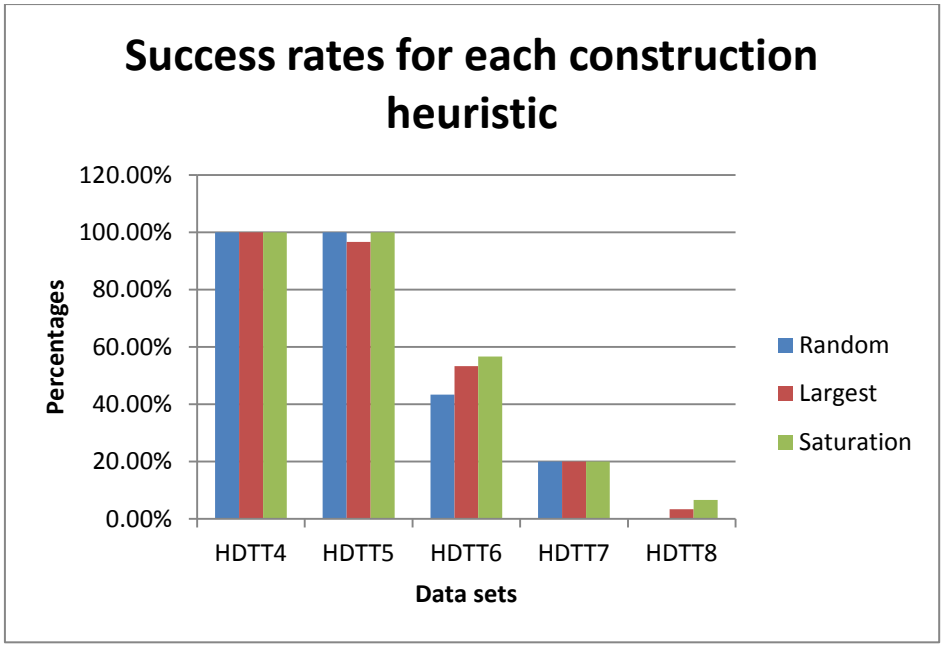


Figure 9.1: Comparison of success rates for each heuristic

Each heuristic, when used with the DGA, creates at least one clash free timetable for the data sets HDTT4, HDTT5, HDTT6 and HDTT7. The DGA using the random allocation heuristic does not produce any feasible solutions for the HDTT8 data set. The bar chart illustrates an inverse relationship between the success rate and the number of resources available for each data set. As the number of teachers, classes and venues increase, the data sets become more constrained. Thus, the difficulty in obtaining a solution increases.

Two hypotheses are tested to determine whether the saturation degree is statistically better than the random allocation and largest degree heuristics. The first hypothesis is that the saturation degree heuristic performs better than the random allocation heuristic and the second hypothesis is that the saturation degree performs better than the largest degree heuristic. The hypotheses and the corresponding Z-values are shown in Table 9.4.

Table 9.4: Hypotheses and Z-values for feasibility

Hypothesis	Z Values				
	HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
H_0: $\mu_{RA} = \mu_{SD}$; H_A: $\mu_{RD} > \mu_{SD}$	0.00	5.48	1.02	0.16	9.58
H_0: $\mu_{LD} = \mu_{SD}$; H_A: $\mu_{LD} > \mu_{SD}$	0.00	0.00	0.26	0.16	3.40

The table shows that the saturation degree heuristic is significantly better than the random allocation heuristic for data sets HDTT5 and HDTT8. This is significant at all levels (see Table 6.3 in Chapter 6). A Z-value of 3.4 indicates that the saturation degree performs significantly better than the largest degree heuristic for data set HDTT8.

Figure 9.2 shows the average number of generations that the DGA takes to find a solution when applying each of the three heuristics. The x-axis lists the data sets for the problem and the y-axis is the number of generations taken to find a feasible solution.

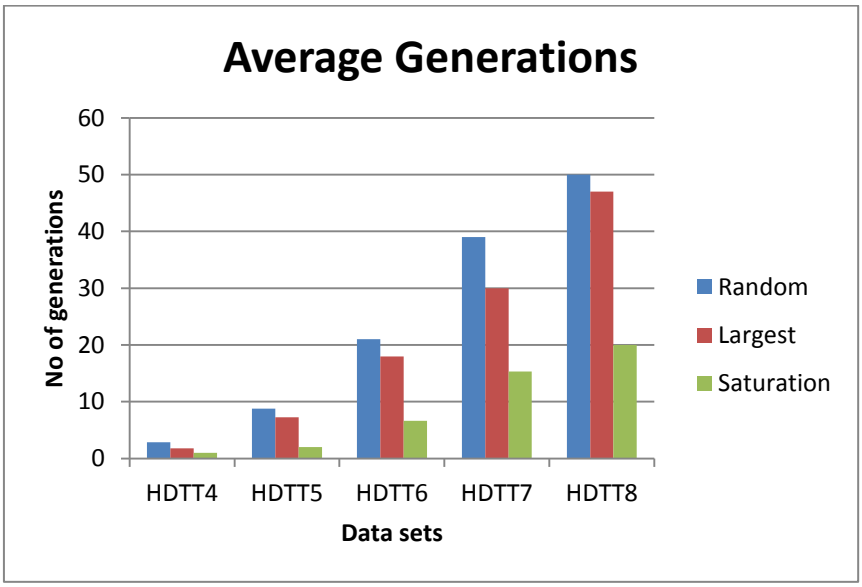


Figure 9.2: Average number of generations taken per data set

The DGA approach using the saturation degree produces solutions in the fewest number of generations. As was stated earlier, this is due to the ability of the saturation degree to produce an initial population of individuals with good fitness values. The disadvantage of the saturation degree is that the runtime taken to produce the initial population may be slower than the random and largest degree heuristics due to the processing overhead of the saturation degree when recalculating the number of violation free periods for each unallocated tuple. The column chart shows a trend that the GA approach will take more generations to produce feasible timetables if the resources of the problem increase.

The saturation degree heuristic is chosen as the best heuristic for this problem due to its higher success rate and its ability to produce solutions in fewer generations. The low averages and standard deviations listed in Table 9.3 also indicated that this heuristic was the most consistent in terms of producing the least number of constraint violations over the thirty runs conducted.

9.2.1.2 Comparison of selection methods

The two selection methods tested with the DGA are standard tournament selection and variant tournament selection. The other processes and parameter values used to test the selection methods are listed below in Table 9.5.

Table 9.5: Processes and parameter values to test best selection method (HDTT problem)

Constant Heuristics and Operators	
Creation	Saturation Degree
Mutation	2VNH
Constant Parameter Values	
SCM	100
Population Size	1000
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates and average number of generations taken to find solutions for each data set are listed in Table 9.6. The averages and standard deviations are listed in Table 9.7.

Table 9.6: Results comparison for selection methods used

	Success Rate			Average number of generations	
	Standard	Variant		Standard	Variant
HDTT4	100.00%	100.00%	HDTT4	1	1
HDTT5	93.33%	100.00%	HDTT5	1	2
HDTT6	33.33%	56.67%	HDTT6	3	7
HDTT7	10.00%	20.00%	HDTT7	8	15
HDTT8	6.67%	6.67%	HDTT8	12	20

Table 9.7: Average constraint violations and standard deviations for different selection methods

	Standard	Variant
HDTT4	0 (0)	0 (0)
HDTT5	0.13 (0.51)	0 (0)
HDTT6	1.33 (0.96)	0.87 (1.01)
HDTT7	1.93 (0.74)	1.63 (0.85)
HDTT8	2.23 (0.82)	2 (0.64)

Figure 9.3 and Figure 9.4 are bar chart illustrations of the results in Table 9.6. Figure 9.3 depicts the success rates found for each data set when using the two selection methods. The success rates are determined by dividing the total number of feasible timetables found with the total number of runs conducted per method. The DGA approach with either selection method produces a 100% success rate for the data set HDTT4.

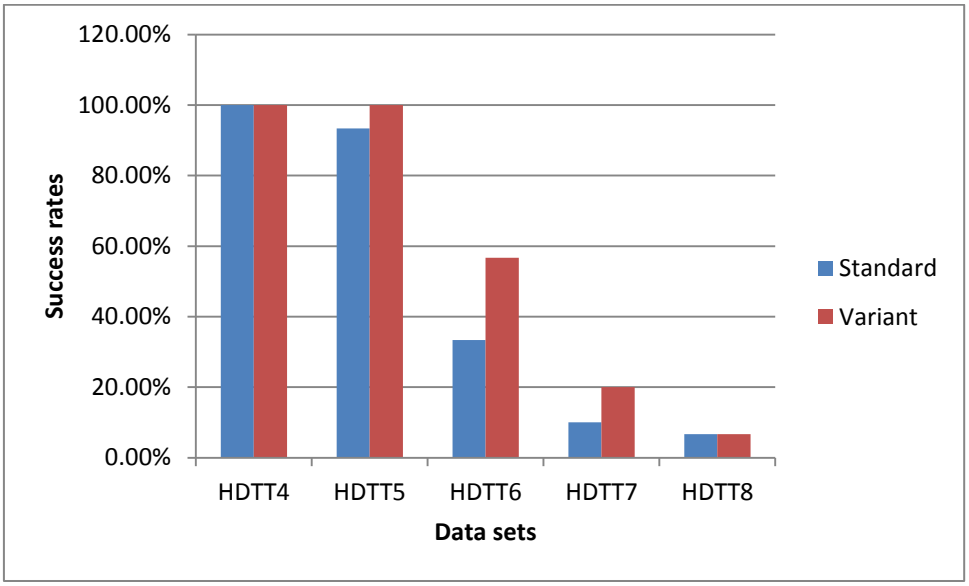


Figure 9.3: Comparison of success rates for various selection methods

Figure 9.3 illustrates that at least one clash free timetable is found for all data sets when using the DGA approach with either of the selection methods. The DGA using variant tournament selection performs better than standard tournament selection when applied to data sets HDTT5, HDTT6 and HDTT7. No difference is found between selection methods for data sets HDTT4 and HDTT8.

Figure 9.4 is a column chart showing the average number of generations required to find a solution. The x-axis represents the data sets and the y-axis lists the average number of generations between 0 and 25.

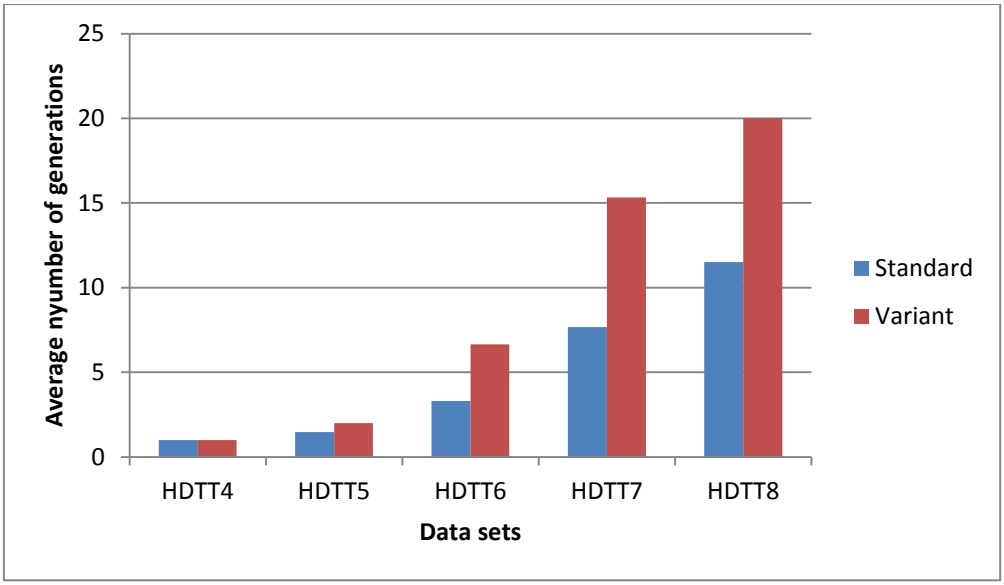


Figure 9.4: Column chart showing average generations for each selection method

The column chart shows that the DGA using standard tournament selection requires fewer generations to produce a clash free timetable than the DGA using variant tournament selection. Standard tournament selection is designed to always choose the best individual from the tournament while variant tournament selection is designed to provide an opportunity for weaker tournament participants to become parents as well. The disadvantage of standard tournament selection is that the elitist nature of the method results in the algorithm converging too quickly (to a local optimum) when the DGA is applied to data sets HDTT6, HDTT7 and HDTT8. The less elitist variant tournament selection allowed for a better exploration of the search space and thus better results are found.

A single hypothesis is used to test whether the DGA using variant tournament selection produces fewer violations than the DGA using standard tournament selection. Table 9.8 shows the hypothesis and the Z-values for each data set.

Table 9.8: Hypotheses and Z-values for timetable feasibility

Hypothesis	Z-Values				
	HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
H_0: $\mu_{std} = \mu_{var}$; H_A: $\mu_{std} > \mu_{var}$	0.00	1.44	1.84	1.46	1.23

Significant results are found for HDTT5, HDTT6 and HDTT7 at the 10% level while there is no statistically significant difference between using the standard or variant selection methods when applying the DGA to data sets HDTT4 and HDTT8.

The number of generations used varies between the data sets. The DGA approach using standard tournament selection has a lower success rate than the DGA approach using variant tournament selection but does manage to produce results within a fewer number of generations. However, due to a higher success rate for three out of the five data sets, the variant tournament selection method is chosen.

9.2.1.3 Comparison of mutation operators

In this study, the DGA is tested using one of four mutation operators described in Chapter 7. The first two mutation operators involve hill climbing along with either the one violation mutation operator (1VH) or the two violation mutation operator (2VH). The remaining two operators involve using either the one violation mutation operator (1VNH) or the two violation mutation operator (2VNH) without hill climbing. The processes and parameter values that were kept constant are listed in Table 9.9.

Table 9.9: Processes and parameter values to test best mutation operator (HDTT problem)

Constant Heuristics and Operators	
Creation	Saturation Degree
Selection	Variant
Constant Parameter Values	
SCM	100
Population Size	1000
Tournament Size	10
Swaps per mutation	20
Generations	50

A comparison of the performance of each of the operators (in terms of success rate) is shown in Table 9.10.

Table 9.10: Comparison of success rates for mutation operators

	SUCCESS RATE				Average number of generations			
	2VH	1VH	2VNH	1VNH	2VH	1VH	2VNH	1VNH
HDTT4	93.33%	76.67%	100.00%	0.00%	1	4	1	NA
HDTT5	16.67%	16.67%	100.00%	0.00%	3	8	2	NA
HDTT6	0.00%	0.00%	56.67%	0.00%	NA	NA	7	NA
HDTT7	0.00%	0.00%	20.00%	0.00%	NA	NA	16	NA
HDTT8	0.00%	0.00%	6.67%	0.00%	NA	NA	20	NA

Table 9.10 shows that the DGA using the 2VNH operator performs better than the DGA using any of the other mutation operators. The use of the DGA with the 2VNH operator results in at least one feasible timetable being produced when applied to any of the data sets. The DGA with the hill climbing mutation operators find feasible timetables for the smaller data sets HDTT4 and HDTT5 but not for the larger and more constrained data sets (HDTT6, HDTT7 and HDTT8).

The averages and standard deviations listed in Table 9.11 also indicate that the 2VNH mutation operator performs best. The DGA using the 1VNH operator performed worst and was not able to induce feasible timetables for even the smallest data set (HDTT4). The performance of the DGA using the 2VH and 1VH were similar and feasible timetables were found for the HDTT4 and HDTT5 data sets only.

Table 9.11: Average constraint violations (and standard deviations) for each mutation operator

	2VH	1VH	2VNH	1VNH
HDTT4	0.13 (0.51)	0.47 (0.86)	0 (0)	48.23 (1.96)
HDTT5	1.67 (0.76)	1.7 (0.79)	0 (0)	68.87 (2.11)
HDTT6	3.27 (0.78)	3.4 (0.81)	0.87 (1.01)	90.47 (1.98)
HDTT7	7.2 (1.06)	7.47 (1.11)	1.63 (0.85)	109.07 (4.63)
HDTT8	9.53 (1.48)	9.93 (1.57)	2 (0.64)	131.37 (3.62)

Three hypotheses are tested and the corresponding Z-values for each data set are listed in the table below (Table 9.12). The three hypotheses are:

- Two violation non hill climbing (2VNH) produces fewer violations than the one violation non hill climbing operators (1VNH).
- 2VNH produces fewer violations than the one violation hill climbing operator (1VH).
- 2VNH produces fewer violations than the two violation hill climbing operator (2VH).

Table 9.12: Hypotheses and corresponding Z-values showing feasibility

Hypotheses	Z-Values				
	HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
H_0: $\mu_{1VNH} = \mu_{2VNH}$; H_A: $\mu_{1VNH} > \mu_{2VNH}$	134.81	178.52	221.06	124.98	191.45
H_0: $\mu_{1VH} = \mu_{2VNH}$; H_A: $\mu_{1VH} > \mu_{2VNH}$	2.97	11.72	10.71	22.90	25.55
H_0: $\mu_{2VH} = \mu_{2VNH}$; H_A: $\mu_{2VH} > \mu_{2VNH}$	1.44	12.04	10.29	22.39	25.58

The DGA using the 2VNH mutation operator performs better than the DGA approach using either of the hill climbing operators. The hypothesis tests confirm these results at all levels of significance (see Table 6.3 in Chapter 6). The only exception is the comparison between the DGA approach using the 2VH and 2VNH operators when applied to data set HDTT4 where the 2VNH operator is better at a 10% level of significance only.

From the operators tested, the 2VNH operator is the best mutation operator as the genetic algorithm using this operator finds at least one feasible timetable for all data sets and performs better than the other mutation operators at significant levels of at least 10%.

9.2.2 The Valouxis Greek school timetabling problem

This section covers the performance of different processes of the DGA when applied to the Valouxis problem. The processes that are covered for this problem are the low-level construction heuristics (section 9.2.2.1), the Phase 1 selection method (section 9.2.2.2), the Phase 1 mutation operator (section 9.2.2.3), the Phase 2 selection method (section 9.2.2.4) and the Phase 2 mutation operator (section 9.2.2.4).

9.2.2.1 Comparison of low-level construction heuristics

As described in Chapter 7, one of three primary heuristics is used with the DGA, namely random allocation, largest degree and saturation degree. A secondary heuristic comparing the teacher degree (teachers with the most class-teacher lessons) is used in the event of ties. In the event of further ties, a third heuristic that compares teacher availability is used (teachers with the fewest days available are given priority).

In order to select the best of the three heuristics, the performance of each heuristic is compared. The processes and parameter values that were kept constant are listed in Table 9.13 below.

Table 9.13: Processes and parameter values to test best low-level construction heuristic (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Selection	Variant
Mutation	1 Violation Hill Climbing (1VH)
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates, average hard constraint (HC) violations, standard deviations, and average quality for each low-level construction heuristic are shown below in Table 9.14. The average quality is the average number of soft constraint violations from all the feasible timetables found.

Table 9.14: Performance comparison of construction heuristics

Success Rates		
Random	Largest Degree	Saturation
100.00%	86.67%	83.33%
Average HC violations (and standard deviations)		
Random	Largest Degree	Saturation
0 (0)	0.13 (0.35)	0.17 (0.38)
Average Quality (Feasible Timetables)		
Random	Largest Degree	Saturation
52.88	52.43	47.20

All three heuristics, when used separately with the DGA, produce high success rates of 80% and above. The DGA with the random allocation heuristic produces the most number of feasible timetables. The low averages and standard deviations indicate that for each run, feasible solutions were found or timetables with only one or two hard constraint violations were produced. In terms of quality, the DGA using the saturation degree performs best, producing timetables with five fewer soft constraint violations on average.

The results from the hypothesis tests support the above statement that the genetic algorithm using the random allocation heuristic produces timetables with the fewest number of hard constraint violations. The hypotheses and the Z-values are shown in Table 9.15.

Table 9.15: Hypotheses and corresponding Z-values

Hypothesis	Z-value – Feasibility	Z-value – Quality
H_0: $\mu_{SAT} = \mu_{RAND}$; H_A: $\mu_{SAT} > \mu_{RAND}$	2.41	2.33
H_0: $\mu_{SAT} = \mu_{LARGE}$; H_A: $\mu_{SAT} > \mu_{LARGE}$	0.36	0.30
H_0: $\mu_{LARGE} = \mu_{RAND}$; H_A: $\mu_{LARGE} > \mu_{RAND}$	2.11	2.12

Although the success rates for the random allocation and largest degree heuristics are higher, the DGA using the saturation degree heuristic produces better quality timetables i.e. timetables with fewer soft constraint violations. The frequency chart below (Figure 9.5) illustrates this. The x-axis of the frequency chart shows the average soft constraint cost over thirty runs and the y-axis lists the number of timetables produced for each range.

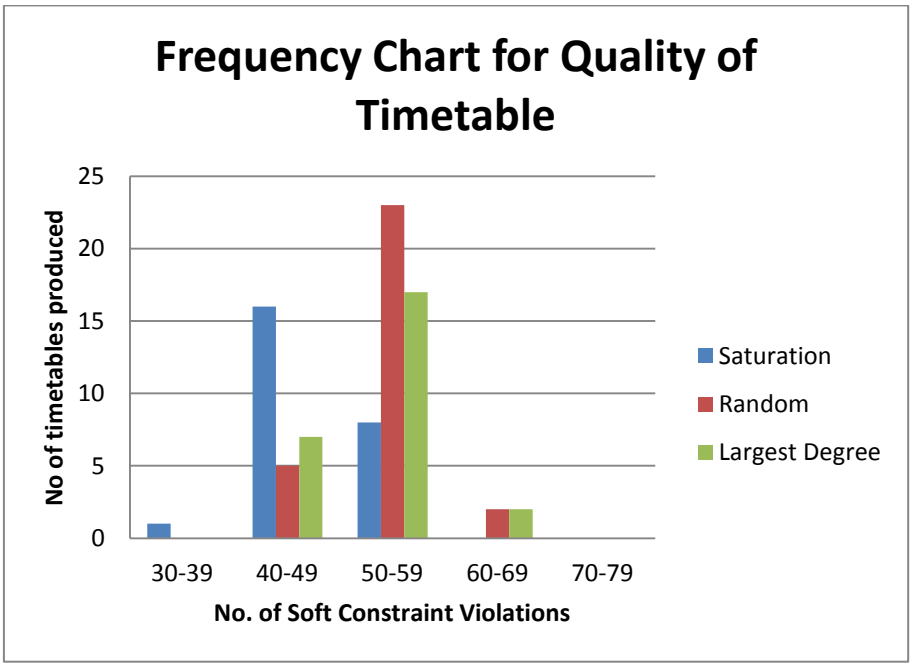


Figure 9.5: Frequency diagram for quality using various heuristics

Figure 9.5 illustrates that the DGA using the saturation degree heuristic produces 17 feasible timetables that each have between 30 and 50 soft constraint violations. While the DGA using the random allocation heuristic produces more feasible timetables, the DGA using the saturation degree produces more timetables of a higher quality.

In order to test the statistical significance of the above conclusion, an independent sample t-test is conducted. The results are summarized in the table below.

Table 9.16: Summary of results for independent sample t-test

Heuristic	N	Mean	Std Deviation	Std. Error Mean
Saturation	30	0.7333	0.44978	0.08212
Random	30	0.2667	0.44978	0.08212

The mean values in Table 9.16 indicate a greater preference towards the saturation degree heuristic. The calculated p-value is less than 0.05, concluding that at a 5% level of significance, the DGA using a saturation degree heuristic produces more timetables of a higher quality than the DGA using the random allocation heuristic.

In conclusion, while the success rates are in favour of the random allocation heuristic, there is a case for the saturation degree heuristic as it produces better quality timetables on average. For this problem, the saturation degree heuristic is therefore chosen as the low-level construction heuristic.

9.2.2.2 Comparison of Phase 1 selection methods

The DGA is tested using standard tournament selection and variant tournament selection. When testing these two methods, the following processes and parameter values remained constant (Table 9.17):

Table 9.17: Processes and parameter values to test best selection method (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates, average hard constraint cost, standard deviations and average quality (soft constraint cost) for the DGA using each selection method is shown in Table 9.18 below.

Table 9.18: Results for selection methods (Phase 1)

Success Rates	
Standard	Variant
70.00%	83.33%
Average HC Cost (and standard deviations)	
Standard	Variant
0.3 (0.47)	0.17 (0.38)
Average Quality (Feasible Timetables)	
Standard	Variant
53.19	47.20

A success rate of at least 70% is found over thirty runs when the DGA is used with either of the selection methods. Variant tournament selection allows for a greater exploration of the search space as it also allows weaker individuals from the tournament to be selected as parents. The DGA with the variant tournament selection also produces better quality timetables on average than the DGA with standard tournament selection.

From Table 9.18, it is concluded that the DGA using variant tournament selection performs better than the DGA using standard tournament selection. Hypothesis tests are conducted

to determine the statistical significance of the above statement. The Z-values for both feasibility and quality are listed in Table 9.19 below.

Table 9.19: Hypotheses and Z-values for feasibility and quality

Hypotheses	Z-values
H_0: $\mu_{STD} = \mu_{VAR}$; HA: $\mu_{STD} > \mu_{VAR}$ (Feasibility)	1.22
H_0: $\mu_{STD} = \mu_{VAR}$; HA: $\mu_{STD} > \mu_{VAR}$ (Quality)	1.26

Based on the results of the hypothesis tests (Table 9.19), there is insufficient evidence to state that the DGA using variant tournament selection is statistically better than the DGA using standard tournament selection.

Figure 9.6 illustrates a frequency chart showing the quality of the timetables produced when using the DGA with standard tournament selection and variant tournament selection.

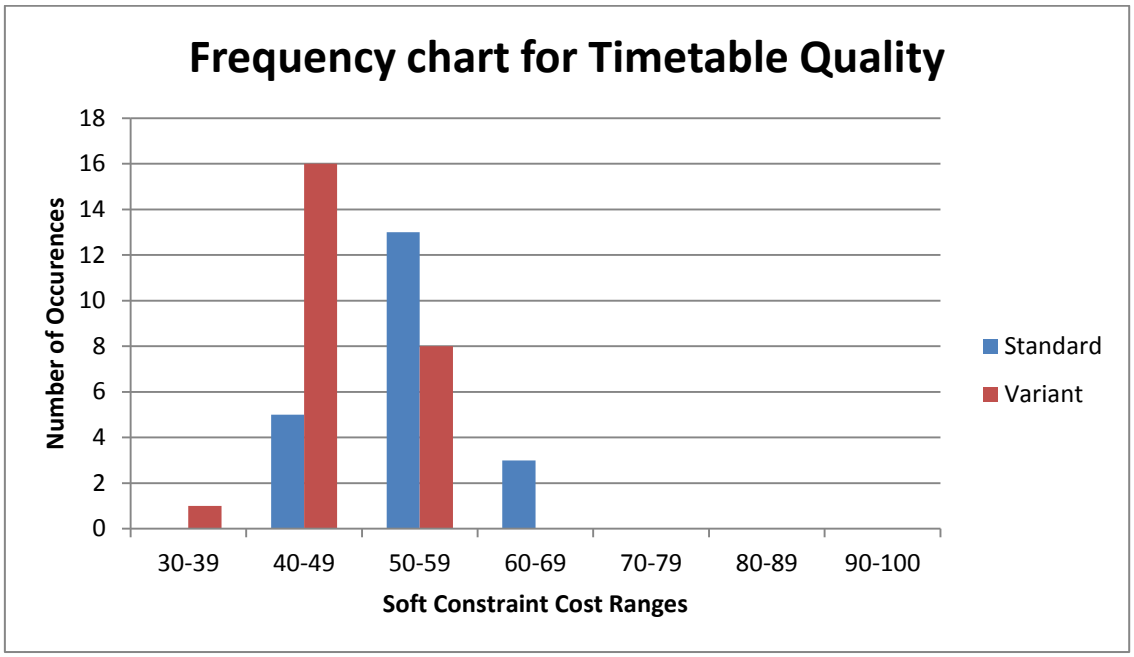


Figure 9.6: Frequency chart showing quality for selection methods

The frequency chart above indicates that the DGA with variant tournament selection produces more timetables of a higher quality than the DGA using standard tournament selection. 68% of the feasible timetables produced by the DGA with variant tournament selection had between 30 and 50 soft constraint violations compared to the DGA using standard tournament selection where only 24% of the feasible timetables produced had soft constraint violation counts of between 30 and 50. Variant tournament selection is the chosen selection method as it produces a higher success rate and better quality timetables

on average. The DGA using variant tournament selection also produces the timetable with the fewest number of soft constraint violations (39).

9.2.2.3 Comparison of Phase 1 mutation operators

Four mutation operators are tested. As discussed in the previous chapter, the four mutation operators, 2VNH, 2VH, 1VNH and 1VH, are tested separately with the DGA approach. The other processes and control parameter values that were kept constant are given in Table 9.20 and a summary of the results for each mutation operator is shown in Table 9.21.

Table 9.20: Processes and parameter values to test best mutation operator (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

Table 9.21: Results for mutation operators (Phase 1)

Success Rates		Success Rates	
2VH	1VH	2VNH	1VNH
0.00%	83.33%	0.00%	0.00%
Average HC Cost (and standard deviation)		Average HC Cost (and standard deviation)	
2VH	1VH	2VNH	1VNH
7 (0.93)	0.17 (0.38)	66 (0.45)	77 (4.17)
Average Quality (Feasible Timetables)		Average Quality (Feasible Timetables)	
2VH	1VH	2VNH	1VNH
NA	47.20	NA	NA

Table 9.21 above indicates that only the DGA using the 1VH mutation operator manages to produce feasible timetables. The poor performance of the 2VNH and 1VNH operators emphasizes the importance of hill climbing for this problem. The DGA using the non-hill

climbing operators averaged 66 and 67 constraints violations respectively. It was determined that these mutation operators, when removing a constraint violation, caused another hard constraint violation in the timetable. This is especially true for this problem where one of the hard constraints is a balance in teacher lessons throughout the week (other school timetabling problems have listed this as a soft constraint). For example, the mutation operator moves a constraint violating tuple to a period that results in another teacher's workload being over the specified limit for that day. In the case of the non-hill climbing mutation operators, this is made worse in that swaps that increase the timetable cost are retained. The DGA approach using the other mutation operators do not produce any solutions and thus the best mutation operator is the 1VH mutation operator. The 1VH operator found tuples that resulted in constraint violations and swapped them with any randomly chosen tuple. Hill climbing assisted in only accepting swaps that improved the timetable while rejecting swaps that increased the hard constraint cost.

9.2.2.4 Comparison of Phase 2 selection methods

Standard tournament selection and variant tournament selection are implemented separately with the DGA approach and the performance of each method is compared. The processes and control parameter values used are shown below.

Table 9.22: Processes and parameter values to test best Phase 2 selection method (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates and average quality obtained after thirty runs per algorithm are shown in Table 9.23.

Table 9.23: Results for selection methods (Phase 2)

Success rate	
Standard	Variant
83.33%	83.33%
Average Quality (Feasible Timetables)	
Standard	Variant
50.04	47.20
Standard deviation - Quality	
Standard	Variant
5.89	4.90

During Phase 2, the DGA aims to improve the quality of the feasible timetables produced from Phase 1. Therefore, the success rates for both methods are the same and the quality of solutions using the DGA with either selection method needs to be compared. Table 9.23 indicates that the DGA using variant tournament selection (47.20) performs better than standard tournament selection DGA (50.04). A hypothesis stating that variant tournament selection is better than standard tournament selection is tested. A Z-value of 0.03 was found, indicating that there is no statistically significant difference between the performances of the two selection methods.

The frequency of results obtained using the two methods also need to be observed. The frequency chart below (Figure 9.7) shows the difference between the DGA using standard tournament selection and the DGA using variant tournament selection (VTS).

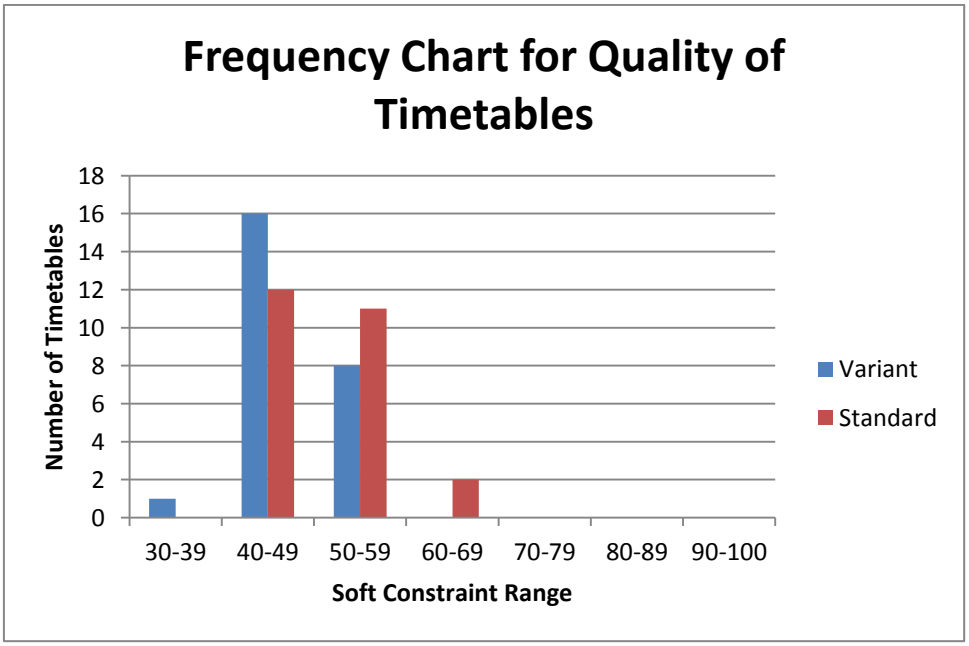


Figure 9.7: Frequency chart showing quality for selection methods

Figure 9.7 illustrates that the DGA using variant tournament selection produces more timetables in the range between 30 and 50 (68% of the timetables produced fall in this range) than the DGA using standard tournament selection (48% of the timetables produced fall in this range). The best selection method is therefore variant tournament selection due to its ability to produce better quality timetables. The variant tournament selection DGA also produces the timetable with the least number of soft constraint violations (39).

9.2.2.5 Comparison of Phase 2 mutation operators

In order to improve the quality of timetables, four soft constraint mutation operators are individually applied with the DGA. The performance of each DGA using a different mutation operator is then compared. The processes and parameter values used are listed in Table 9.24 below.

Table 9.24: Processes and parameter values to test best Phase 2 mutation operator (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The average quality and best timetables for each mutation operator after thirty runs is listed in Table 9.25.

Table 9.25: Results for mutation operators (Phase 2)

Average Quality (Feasible Timetables)			
Random Swap	Row Swap	2V	1V
47.20	73.24	47.00	65.08
Best Results			
Random Swap	Row Swap	2V	1V
39	58	36	54
Standard deviation – Quality			
Random Swap	Row Swap	2V	1V
4.90	10.43	4.80	6.88

Based on the table above, the two best mutation operators are the random swap mutation operator and the two violation (2V) mutation operator. The DGA with these mutation operators produce average soft constraint violation costs of 47.20 and 47.00 respectively. The row swap operator performed the worst due to multiple tuples of a row being swapped resulting in more soft constraint violations (such as the morning afternoon preference) and hard constraint violations (teachers being moved to days that they are unavailable to teach).

Hypothesis tests are conducted to determine the significance of the results. The hypotheses are focused on comparing the random swap operator and the 2V operator to the other mutation operators as well as each other. Five hypotheses are tested and are listed below:

- Random Swap (RaS) produces better quality timetables than 1V.
- Two Violation (2V) swapping produces better quality timetables than 1V.
- RaS produces better quality timetables than RS.
- 2V produces better quality timetables than RS.
- 2V produces better quality timetables than RaS.

The hypotheses and the resultant Z-values found are shown in the table below (Table 9.26).

Table 9.26: Hypotheses and Z-values for quality

Hypotheses	**Z-values**
H_0: $\mu_{1V} = \mu_{RaS}$; H_A: $\mu_{1V} > \mu_{RaS}$	11.60
H_0: $\mu_{1V} = \mu_{2V}$; H_A: $\mu_{1V} > \mu_{2V}$	11.81
H_0: $\mu_{RS} = \mu_{RaS}$; H_A: $\mu_{RS} > \mu_{RaS}$	12.38
H_0: $\mu_{RS} = \mu_{2V}$; H_A: $\mu_{RS} > \mu_{2V}$	12.52
H_0: $\mu_{RaS} = \mu_{2V}$; H_A: $\mu_{RaS} > \mu_{2V}$	0.16

The Z-values listed in Table 9.26 indicate that the random swap and 2V mutation operators perform significantly better than the row swap and 1V mutation operators. There is no significant difference between the performance of the random swap and the 2V mutation operators. The frequency chart for the two operators is shown below.

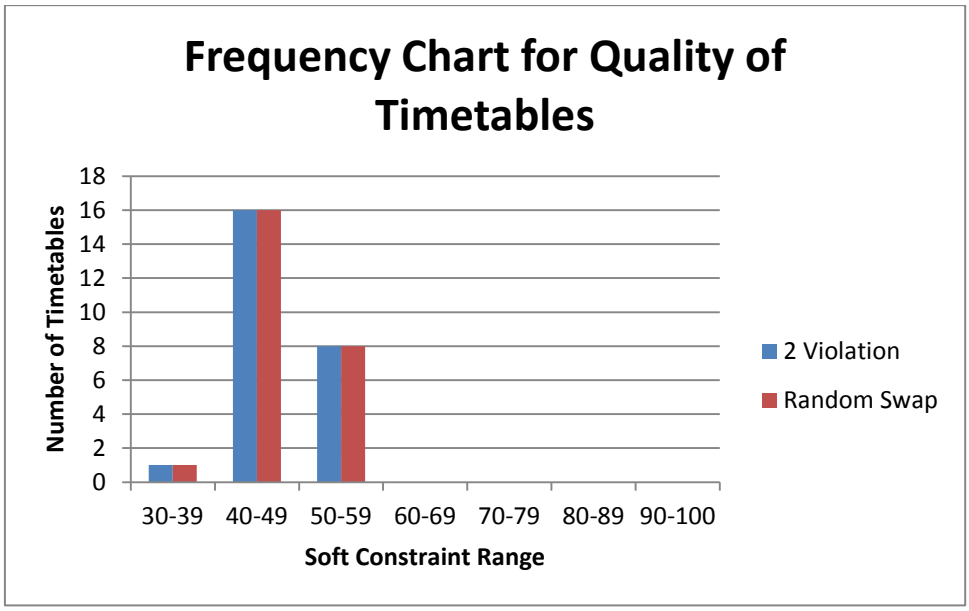


Figure 9.8: Frequency chart showing quality for two mutation operators

Figure 9.8 illustrates that the frequencies between the random swap operator and the 2V operator are the same. This further confirms the hypothesis that there is very little difference between the two mutation operators in terms of performance. The DGA approach with the 2V mutation operator produced the best timetable with 36 soft constraint violations compared to the DGA using the random swap operator (39 soft constraints violations). While either mutation operator could be applied to produce quality timetables, the two violation (2V) mutation operator is chosen as it produced a timetable of a higher quality (fewer soft constraint violations).

9.2.3 The Beligiannis Greek high school timetabling problem

This section describes the performance of the DGA when applied to the Beligiannis Greek high school timetabling problem. The first section outlines the performance of the DGA approach with different low-level construction heuristics. Section 9.2.3.2 discusses the performance of the two selection methods used with the DGA during Phase 1. Section 9.2.3.3 reports on the performance of the DGA when using different Phase 1 mutation operators. Section 9.2.3.4 describes the performance of the DGA when applying two different selection methods during Phase 2. Finally, section 9.2.3.5 explains the

performance of the DGA when using different mutation operators that focus on improving the quality of timetables.

9.2.3.1 Comparison of low-level construction heuristics

One of three low-level construction heuristics is applied with the DGA. These heuristics are the saturation degree, the largest degree and the random allocation heuristics. For all three heuristics, all tuples involving co-teaching and subclasses are allocated to the timetable first (primary heuristic). The remaining class-teacher tuples are allocated thereafter. Table 9.27 lists the common processes and parameter values used when testing each of the construction heuristics.

Table 9.27: Processes and parameter values to test best construction heuristic (Beligianis problem)

Constant Methods and Operators	
Phase 1	
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

Table 9.28 shows the success rates found for each heuristic when applied to the data sets.

Table 9.28: Results for different heuristics

Success Rates			
	Saturation	Random	Largest Degree
HS1	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%
HS5	3.33%	56.67%	76.67%
HS7	100.00%	100.00%	100.00%

A bar chart depicting the success rates listed in Table 9.28 is shown below (Figure 9.9).

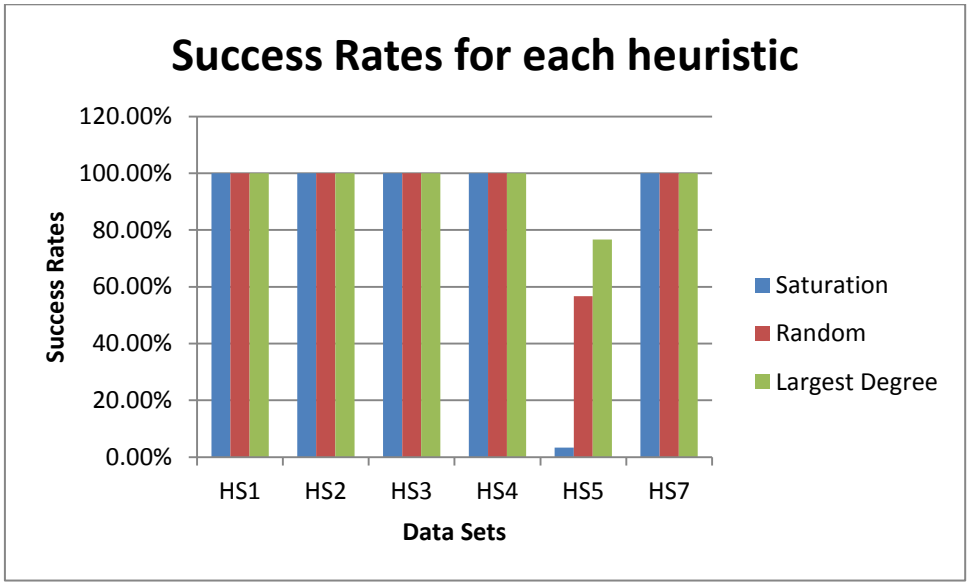


Figure 9.9: Success rates for each heuristic

Figure 9.9 illustrates that feasible timetables are found for all runs when the DGA (using any heuristic) is applied to data sets HS1, HS2, HS3, HS4 and HS7.

For the data set HS5, the DGA using the largest degree heuristic performs best, producing the most feasible timetables. No reason could be found as to why the results of HS5 differed to that of the other data sets. Hypothesis tests are performed to check the significance of this result. Two hypotheses are tested for feasibility for the data set HS5. These hypotheses are:

- The largest degree (LD) produces fewer hard constraint violations than the random allocation heuristic (RA).
- LD produces fewer hard constraint violations than the saturation degree heuristic (SD).

The hypotheses and Z-values are listed in Table 9.29.

Table 9.29: Hypotheses and Z-values for feasibility for data set HS5

Hypotheses	Z-values
H_0: $\mu_{RA} = \mu_{LD}$; H_A: $\mu_{RA} > \mu_{LD}$	1.65
H_0: $\mu_{SD} = \mu_{LD}$; H_A: $\mu_{SD} > \mu_{LD}$	12.27

From the Z-values in Table 9.29, it is statistically confirmed that the DGA using the largest degree heuristic performs better than the DGA using saturation degree or random allocation (for data set HS5).

In order to select the best heuristic for the remaining data sets (HS1, HS2, HS3, HS4 and HS7), the quality of the timetables produced needs to be assessed since the DGA produces feasible timetables for all runs. Table 9.30 lists both the average number of soft constraint violations and the standard deviations found for each data set when the DGA using each of the three heuristics is applied.

Table 9.30: Average quality for each heuristic

Average Soft Constraint Violations (and standard deviations)			
	Saturation	Random	Largest Degree
HS1	120.20 (3.75)	154.37 (6.54)	143.00 (3.91)
HS2	129.60 (3.79)	160.47 (5.02)	147.33 (6.84)
HS3	47.37 (4.13)	59.80 (4.52)	51.33 (5.47)
HS4	72.57 (3.82)	83.50 (5.86)	83.40 (4.73)
HS5	52.00 (NA)	52.29 (6.89)	46.35 (6.26)
HS7	143.23 (4.31)	184.37 (5.71)	159.60 (4.88)

A bar chart depicting the average quality for each data set is shown below. The x-axis represents each data set and the y-axis lists the average number of soft constraint violations per data set. As mentioned in earlier chapters, a lower soft constraint cost indicates a higher quality timetable.

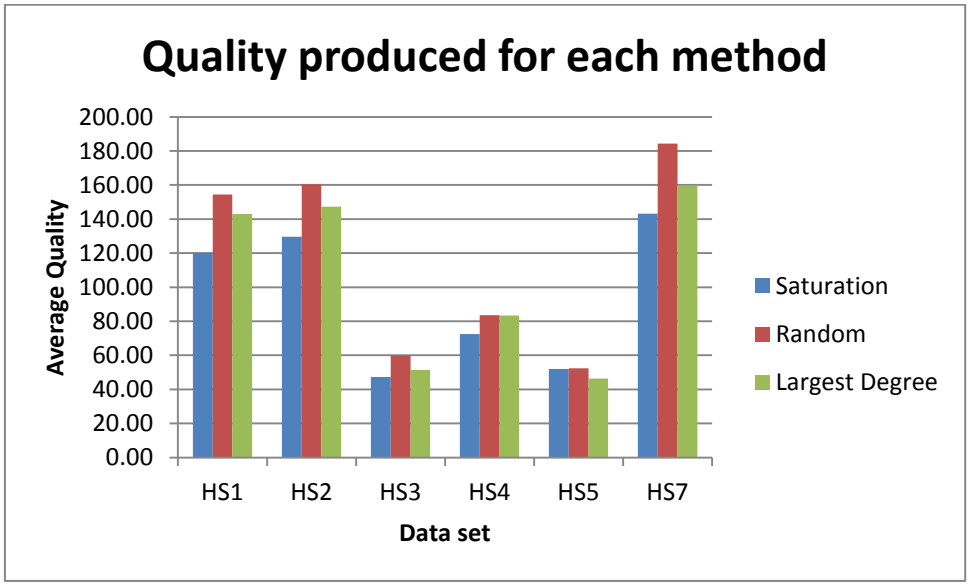


Figure 9.10: Bar chart showing quality for each heuristic

Figure 9.10 shows that, with the exception of data set HS5, the DGA with the saturation degree heuristic produces the best quality timetables. The relatively low standard deviations for the saturation degree also indicate that for all thirty runs, the genetic algorithm using this construction heuristic consistently produced timetables very close to the average quality timetable. Hypothesis tests are conducted to determine if the average quality of the DGA using the saturation degree is significantly better than the DGA using the other two low-level heuristics. The hypotheses tested are:

- Saturation degree (SD) produces fewer soft constraint violations than random allocation (RA).
- Saturation degree (SD) produces fewer soft constraint violations than largest degree (LD).

The Z-values for the above hypotheses for data sets HS1, HS2, HS3, HS4, and HS7 are shown in Table 9.31.

Table 9.31: Hypotheses and Z-values for quality for various data sets

Hypotheses	Z-values				
	HS1	**HS2**	**HS3**	**HS4**	**HS7**
H_0: $\mu_{RA} = \mu_{SD}$; H_A: $\mu_{RA} > \mu_{SD}$	8.17	8.49	6.53	0.07	18.05
H_0: $\mu_{LD} = \mu_{SD}$; H_A: $\mu_{LD} > \mu_{SD}$	23.03	12.41	3.17	9.76	13.76

The Z-values found and listed in Table 9.31 confirms that the DGA using saturation degree produces better quality timetables than the DGA using either of the other two heuristics. The Z-values further confirm the results illustrated in Figure 9.10. Significant results are found on all levels (1%, 5% and 10%). For data set HS4, A Z-value of 0.07 indicates that there is no difference between using the saturation degree and the random allocation heuristic to improve timetable quality.

Due to the high quality of timetables obtained for data sets HS1, HS2, HS3, HS4, and HS7, the best timetable construction heuristic for these data sets is the saturation degree. For the data set HS5, the largest degree is the chosen heuristic used to create the initial population.

9.2.3.2 Comparison of Phase 1 selection methods

The two selection methods available for Phase 1 are standard tournament selection and variant tournament selection. When testing the performance of these two selection methods, the following processes and parameter values remained constant.

Table 9.32: Processes and parameter values to test best selection method (Beligianis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for data set HS5)
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates found for each data set are shown in the table below. Figure 9.11 is a bar graph of the data in Table 9.33.

Table 9.33: Success rates for selection methods (Phase 1)

Success Rates		
	Standard	Variant
HS1	100.00%	100.00%
HS2	100.00%	100.00%
HS3	100.00%	100.00%
HS4	100.00%	100.00%
HS5	36.67%	76.67%
HS7	100.00%	100.00%

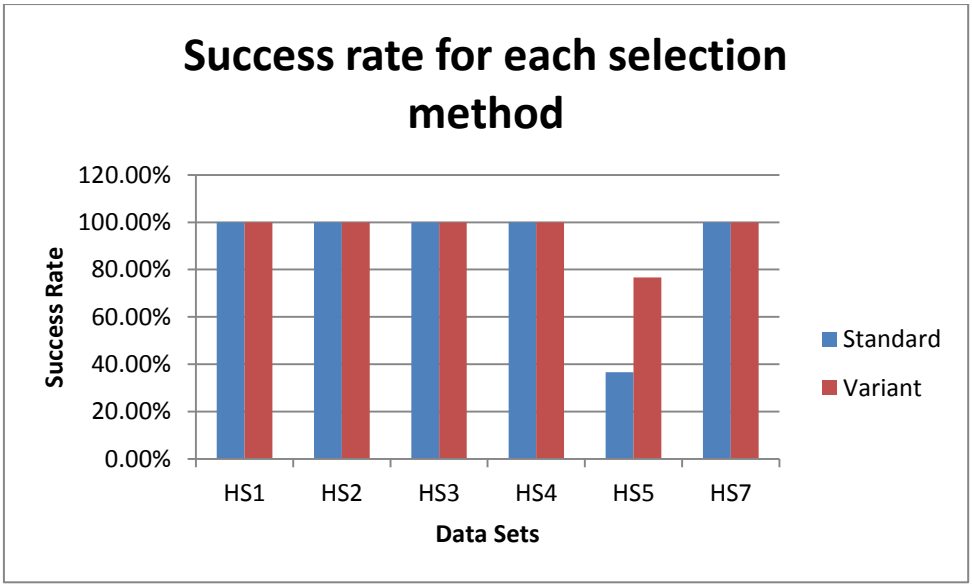


Figure 9.11: Success rates for selection methods

The DGA approach using either of the selection methods produces feasible timetables for all runs when applied to data sets HS1, HS2, HS3, HS4 and HS7. For the data set HS5, variant tournament selection is the best selection method due to this DGA producing more feasible timetables over the thirty runs. When testing this conclusion using a hypothesis test, a Z value of 3.38 indicates that the DGA using variant tournament selection performs better than the DGA using standard tournament selection. Table 9.34 shows the average quality and standard deviations for each data set when applying the DGA approach with each of the selection methods.

Table 9.34: Average quality for each selection method (Phase 1)

Average Soft Constraint Violations (and standard deviations)		
	Standard	Variant
HS1	122.73 (5.51)	120.20 (3.75)
HS2	131.07 (3.89)	129.60 (3.79)
HS3	47.47 (3.95)	47.37 (4.13)
HS4	73.03 (4.05)	72.57 (3.82)
HS5	50.18 (6.37)	46.35 (6.26)
HS7	143.93 (4.87)	143.23 (4.31)

The DGA using variant tournament selection produces better quality timetables for all data sets. Hypothesis tests are used to determine the significance of this conclusion. The hypothesis being tested is that the DGA approach using VTS produces better quality timetables than the DGA approach using standard tournament selection. The Z-values for each data set are shown in Table 9.35.

Table 9.35: Hypothesis and Z-values for quality for various data sets

Hypothesis	Z-values					
	HS1	**HS2**	**HS3**	**HS4**	**HS5**	**HS7**
H_0: $\mu_{STD} = \mu_{VAR}$; H_A: $\mu_{STD} > \mu_{VAR}$	2.08	1.48	0.10	0.46	3.38	0.59

The hypothesis stating that the DGA using variant tournament selection produces better quality timetables than the DGA using standard tournament selection is statistically significant for data sets HS1 (5% and 10% significance levels), HS2 (10% significant level), and HS5 (all levels). Z values of 0.10 for data set HS3, 0.46 for data set HS4 and 0.59 for data set HS7 indicate that there is no difference between the performances of the selection methods when applied to these data sets. Variant tournament selection is chosen as the selection method to choose parents.

9.2.3.3 Comparison of Phase 1 mutation operators

Mutation is used to evolve the timetables and reduce the hard constraint cost of each timetable. With the exception of data set HS5, it is determined that the evolutionary process is not needed to produce feasible timetables. Many of the timetables created using the saturation degree heuristic are already feasible. Any mutation operator could be used to successfully remove the remaining hard constraint violations from unfeasible timetables.

For completeness, the success rates found when applying the DGA approach with each mutation operator are shown in Table 9.37. Table 9.36 below shows the processes and parameter values used to test each mutation operator.

Table 9.36: Processes and parameter values to test best mutation operator (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for data set HS5)
Selection	Variant
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

100% success rates are produced when the DGA is applied to all data sets except data set HS5.

Table 9.37: Success rates for mutation operators (Phase 1)

Success Rates				
	2VH	1VH	2VNH	1VNH
HS1	100.00%	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%	100.00%
HS5	0.00%	76.67%	0.00%	100.00%
HS7	100.00%	100.00%	100.00%	100.00%

When applied to data set HS5, the DGA approach with the 1VNH mutation operator produces the most feasible timetables when compared to the success rates using other mutation operators. A hypothesis test is conducted for feasibility for the data set HS5. The hypothesis states that the DGA using the 1VNH operator performs better than the DGA approach using the 1VH operator. A Z value of 2.08 indicates that the null hypothesis is rejected in favour of the alternate hypothesis at 5% and 10% levels of significance. When the DGA is applied to the data set HS5, the 1VNH operator is chosen as the best mutation operator due to a higher success rate. As mentioned earlier, a genetic algorithm is not required in order to produce feasible timetables for the other data sets.

9.2.3.4 Comparison of Phase 2 selection methods

Standard tournament selection and variant tournament selection are utilized as selection methods for Phase 2 of the algorithm. The timetables obtained when applying the DGA approach with either of the two methods are evaluated and compared. Phase 2 of the DGA approach focuses on reducing the soft constraint cost of the timetables. Therefore, the quality of the timetables produced is evaluated rather than the success rates.

Table 9.38: Processes and parameter values to test best Phase 2 selection method (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for data set HS5)
Selection	Variant
Mutation	1VH (1VNH for data set HS5)
Phase 2	
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The average qualities and standard deviations of the timetables produced by the DGA using each of the selection methods are listed below.

Table 9.39: Tabling comparing quality produced by selection methods (Phase 2)

Average Quality (and standard deviations)		
	Standard	Variant
HS1	115.67 (5.99)	120.20 (3.75)
HS2	121.63 (7.54)	129.60 (3.79)
HS3	46.53 (4.07)	47.37 (4.13)
HS4	73.30 (4.77)	72.57 (3.82)
HS5	55.53 (7.97)	59.00 (5.28)
HS7	138.97 (6.23)	143.23 (4.31)

The average qualities in Table 9.39 indicate that the DGA using standard tournament selection produces better quality timetables than the DGA using variant tournament selection. The DGA using variant tournament selection only performs better when applied to

data set HS4. A hypothesis is tested to determine if the DGA using variant tournament selection performs better than the DGA using standard tournament selection when applied to data set HS4. A Z-value of 0.66 indicates that there is no statistically significant difference in performance.

Hypothesis tests are also used to test for quality for data sets HS1, HS2, HS3, HS5, and HS7. The hypothesis states that the DGA using the standard tournament selection produces better quality timetables than the DGA using variant tournament selection. Table 9.40 shows the Z-values of the tests.

Table 9.40: Hypothesis and Z-values for quality

Hypothesis	Z-values				
	HS1	HS2	HS3	HS5	HS7
H_0: $\mu_{VAR} = \mu_{STD}$; H_A: $\mu_{VAR} > \mu_{STD}$	3.51	5.17	0.79	1.99	3.06

Based on the Z-values in the table above, it is concluded that for data sets HS1, HS2 and HS7, the DGA using standard tournament selection performs better than the DGA using variant tournament selection (at all levels of significance). For data set HS5, the alternate hypothesis is favoured at 5% and 10% levels of significance. When testing data sets HS3 for quality, a Z-value of 0.79 suggests that there is no difference in performance between the two selection methods.

In conclusion, the best method of selection during Phase 2 is standard tournament selection as it produces better quality timetables when the DGA is applied to five out of the six data sets.

9.2.3.5 Comparison of Phase 2 mutation operators

At this stage, the main objective of the DGA is to improve the quality of the feasible timetables obtained from Phase 1. To evolve the timetables, four mutation operators are considered, namely random swap, row swap, one violation mutation and two violation mutation. When testing each of these mutation operators, the following processes and parameter values were used (Table 9.41):

Table 9.41: Processes and parameter values to test best Phase 2 mutation operator (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for data set HS5)
Selection	Variant
Mutation	1VH (1VNH for data set HS5)
Phase 2	
Selection	Standard
Constant Parameter Values	
SCM	10
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The table below shows the average quality and standard deviations produced over thirty runs when applying the DGA with each operator.

Table 9.42: Average Quality and standard deviations obtained for different mutation operators (Phase 2)

Average Soft Constraint Violations (and standard deviations)				
	Random Swap	Row Swap	1V	2V
HS1	115.67 (5.99)	159.27 (4.96)	115.10 (7.02)	117.70 (6.30)
HS2	121.63 (7.54)	173.20 (5.03)	122.87 (8.34)	123.13 (7.60)
HS3	46.53 (4.07)	68.73 (2.46)	48.00 (3.99)	47.90 (4.45)
HS4	73.30 (4.77)	91.03 (3.10)	74.27 (4.28)	73.23 (4.72)
HS5	55.53 (7.97)	105.37 (5.90)	57.43 (7.95)	66.37 (5.96)
HS7	138.97 (6.23)	181.33 (6.98)	138.13 (7.44)	140.13 (6.49)

The row swap mutation operator is immediately removed as a candidate for best mutation operator as the DGA with this operator produces the worst quality timetables on average. This is due to many of the swaps being rejected as they resulted in moving teachers to days when they are unavailable to teach. Hypothesis tests also find that the row swap operator performs the worst at all levels of significance.

For data set HS1, the DGA using the 1V operator performs slightly better than the DGA approach using the random swap operator. For data sets HS2, HS3 and HS5, the DGA approach using the random swap operator provides the best quality timetables. For data set

HS4, the DGA approach using the 2V operator produces the best quality timetables. For data set HS7, the DGA approach using the 1V operator performs best. For data set HS5, the DGA approach using the random swap operator produces timetables with the lowest number of constraint violations on average. Hypothesis tests are conducted for quality for all the data sets. The hypotheses and corresponding Z-values for each data set are listed in Table 9.43.

Table 9.43: Hypotheses and Z-values

Data set HS1		
H_0: $\mu_{2V} = \mu_{RaS}$; H_A: $\mu_{2V} > \mu_{RaS}$	H_0: $\mu_{2V} = \mu_{1V}$; H_A: $\mu_{2V} > \mu_{1V}$	H_0: $\mu_{RaS} = \mu_{1V}$; H_A: $\mu_{RaS} > \mu_{1V}$
1.28	1.51	0.33
Data set HS2		
H_0: $\mu_{2V} = \mu_{1V}$; H_A: $\mu_{2V} > \mu_{1V}$	H_0: $\mu_{2V} = \mu_{RaS}$; H_A: $\mu_{2V} > \mu_{RaS}$	H_0: $\mu_{1V} = \mu_{RaS}$; H_A: $\mu_{1V} > \mu_{RaS}$
0.13	0.77	0.60
Data set HS3		
H_0: $\mu_{1V} = \mu_{2V}$; H_A: $\mu_{1V} > \mu_{2V}$	H_0: $\mu_{1V} = \mu_{RaS}$; H_A: $\mu_{1V} > \mu_{RaS}$	H_0: $\mu_{2V} = \mu_{RaS}$; H_A: $\mu_{2V} > \mu_{RaS}$
0.10	1.41	1.24
Data set HS4		
H_0: $\mu_{1V} = \mu_{RaS}$; H_A: $\mu_{1V} > \mu_{RaS}$	H_0: $\mu_{1V} = \mu_{2V}$; H_A: $\mu_{1V} > \mu_{2V}$	H_0: $\mu_{RaS} = \mu_{2V}$; H_A: $\mu_{RaS} > \mu_{2V}$
0.83	0.89	0.05
Data set HS5		
H_0: $\mu_{2V} = \mu_{1V}$; H_A: $\mu_{2V} > \mu_{1V}$	H_0: $\mu_{2V} = \mu_{RaS}$; H_A: $\mu_{2V} > \mu_{RaS}$	H_0: $\mu_{1V} = \mu_{RaS}$; H_A: $\mu_{1V} > \mu_{RaS}$
4.92	5.96	0.92
Data set HS7		
H_0: $\mu_{2V} = \mu_{RaS}$; H_A: $\mu_{2V} > \mu_{RaS}$	H_0: $\mu_{2V} = \mu_{1V}$; H_A: $\mu_{2V} > \mu_{1V}$	H_0: $\mu_{RaS} = \mu_{1V}$; H_A: $\mu_{RaS} > \mu_{1V}$
0.71	1.11	0.47

The best mutation operators vary depending on the data set. For data set HS1, the random swap and 1V operators are found to have performed significantly better than the two violation mutation operator (10% level of significance). There is, however, no significant

difference between these two mutation operators when applied to HS1. For data sets HS2, HS4 and HS7, no statistically significant difference is found when comparing the performance of the three operators. From the Z-values for data set HS5, a conclusion is made favouring the use of either the 1V mutation operator or the random swap operator. The Z-values for data set HS3 allows for the conclusion that the random swap operator performs better than the one violation operator at a 10% level of significance.

Based on the results above, it is concluded that the best mutation operator varies depending on the data set. For the remainder of the runs, the one violation (1V) mutation operator will be used when applying the DGA to data sets HS1, HS2, HS3, HS4 and HS7. The random swap operator will be used when applying the DGA to data set HS5.

9.2.4 W.A. Lewitt primary school timetabling problem

This section discusses the performance of the DGA approach when applied to the W.A. Lewitt primary school timetabling problem. Section 9.2.4.1 describes how the DGA performed with different low-level construction heuristics. Section 9.2.4.2 reports on the performance of the DGA using each of the two selection methods. Section 9.2.4.3 describes the performance of the DGA when using different mutation operators. Sections 9.2.4.4 and section 9.2.4.5 details the Phase 2 performance of the DGA in terms of selection method and mutation operators.

9.2.4.1 Comparison of low-level construction heuristics

The DGA uses one of three low-level construction heuristics. These heuristics are the saturation degree, the random allocation and the largest degree heuristic described in Chapter 7. In order to satisfy the double period constraint specified by the school, all tuples that are set as double periods are first allocated to the timetable.

Table 9.44: Processes and parameter values to test best construction heuristic (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Selection	Variant
Mutation	Hybrid mutation
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	200
Generations	50

The results of using the DGA approach with each of the three construction heuristics are shown in Table 9.45. These are used as secondary heuristics with the consecutive periods heuristic as the primary heuristic.

Table 9.45: Results for best heuristic

Success Rates		
Saturation	Random	Largest Degree
40.00%	6.67%	10.00%
Average Hard Constraint Violations (and standard deviations)		
Saturation	Random	Largest Degree
0.73 (0.69)	2.13 (1.2)	2.37 (1.27)
Average Quality (Feasible Timetables)		
Saturation	Random	Largest Degree
12.25	12.00	9.67
Standard deviations (Quality)		
Saturation	Random	Largest Degree
5.38	1.41	3.79

Table 9.45 shows that the DGA using the saturation degree heuristic produces the largest number of feasible timetables. On average, the DGA using the saturation degree produces the fewest number of hard constraint violations. An observation made was that the fitness of the initial populations of the DGA when using the random allocation and largest degree

heuristics were very high. The evolutionary process was then unable to induce feasible timetables.

Hypothesis tests are conducted for feasibility. The hypotheses tested and the Z-values are shown in Table 9.46.

Table 9.46: Hypotheses and Z-values for feasibility

Hypothesis	Z-value
H_0: $\mu_{LD} = \mu_{SD}$; H_A: $\mu_{LD} > \mu_{SD}$	6.07
H_0: $\mu_{RA} = \mu_{SD}$; H_A: $\mu_{RA} > \mu_{SD}$	5.55

The Z-values in the table above allow for the null hypothesis to be rejected in favour of the alternate hypothesis at all levels of significance. This means that the DGA using the saturation degree heuristic performs better than the DGA using either the random allocation or largest degree heuristics. The DGA using the saturation degree heuristic also produces the best quality timetable with the fewest number of soft constraint violations (7).

9.2.4.2 Comparison of Phase 1 selection methods

The two selection methods are variant tournament selection and standard tournament selection. The constant processes and parameter values used to test the two selection methods are listed in Table 9.47.

Table 9.47: Processes and parameter values to test best selection method (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Mutation	Hybrid mutation
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	200
Generations	50

The success rate, average quality and standard deviations produced is listed in Table 9.48 and the frequency diagram is shown in Figure 9.12.

Table 9.48: Success rates and average quality for selection methods (Phase 1)

Success Rates	
Variant	Standard
40.00%	43.33%
Average Quality (Feasible timetables)	
Variant	Standard
12.25	10.54
Standard deviation	
Variant	Standard
5.38	3.36

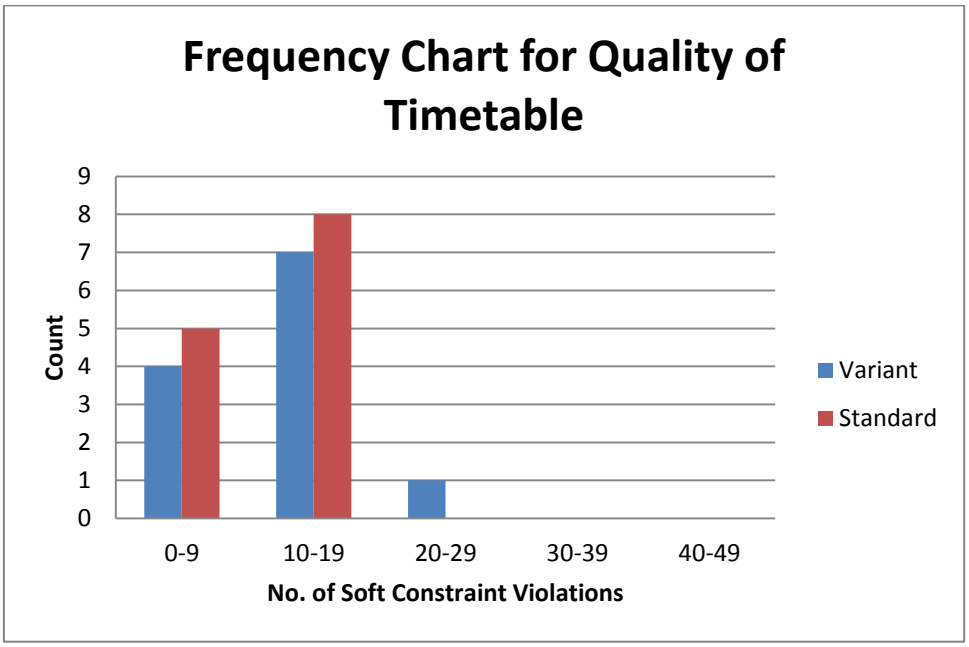


Figure 9.12: Frequency chart showing quality for two selection methods

Table 9.48 indicates that the DGA using standard tournament selection produces a slightly higher success rate and better quality timetables than the DGA approach using variant tournament selection. The standard deviation also indicates that the quality of the timetables produced tend to be close to the mean. The frequency chart (Figure 9.12) shows that more quality timetables are produced when using the DGA with standard tournament selection. A hypothesis is tested in order to determine the better selection method. The hypothesis states that the DGA using standard tournament selection performs better than the DGA using variant tournament selection method (in terms of feasibility). A Z-value of 0.17

indicates that there is no statistically significant difference in the performance of the two selection methods.

Standard tournament selection is chosen as the selection method due to its slightly higher success rate in producing feasible timetables. The frequency chart also indicates that DGA using standard tournament selection produces more timetables of better quality than when using variant tournament selection.

9.2.4.3 Comparison of Phase 1 mutation operators

Four mutation operators were considered and the DGA was applied using each of these mutation operators. From all the runs conducted, only one feasible timetable was produced. The low success rates produced indicate that the mutation operators are not sufficient in producing feasible timetables. The complexity of the problem and the large number of double periods to be allocated contribute to the poor performance of the mutation operators. In addition, all tuples must be allocated to all available periods. An alternative mutation operator was considered incorporating a combination of 2VH, 1VH and a random swap.

Table 9.49: Processes and parameter values to test best mutation operator (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	200
Generations	50

The success rates and average hard constraint costs of the DGA using each of the mutation operators as well as the hybrid operator are listed in Table 9.50.

Table 9.50: Success rates using different genetic operators (Phase 1)

Success Rates				
2VH	1VH	2VNH	1VNH	Hybrid
0.00%	3.33%	0.00%	0.00%	43.33%
Average Hard Constraint Cost (and standard deviation)				
2VH	1VH	2VNH	1VNH	Hybrid
4.57 (1.43)	2.97 (1.07)	16.4 (4.69)	110.67 (13.92)	0.77 (0.82)

The success rates listed in Table 9.50 indicate that the DGA using the hybrid operator is far superior to the DGA using any of the other mutation operators. When using the DGA approach with either the 2VH or 1VH operators, the average number of hard constraint violations ranges from 3 to 5, while the average number of violations when using either the 2VNH or 1VNH operators ranges from 16 (2VNH) to 111 (1VNH). Two factors that contributed to the success of the hybrid operator are the fact that three operators were being applied, thus increasing the probability of finding and removing a constraint violation. This also increased the number of swaps by a factor of three. A third factor is the mix of finding constraint violating tuples and random tuples, thus potentially increasing the probability of moving to a new area of the search space.

The number of feasible timetables produced by the DGA using the hybrid mutation operator was greater than the single feasible timetable produced by the DGA using the 1VH operator. A hypothesis is tested for feasibility. This hypothesis states that the DGA using the hybrid mutation operator performs better than the DGA using the 1VH operator. A Z-value of 8.97 indicates that at all levels of significance, the performance of the DGA using the hybrid operator is better than the DGA using the 1VH operator.

9.2.4.4 Comparison of Phase 2 selection methods

Phase 2 of the DGA focuses on improving the quality of the timetables. The DGA is used with either variant tournament selection or standard tournament selection. Table 9.51 lists the processes and parameter values used when testing each selection method.

Table 9.51: Processes and parameter values to test best Phase 2 selection method (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid mutation
Phase 2	
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	200
Generations	50

When analyzing the results, no difference in performance is found when comparing the performance of the DGA with the two selection methods. The average quality of the timetables produced is exactly the same. According to the frequency diagram below (Figure 9.13), the DGA using either selection method produces timetables of an equivalent quality. The DGA using either selection method produces the same number of timetables that have between 0 and 9 (and between 10 and 19) soft constraint violations.

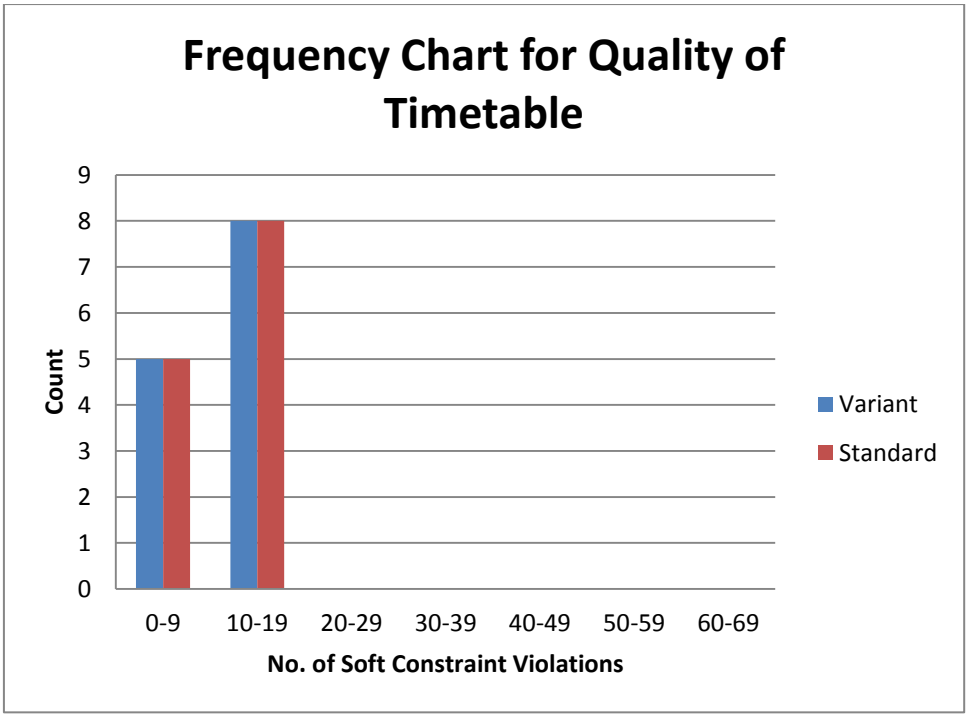


Figure 9.13: Frequency chart for two selection methods

Thus, either selection method can be used for Phase 2. Hypothesis tests are not conducted as the average qualities of the timetables produced when using either method is the same.

9.2.4.5 Comparison of Phase 2 mutation operators

In order to improve the quality of the timetables produced from Phase 1, the DGA uses one of four mutation operators during Phase 2. These operators are the random swap, 1V mutation, 2V mutation and the row swap operators. The row swap mutation operator was immediately not considered since this mutation was found to conflict with the hard constraint regarding double periods. The processes and parameter values used when testing the mutation operators are listed below.

Table 9.52: Processes and parameter values to test best Phase 2 mutation operator (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid mutation
Phase 2	
Selection	Variant
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	200
Generations	50

The average quality of the DGA when using each mutation operator is listed in the table below (Table 9.53)

Table 9.53: Average quality produced using different soft mutation operators

Average Quality		
Random Swap	1V	2V
10.54	15.62	24.31

It is evident from the table above that the DGA using the random swap operator produces the best quality timetables. The frequency diagram (Figure 9.14) also illustrates that the random swap produces better quality timetables.

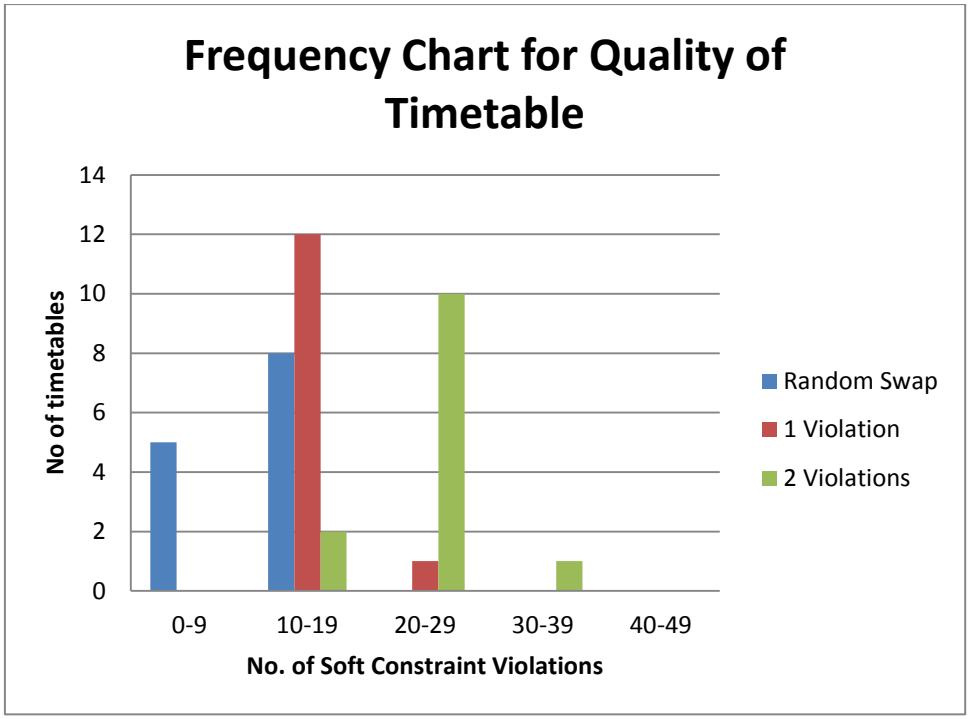


Figure 9.14: Frequency chart showing quality for different soft mutation operators

The DGA using the random swap operator produces timetables that have between 6 and 15 soft constraint violations. Hypothesis tests are conducted for quality and two hypotheses are established. The first hypothesis states that the DGA using the random swap performs better than the DGA using the 1V operator. The second hypothesis states that the DGA using the random swap performs better than the DGA using the two violation (2V) operator. Z-values of 0.05 and 0.02 respectively indicate that there is no significant difference when choosing between the mutation operators. While not statistically significant, the random swap is the chosen mutation operator to use when addressing soft constraint violations. The DGA using this operator produces better quality timetables on average.

9.2.5 The Woodlands secondary school timetabling problem

This section describes the performance of the DGA when applied to the Woodlands secondary school timetabling problem. Section 9.2.5.1 reports on the performance of the DGA with different low-level construction heuristics. Sections 9.2.5.2 and 9.2.5.4 discusses the results when comparing the DGA using two different selection methods for Phase 1 and Phase 2 respectively. Sections 9.2.5.3 and 9.2.5.5 cover the performance of the DGA using the different mutation operators for Phase 1 and Phase 2 respectively.

9.2.5.1 Comparison of low-level construction heuristics

Three low-level construction heuristics are individually used to create an individual in the initial population. The three construction heuristics are the saturation degree, random

allocation and largest degree. For the largest degree and saturation degree heuristics, several secondary heuristics are used for selecting tuples in the event of ties. These secondary heuristics, in order of importance are:

- Co-teaching and subclass requirements take precedence over single class tuples.
- In the event that two tuples are both single class tuples or co-teaching or subclass tuples, then priority goes to the tuple with the greater number of lessons for the week.
- In the event that two tuples have the same number of lessons in the week, then priority goes to the tuples with the greater number of teachers involved for that tuple. This secondary heuristic only applies to co-teaching tuples.
- In the event that the two tuples use the same number of teachers, then priority goes to the tuple with the class that are taught by the greater number of teachers throughout the week.

The secondary heuristics were determined based on observation. Only the DGA using the saturation degree heuristic was able to find feasible timetables.

Table 9.54: Processes and parameter values to test best construction heuristic (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	75
Generations	50

The success rates and average hard constraint cost found for the DGA approach using each heuristic is shown in the table below (Table 9.55).

Table 9.55: Success rates found for different heuristics

Success Rates		
Random	Largest Degree	Saturation
0.00%	0.00%	23.33%
Average Hard Constraint Cost (and standard deviations)		
Random	Largest Degree	Saturation
72.93 (8.99)	49.24 (7.43)	2.07 (1.62)

The DGA using the saturation degree heuristic produced seven feasible timetables from the 30 runs. On average, there are 9.41 soft constraint violations found per timetable. The DGA using random allocation creates timetables with approximately 72 hard constraint violations (on average) and the DGA using the largest degree heuristic creates timetables with approximately 49 hard constraint violations (on average). The best performing timetable construction heuristic is therefore the saturation degree. This construction heuristic creates timetable with a low number of hard constraint violations, allowing for the mutation operators to evolve the individuals into feasible timetables. The DGA using the largest degree or random allocation heuristics creates initial population individuals with too many hard constraint violations. The mutation operators were not able to remove all the violations in the required number of generations.

9.2.5.2 Comparison of Phase 1 selection methods

A DGA using standard tournament selection and variant tournament selection are individually implemented and the results are compared. When testing the performance of the two selection methods, the following processes and parameter values were used:

Table 9.56: Processes and parameter values to test best selection method (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	75
Generations	50

The success rates and average quality of timetables obtained are listed in the table below (Table 9.57).

Table 9.57: Results obtained using two different selection methods (Phase 1)

Success Rate	
Standard	Variant
40.00%	23.33%
Average Quality	
Standard	Variant
8.50	9.43
Standard deviation	
Standard	Variant
1.51	2.23

The DGA approach using standard tournament selection produces more feasible timetables. The timetables produced are also of a better quality. Standard tournament selection is therefore chosen as the best selection method when choosing parents. When conducting a hypothesis test for feasibility, a Z-value of 1.39 indicates that the DGA using standard tournament selection performs better than the DGA using variant tournament selection at a 10% level of significance. The vastly superior success rate and better quality allows for the conclusion that standard tournament selection be used to select parents during Phase 1.

9.2.5.3 Comparison of Phase 1 mutation operators

Four mutation operators are used separately with the DGA. These operators are the 2VH, 1VH, 2VNH and 1VNH operators discussed in Chapter 7. While testing these operators, the processes and parameter values listed in Table 9.58 were used.

Table 9.58: Processes and parameter values to test best mutation operator (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	75
Generations	50

The success rates and average quality found when using the DGA with each of the four mutation operators are shown below.

Table 9.59: Results obtained using different mutation operators (Phase 1)

Success Rate			
2VH	1VH	2VNH	1VNH
23.33%	40.00%	0.00%	0.00%
Average HC Violations (and standard deviations)			
2VH	1VH	2VNH	1VNH
2.27 (1.8)	1.80 (1.85)	49.13 (33.69)	900.07 (112.8)
Average Quality			
2VH	1VH	2VNH	1VNH
8.14	8.50	NA	NA

The success rates listed in Table 9.59 indicate that the DGA using the 2VNH or the 1VNH operators do not produce any feasible timetables. The DGA did find feasible timetables when using the 1VH or 2VH operators. The large difference in the average number of hard constraint violations indicates the importance of hill climbing when addressing this school

timetabling problem. Similar to the Lewitt and Abramson problem, the Woodlands problem requires that all tuples are allocated to all available periods with no free periods. With the addition of subclass and co-teaching requirements that involve several classes and teachers, the probability of clashes when swapping increases (thus the need for hill climbing). The DGA using the 1VH operator produces more feasible timetables while the DGA using the 2VH operator produces better quality timetables. Hypothesis tests are conducted for feasibility. The hypothesis tested is that the DGA using the 1VH operator performs better than the DGA using the 2VH operator. A Z-value of 0.99 is calculated, indicating that there is not enough evidence to suggest that there is a statistically significant difference in performance between the two mutation operators. The 1VH operator is chosen as the preferred mutation operator due to the high success rate obtained.

9.2.5.4 Comparison of Phase 2 selection methods

For Phase 2, standard tournament selection and variant tournament selection are considered when selecting parents.

Table 9.60: Processes and parameter values to test best Phase 2 selection method (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	1VH
Phase 2	
Mutation	Random swap
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	75
Generations	50

The calculated success rates and average qualities are exactly the same and can be seen in Table 9.61 below. Therefore, either selection method can be used.

Table 9.61: Results using two selection methods (Phase 2)

Success Rate	
Standard	Variant
40.00%	40.00%
Average Quality	
Standard	Variant
8.50	8.50

9.2.5.5 Comparison of Phase 2 mutation operators

Four mutation operators are used during Phase 2 to improve the quality of the feasible timetables. The four mutation operators are the random swap, 1V mutation, 2V mutation and the row swap. The 2V and 1V mutation operators are not considered as these mutation operators had difficulty in addressing subclass and co-teaching violations (tuples were often swapped with non-subclass or non-coteaching tuples, resulting in this constraint not being satisfied).

Two new mutation operators were considered; with both operators being variations of the 1V mutation operator. The first new operator finds a single soft constraint violation. If the violation found involves subclass violations, then the entire row is swapped rather than just the cells involved. This mutation is referred to as 1 violation subclass row swap (1VSRS). The second new operator is a one violation row mutation where the entire row of a cell containing any soft constraint violation is swapped with a randomly chosen row. This mutation operator is referred to as the 1 violation row swap (1VR). The use of the row swap was justified as swapping of rows rather than specific tuples prevents the possibility of clashes and prevents tuples involved in subclass or co-teaching requirements from being separated. Table 9.62 lists the processes and parameter values used when testing each mutation operator.

Table 9.62: Processes and parameter values to test best Phase 2 mutation operator (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid mutation
Phase 2	
Selection	Standard
Constant Parameter Values	
SCM	10
Population Size	500
Tournament Size	10
Swaps per mutation	75
Generations	50

The average qualities found for the DGA approach using each mutation operator is listed in the table below (Table 9.63).

Table 9.63: Results obtained using proposed soft mutation operators (Phase 2)

Average Quality (Feasible Timetables)			
Random Swap (RaS)	Row swap (RS)	1VSRS	1VR
8.50	6.92	5.00	6.92
Standard deviation			
Random Swap (RaS)	Row swap (RS)	1VSRS	1VR
1.51	0.79	1.41	0.79

The DGA using the 1VSRS produces the best quality timetables with an average of five soft constraint violations per feasible timetable produced. The frequency distribution (Figure 9.15) showing the number of quality timetables produced also indicates that this mutation operator performs better than the other operators. From all the feasible timetables produced, the DGA using this mutation operator produces the best timetable with three soft constraint violations. Based on the quality obtained and the frequency chart shown below, the best mutation operator is the 1VSRS mutation where rows are swapped only when a co-teaching/subclass violation is found.

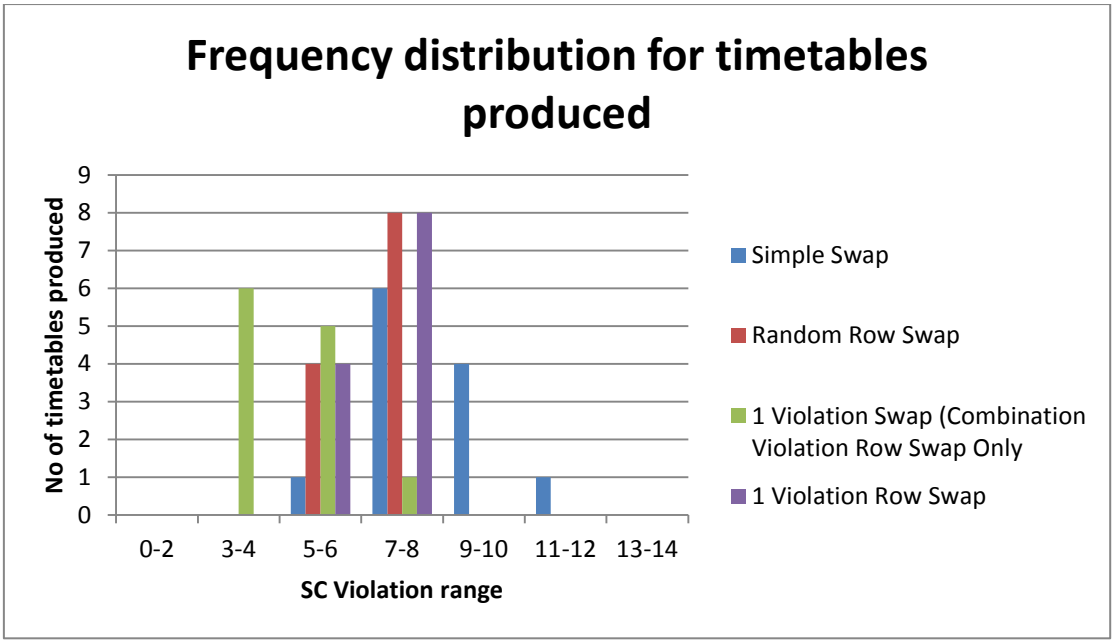


Figure 9.15: Frequency chart showing quality for proposed soft mutation operators

Hypothesis tests are conducted for quality and three hypotheses are tested. The Hypotheses are:

- The DGA using the 1VSRS operator performs better than the DGA using the RaS
- The DGA using the 1VSRS performs better than the DGA using RS
- The DGA using the 1VSRS performs better than the DGA using 1VR

The hypotheses and corresponding Z-values are listed in Table 9.64.

Table 9.64: Hypotheses and Z-values for quality

Hypothesis	Z-value
H_0: $\mu_{RaS} = \mu_{1VSRS}$; H_A: $\mu_{RaS} > \mu_{1VSRS}$	0.01
H_0: $\mu_{RS} = \mu_{1VSRS}$; H_A: $\mu_{RS} > \mu_{1VSRS}$	0.01
H_0: $\mu_{1VR} = \mu_{1VSRS}$; H_A: $\mu_{1VR} > \mu_{1VSRS}$	0.01

From the Z-values, it is concluded that there is no statistically significant difference in performance between any of the mutation operators. This is due to the average cost and the standard deviation values of the results being close to each other. The chosen Phase 2 mutation operator is the 1VSRS mutation operator. The DGA using this mutation operator

produces the better quality timetables more frequently than the DGA using the other tested operators.

9.2.6 Best performing DGA processes

Table 9.65 shows a summary of all the heuristics, selection methods and genetic operators that were chosen for each of the data sets.

Table 9.65: Summary of best heuristics, methods and operators for each data set

	PHASE 1				PHASE 2	
Data Set	**Primary construction Heuristic**	**Secondary heuristics**	**Selection Method**	**Genetic Operators**	**Selection Method**	**Genetic Operator**
HDTT4	Saturation		Std/Variant	2VNH		
HDTT5	Saturation		Variant	2VNH		
HDTT6	Saturation		Variant	2VNH		
HDTT7	Saturation		Variant	2VNH		
HDTT8	Saturation		Standard	2VNH		
Valouxis	Saturation	Teacher lessons, Teacher availability	Variant	1VHC	Variant	Random Swap
HS1	Saturation	SubClass/Co-Teaching	Variant	1VHC	Standard	1 Violation
HS2	Saturation	SubClass/Co-Teaching	Variant	1VHC	Standard	1 Violation
HS3	Saturation	SubClass/Co-Teaching	Variant	1VHC	Standard	1 Violation
HS4	Saturation	SubClass/Co-Teaching	Variant	1VHC	Variant	1 Violation
HS5	Largest Degree	SubClass/Co-Teaching	Variant	1VNH	Standard	Random Swap
HS7	Saturation	SubClass/Co-Teaching	Variant	1VHC	Standard	1 Violation
Lewitt	Saturation	Double Periods	Standard	Hybrid	Variant	Random Swap
Woodlands	Saturation	SubClass/Co-Teaching, Teachers involved in Co-Teaching, Teacher lessons	Standard	1VHC	Standard	Hybrid

In terms of the construction heuristic, with the exception of data set HS5, the DGA using the saturation degree heuristic was able to produce the most number of feasible timetables. In addition, the quality of the timetables produced was better than when using the other low-level heuristics. Hypothesis tests conducted indicate that the saturation degree heuristic performs better than both the random allocation and largest degree heuristics. This is due to

the ability of the saturation degree to continuously revise priorities for tuples every time an allocation is performed. As a result, a strong initial population is created allowing the mutation operators to remove the few remaining constraint violations. In the event that two or more tuples have the same saturation degree value, secondary heuristics must be applied. The secondary heuristics used may vary depending on the complexity and size of the problem.

In terms of Phase 1 selection, the choice of variant or standard tournament selection vary between problems. The objective of this phase is to find feasible timetables. The DGA approach using the more elitist standard tournament selection performed better when applied to the tightly constrained problem sets such as HDTT8, Lewitt and Woodlands. The DGA approach using the variant tournament selection performed better when applied to the smaller and less complex HDTT data sets and the Valouxis and Beligiannis problems.

The chosen mutation operator for Phase 1 varies depending on the problem. The best mutation operator involves searching for a violation and swapping it with either another violation or any random tuple. A hill climbing strategy may be adopted but is not always the best option as is found when solving the Abramson problem. When applying the approach to the Abramson problem, it is found that hill climbing adversely affects the performance of the DGA. When conducting hypothesis tests, it is found that the DGA using the mutation operators listed in the table above performs better than the DGA using the other mutation operators tested.

Similar to Phase 1, the best selection method to use for Phase 2 varies between problems. In the case of the Lewitt and Woodlands problems, either selection method can be used. For the Lewitt, Woodlands and Valouxis problems, hypothesis tests conducted found that there is no significant difference in performance when comparing the DGA using the different selection methods. For the Beligiannis problems, hypothesis tests statistically show (for three out of the five data sets) that the DGA using variant tournament selection performed better than the DGA approach using standard tournament selection.

As with Phase 1, there is no single dominant mutation operator for Phase 2 and the choice of the mutation operator depends on the problem. In the case of the Woodlands problem, a 1VH mutation operator is used where a row swap occurs in the event of a soft constraint violation for tuples that have subclasses or co-teaching requirements.

9.3 Fine-tuning of DGA control parameter values

This section provides a discussion on the fine-tuning of the control parameter values. The fine-tuning is performed by hand [EIBE99], i.e. experimenting with the DGA using different parameter values and choosing the parameter value that produced the best results. It must be noted that parameters are not independent and different sets of parameter values may perform better than others. However, experimentation using different combinations of parameter values will be a time consuming process that will involve a large number of experiments. It is thus practically unfeasible and as a result the parameter values obtained in this study cannot be regarded as optimal.

Each parameter is fine-tuned while other processes and parameter values remain constant. Similar to experiments conducted by Caldeira [CALD97], Tongchim [TONG99], Sarmady [SARM07], Goldberg et al. [GOLE89], and Sigl [SIGL03], between three and four values are tested per parameter with each value being within a specified range. The tested parameters along with the range of values being tested are listed in Table 9.66 below. The range of population sizes are based on the population sizes used in studies by Goldberg et al. [GOLE89], Spears [SPED91] and Zitzler [ZITZ99]. In the study by Xie [XIE09], tournament sizes of between 5 and 20 were used and similar values are tested in this study.

Table 9.66: Ranges for each parameter value

Parameter	Tested range	Note:
SCM size	1 to 100	Only applicable to Phase 1.
Population size	200 to 1000	Constant population size adopted for every generation.
Tournament size	5 to 20	Applicable to tournament selection for Phase 1 and Phase 2.
Swaps	20 to 200	Applicable to mutation operators for Phase 1 and Phase 2.
Maximum number of generations	20 to 75	Applicable to Phase 1 and Phase 2.

Based on the performance of the algorithm with the different parameter values, a decision will be made as to which parameter values to use.

9.3.1 The Abramson benchmark school timetabling problem

This section describes the results of fine-tuning the parameter values for the Abramson benchmark problem.

9.3.1.1 Fine tuning the SCM size

The SCM creates and evaluates a set of timetables with the best timetable being added to the initial population. The processes of the genetic algorithms as well as the parameter values used when testing each SCM size is listed in the table below:

Table 9.67: Processes and parameter values to test best SCM size (HDTT problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	2VNH
Constant Parameter Values	
Population Size	1000
Tournament Size	10
Swaps per mutation	20
Generations	50

Table 9.68 lists the success rates found when using each SCM size as well as the number of generations required to find a solution.

Table 9.68: SCM parameter value – Performance comparison

	SUCCESS RATE				AVERAGE GEN SOLUTION FOUND		
	SCM Size				SCM Size		
	SCM Size 1	SCM Size 50	SCM Size 100		SCM Size 1	SCM Size 50	SCM Size 100
HDTT4	100.00%	100.00%	100.00%	HDTT4	2	1	1
HDTT5	93.33%	100.00%	100.00%	HDTT5	5	2	2
HDTT6	46.67%	63.33%	56.67%	HDTT6	9	6	7
HDTT7	20.00%	23.33%	20.00%	HDTT7	19	16	15
HDTT8	6.67%	6.67%	6.67%	HDTT8	40	26	20

The table shows that the DGA manages to produce feasible solutions for all SCM values tested. The average hard constraint (HC) cost and the standard deviations are shown in Table 9.69.

Table 9.69: Average HC Cost (and standard deviation) for different SCM values

	SCM Size 1	SCM Size 50	SCM Size 100
HDTT4	0 (0)	0 (0)	0 (0)
HDTT5	0.13 (0.51)	0 (0)	0 (0)
HDTT6	1.07 (1.01)	0.73 (0.98)	0.87 (1.01)
HDTT7	1.6 (0.81)	1.57 (0.9)	1.63 (0.85)
HDTT8	1.9 (0.55)	1.9 (0.55)	2 (0.64)

By observing the success rates, it is noted that for each parameter value, there is very little difference in performance. The DGA finds feasible timetables for almost every run when applied to the data sets HDTT4 and HDTT5. As the number of requirements increases, the success rate begins to fall. Figure 9.16 illustrates that the DGA using an SCM value of 50 produces more feasible timetables than when using other SCM parameter values, but the difference is small. Based on the SCM values tested, it is concluded that the SCM does not play a major role in producing feasible timetables for the Abramson school timetabling problem.

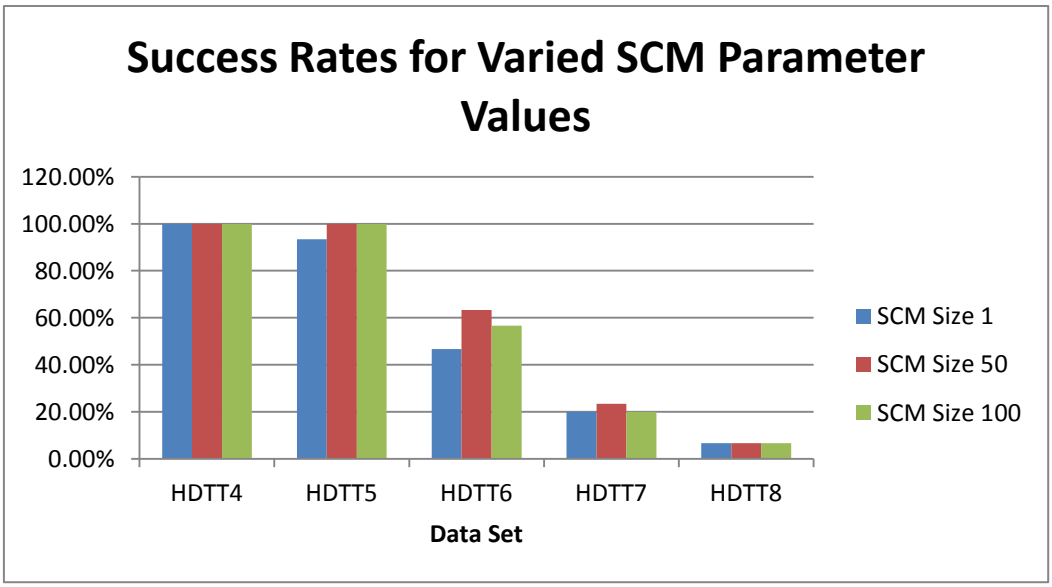


Figure 9.16: Success rates for various SCM Values

Figure 9.17 illustrates the average number of generations required to find a solution for each data set. The algorithm takes the longest time (in generations) to find a solution when no SCM is used. The main difference is found when using SCM values of 50 and 100 for the final data set (HDTT8), with the difference being 5 generations apart. From the SCM values tested, a value of 50 will be used as the DGA produces a slightly higher success rate when using this value.

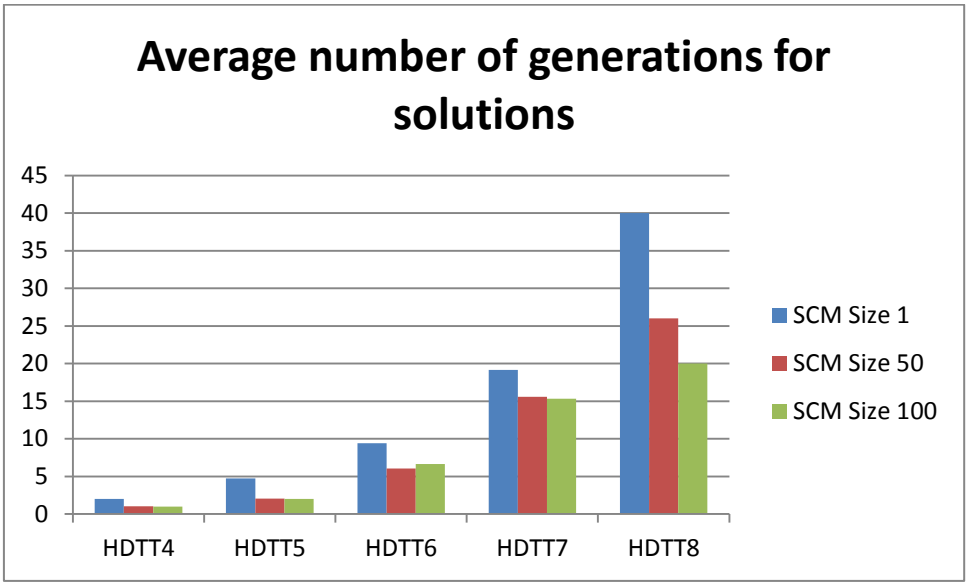


Figure 9.17: Bar chart showing average generations for each SCM size

9.3.1.2 Fine-tuning the population size

Initially, smaller population sizes were attempted but as can be seen in Table 9.70, results were poor as feasible timetables were only induced for the smaller data sets. As the number of resources increases, the search area becomes larger and the DGA using smaller population sizes of 50 and 100 performed poorly.

Table 9.70: Trial runs using population sizes of 100 and 50

Average HC Violations (and best HC cost): Population size = 100				
HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
0 (0)	1.11 (0)	1.78 (0)	2.89 (2)	3.33 (0)
Average HC Violations (and best HC cost): Population size = 50				
HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
1.33 (0)	1.55 (0)	2.22 (2)	2.67 (2)	4.33 (2)

As a result of the poor performance using smaller population sizes, larger population sizes of 500, 750 and 1000 were considered. Table 9.71 lists the processes and parameter values that were kept constant when testing the different population sizes.

Table 9.71: Processes and parameter values to test best population size (HDTT problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	2VNH
Constant Parameter Values	
SCM Size	50
Tournament Size	10
Swaps per mutation	20
Generations	50

The success rates and the number of generations taken to find a solution are shown in Table 9.72.

Table 9.72: Population Size results

	SUCCESS RATE				AVERAGE GEN SOLUTION FOUND		
	Pop Size				Pop Size		
	500	750	1000		500	750	1000
HDTT4	100.00%	100.00%	100.00%	HDTT4	1	1	1
HDTT5	86.67%	100.00%	100.00%	HDTT5	3	2	2
HDTT6	33.33%	46.67%	63.33%	HDTT6	6	6	6
HDTT7	3.33%	20.00%	23.33%	HDTT7	18	14	16
HDTT8	3.33%	0.00%	6.67%	HDTT8	34	50	26

Table 9.73 lists the average cost (and standard deviations) found when using the DGA with different population sizes.

Table 9.73: Average hard constraint violations (and standard deviations) for different population sizes

	Population Size		
	500	750	1000
HDTT4	0 (0)	0 (0)	0 (0)
HDTT5	0.27 (0.69)	0 (0)	0 (0)
HDTT6	1.33 (0.96)	1.07 (1.01)	0.73 (0.98)
HDTT7	1.97 (0.41)	1.6 (0.81)	1.57 (0.9)
HDTT8	2.3 (0.7)	2.3 (0.6)	1.9 (0.55)

From Table 9.72 and 9.73 above, it appears that an increase in the population size not only resulted in more feasible timetables, but the average number of constraint violations was also reduced. A bar graph of the success rates for different population sizes is depicted below (Figure 9.18):

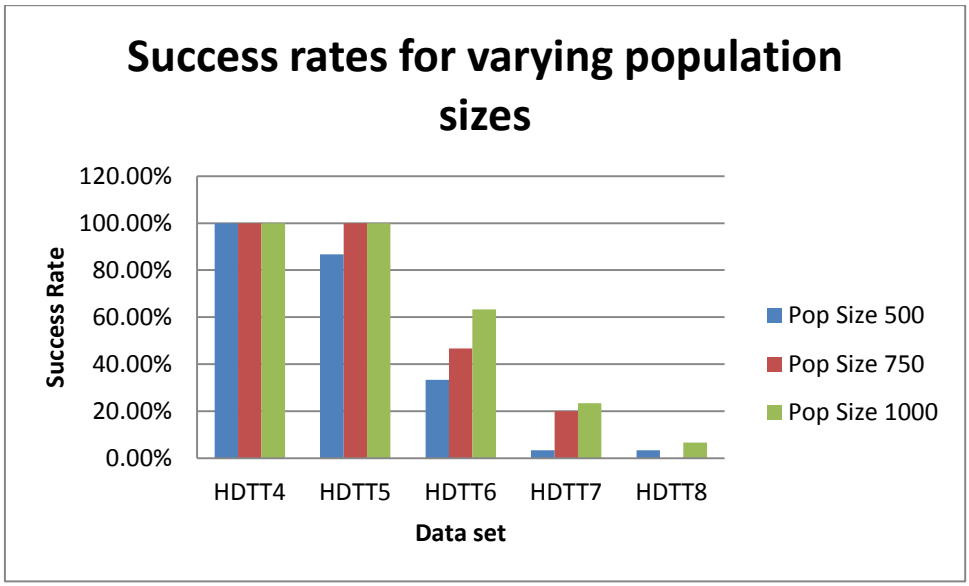


Figure 9.18: Bar chart showing success rates for various population sizes

The bar chart illustrates that a population size of 1000 produces more feasible timetables than the other population sizes tested. The success rates also tend to decrease as the size and complexity of the problem increases. A population size of 1000 will be used as the DGA performs best when using this value.

9.3.1.3 The best tournament size

The tournament size parameter determines the number of participants to be considered during the selection phase of the DGA approach. Table 9.74 lists the processes and parameter values used when testing the different tournament sizes.

Table 9.74: Processes and parameter values to test best tournament size (HDTT problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	2VNH
Constant Parameter Values	
SCM Size	50
Population size	1000
Swaps per mutation	20
Generations	50

Table 9.75 shows the success rates and the average number of generations taken to find a solution. Table 9.76 shows the average hard constraint cost and the standard deviations found when running the DGA with the different tournament sizes.

Table 9.75: Tournament size results

	SUCCESS RATE					AVERAGE GEN SOLUTION FOUND			
	Tournament Size					Tournament Size			
	5	10	15	20		5	10	15	20
HDTT4	100.00%	100.00%	100.00%	100.00%	HDTT4	1	1	1	1
HDTT5	100.00%	100.00%	96.67%	100.00%	HDTT5	2	2	2	2
HDTT6	56.67%	63.33%	76.67%	56.67%	HDTT6	8	6	6	5
HDTT7	30.00%	23.33%	20.00%	10.00%	HDTT7	20	16	14	14
HDTT8	6.67%	6.67%	3.33%	3.33%	HDTT8	33	26	29	20

Table 9.76: Average HC Cost (and standard deviations)

	Tournament Size			
	5	10	15	20
HDTT4	0 (0)	0 (0)	0 (0)	0 (0)
HDTT5	0 (0)	0 (0)	0.07 (0.37)	0 (0)
HDTT6	0.87 (1.01)	0.73 (0.98)	0.47 (0.86)	0.87 (1.01)
HDTT7	1.4 (0.93)	1.57 (0.9)	1.6 (0.81)	1.83 (0.65)
HDTT8	1.93 (0.58)	1.9 (0.55)	2 (0.45)	2.03 (0.49)

A bar graph depicting the success rate of each tournament size is shown below:

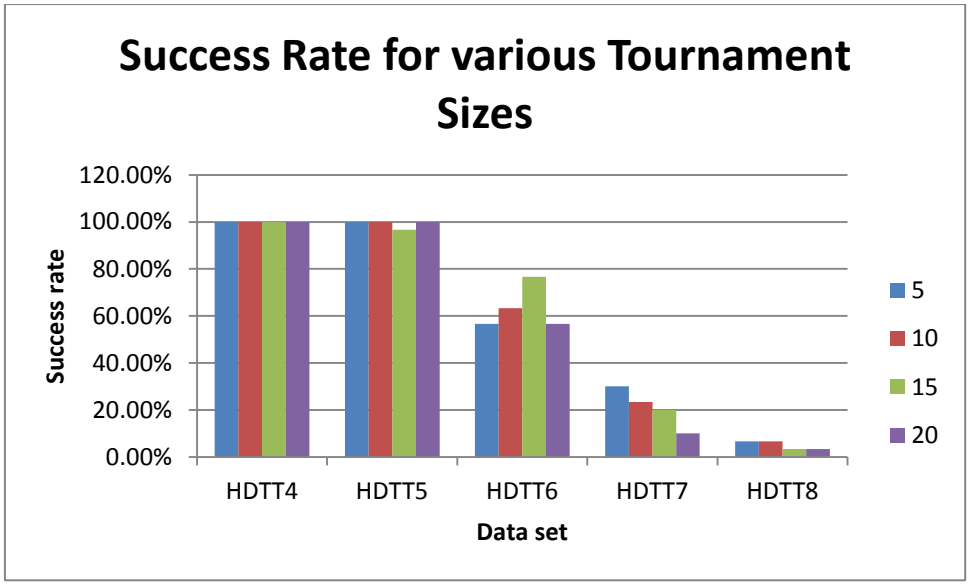


Figure 9.19: Column chart showing success rates for various tournament sizes

Feasible solutions are found when using any tournament parameter value with the success rates differing for each data set. For data sets HDTT4 and HDTT5, high success rates of at least 96% are found when using any tournament size. As the number of requirements increase, the success rate once again decreases. For HDTT6, the best tournament size to use is 15, while for HDTT7; the best tournament size to use is 5. For the HDTT8 data set, the algorithm performs best when using tournament sizes of 5 and 10. A bar chart showing the time taken to find solutions in terms of generations is shown below (Figure 9.20).

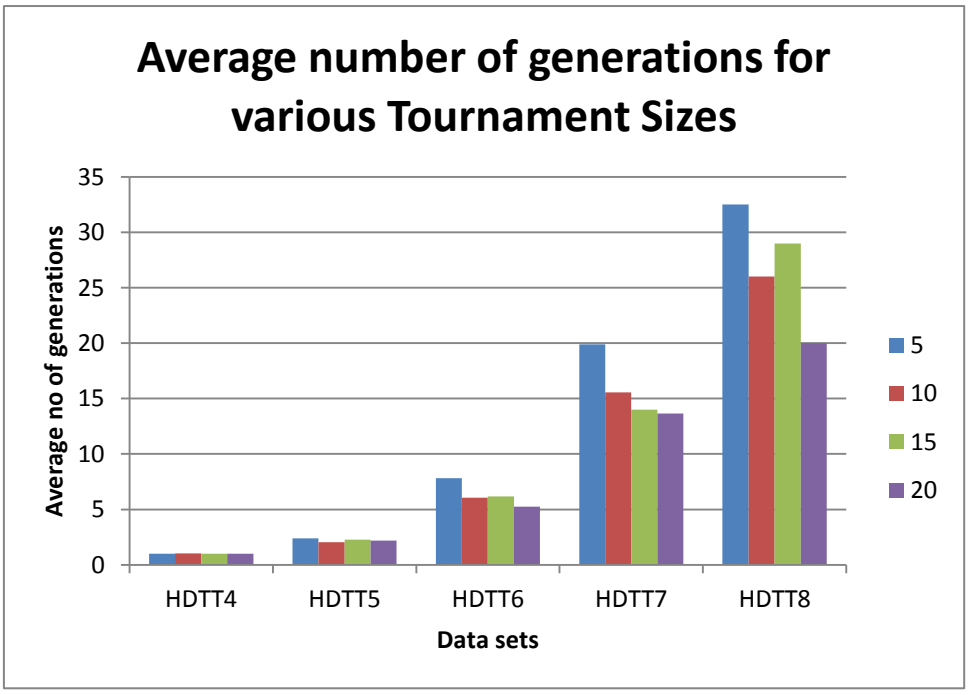


Figure 9.20: Column chart showing average generations for various tournament sizes

The chart illustrates a relationship between the tournament size and the number of generations in which solutions are found. A disadvantage of a higher tournament size is that the genetic algorithm converged prematurely due to the higher selection pressure. While feasible solutions are found in a fewer number of generations, the success rate is affected as the algorithm converges prematurely. To conclude, the best tournament size is dependent on the problem. A tournament size of 10 is used as this tournament size provides a fair balance between the success rate and number of generations taken to find a solution.

9.3.1.4 Fine-tuning the number of swaps

The mutation operator performs a number of swaps to reduce hard constraint violations. Table 9.77 lists the processes and parameter values used to test the different number of swaps. A table listing the success rates and average number of generations to find a solution for each swap parameter value is shown below in Table 9.78.

Table 9.77: Processes and parameter values to test best swaps per mutation (HDTT problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	2VNH
Constant Parameter Values	
SCM Size	50
Population size	1001
Swaps per mutation	200

Table 9.78: Results based on number of swaps

	SUCCESS RATE					AVERAGE GEN SOLUTION FOUND			
	Swaps					Swaps			
	20	50	100	200		20	50	100	200
HDTT4	100.00%	100.00%	100.00%	100.00%	HDTT4	1	1	1	1
HDTT5	100.00%	100.00%	100.00%	100.00%	HDTT5	2	1	1	1
HDTT6	63.33%	76.67%	80.00%	100.00%	HDTT6	6	5	3	2
HDTT7	23.33%	23.33%	40.00%	46.67%	HDTT7	16	10	7	6
HDTT8	6.67%	10.00%	6.67%	13.33%	HDTT8	26	19	12	12

A bar chart depicting the success rates for each data set is shown below (Figure 9.21).

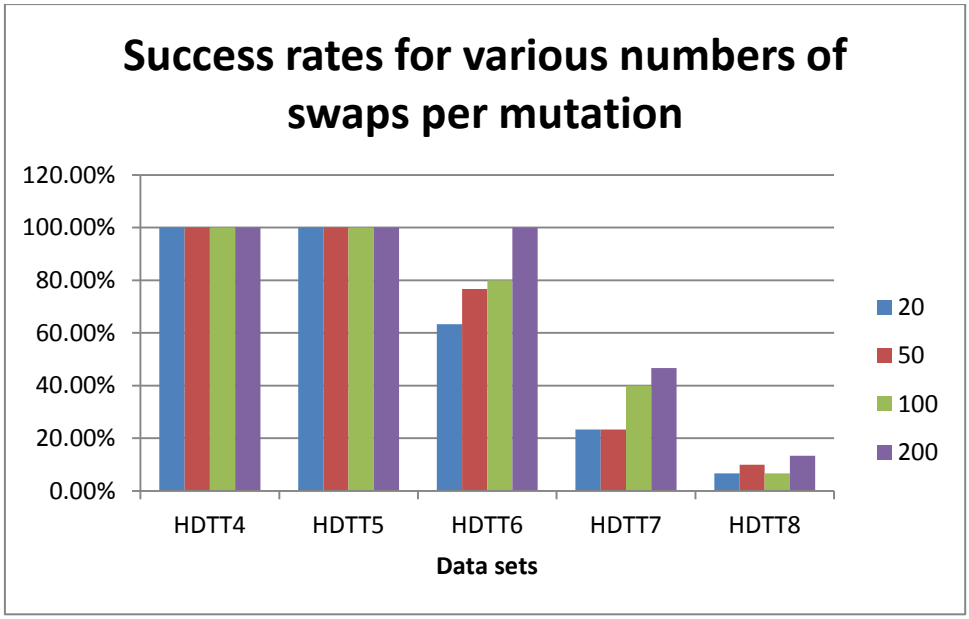


Figure 9.21: Bar chart showing success rates for various numbers of swaps

As can be seen in Figure 9.21 and Table 9.78, there appears to be a relationship between the success rate and number of swaps. The success rate increases as the number of swaps increase. An increase in the number of swaps also affects the number of generations taken to find a solution as can be seen in the column chart in Figure 9.22.

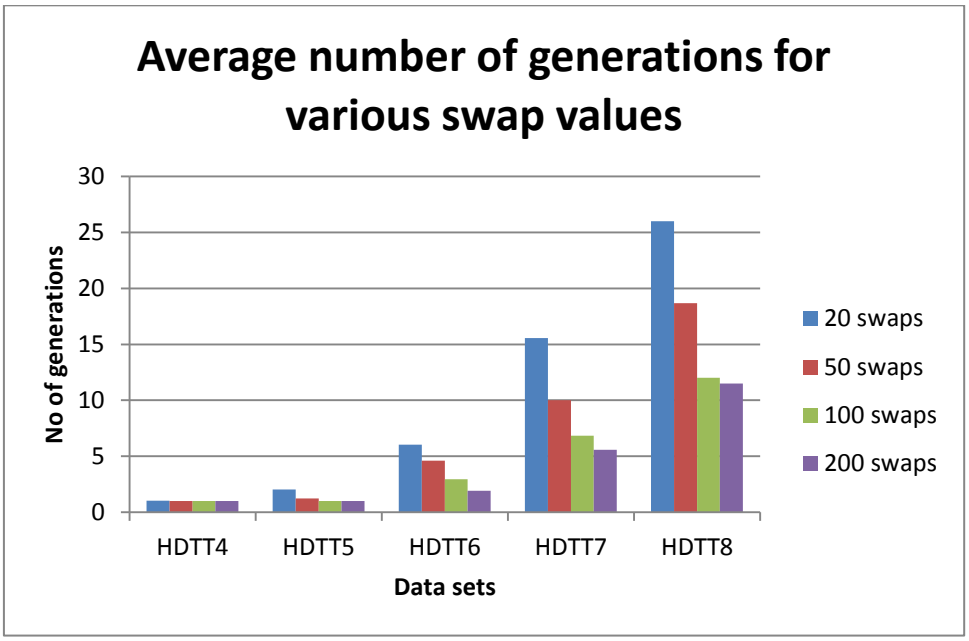


Figure 9.22: Column chart showing average generations for each swap parameter value

From the column chart illustrated in Figure 9.22, it appears that an increase in the number of swaps reduces the number of generations required to find a feasible solution. The number

of swaps to be used for the DGA is 200 due to the increase in success rate and the reduction in generations taken to find a solution.

9.3.1.5 Maximum number of generations

Initially, the maximum number of generations is set to fifty. While conducting tests described in sections 9.3.1.1 to 9.3.1.4, it was found that either a solution is found before generation 50 or the algorithm converges prematurely. Thus the generation limit for this problem will be set at fifty generations.

9.3.2 The Valouxis school timetabling problem

This section describes the fine-tuning of the parameter values for the DGA when applied to the Valouxis school timetabling problem and the results obtained.

9.3.2.1 Fine-tuning the SCM size

The SCM parameter value sets the number of timetables to develop when creating a single individual of the initial population. Once these timetables are created, they are evaluated and the fittest timetable is added to the initial population.

Table 9.79: Processes and parameter values to test best SCM size (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	2 Violation
Constant Parameter Values	
Population Size	750
Tournament Size	10
Swaps per mutation	20
Generations	50

The influence of the SCM parameter values on success rates, average HC violations and average timetable quality is shown in Table 9.80 below.

Table 9.80: Results for various SCM values

Success Rate		
SCM = 1	SCM = 10	SCM = 50
70.00%	83.33%	93.33%
Average HC Cost (and standard deviations)		
SCM = 1	SCM = 10	SCM = 50
0.3 (0.47)	0.17 (0.38)	0.07 (0.25)
Average Quality (Feasible Timetables)		
SCM = 1	SCM = 10	SCM = 50
49.86	47.00	46.89

When using the DGA approach with an SCM value of 50, a 93.33% success rate is found over thirty runs. This is the highest success rate when compared to using SCM values of 1 and 10. Over the thirty runs conducted, the DGA approach with an SCM size of 50 also produces the best quality timetables on average. In terms of runtime, a DGA approach with a larger SCM size may take longer as more timetables are produced for every individual of the initial population.

A frequency chart, showing the distribution of timetables in terms of quality, is shown below:

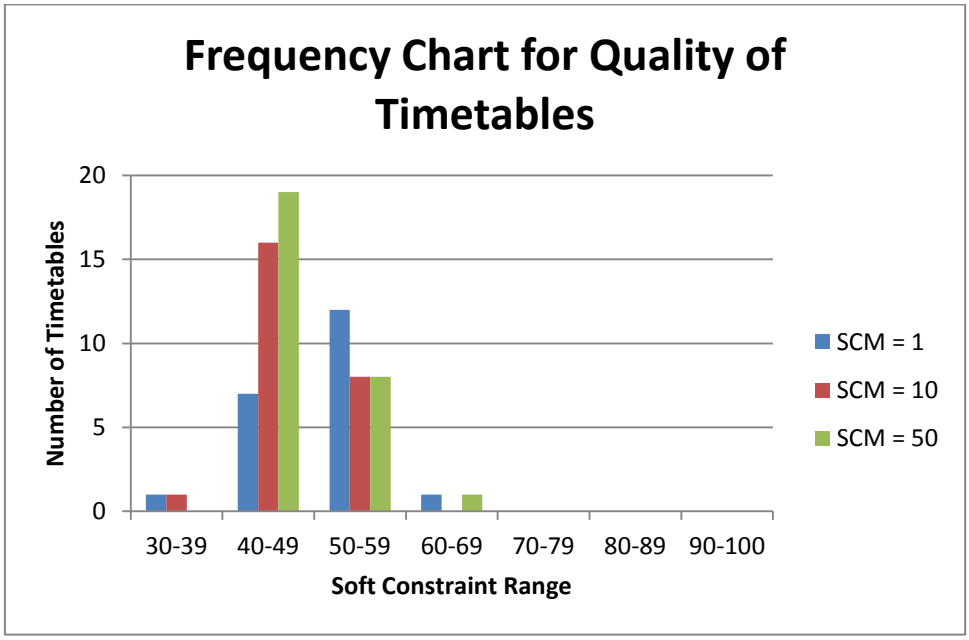


Figure 9.23: Frequency chart showing quality for various SCM values

When using an SCM value of 50, most timetables produced contain between 40 and 49 soft constraint violations. The DGA with an SCM size of 10 produces the best quality timetable with 36 soft constraint violations. An SCM value of 50 is chosen due to a higher success

rate obtained when using this value. The DGA also produces the best quality timetables on average when using an SCM size of 50.

9.3.2.2 Fine-tuning the population size

The DGA is tested using three different population sizes (500, 750 and 1000) and the results are evaluated and compared in order to determine the population size that produces the best success rate and timetable quality. Trial runs using smaller population sizes of 50 and 100 were attempted and it was found that feasible timetables were not always produced. Additionally, the quality of the feasible timetables was poor when compared to timetables produced using larger population sizes. The results of the trial runs are shown below. The low success rates indicate that the initial coverage of the search space was too small and the evolutionary process found difficulty in finding feasible solutions.

Table 9.81: Trial runs using small population sizes (Valouxis)

	Population size = 100	Population size = 50
Success rate	40%	20%
Average HC Cost	0.78	1.22
Best HC Cost	0	0
Average SC Cost	51.75	58.5
Best SC Cost	46	57

The following table shows the processes and parameters used when testing the effect of different population sizes on the performance of the DGA.

Table 9.82: Processes and parameter values to test best population size (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	2 Violation
Constant Parameter Values	
SCM Size	50
Tournament Size	10
Swaps per mutation	20
Generations	50

A summary of the performance of the algorithm using each of the population sizes is shown in Table 9.83 below.

Table 9.83: Results for various population sizes

Success Rate		
Pop = 500	Pop = 750	Pop = 1000
73.33%	93.33%	100.00%
Average HC Cost (and standard deviations)		
Pop = 500	Pop = 750	Pop = 1000
0.27 (0.45)	0.07 (0.25)	0 (0)
Average Quality (Feasible Timetables)		
Pop = 500	Pop = 750	Pop = 1000
48.68	46.89	45.53

Using a population size of 1000 allows the DGA to find feasible solutions for all thirty runs that are conducted. The difference in quality when using different population sizes is small, with the DGA producing between 45 and 46 soft constraint violations when using population sizes of 750 and 1000 respectively. A frequency chart showing the number of timetables found in each soft constraint cost is displayed below (Figure 9.24).

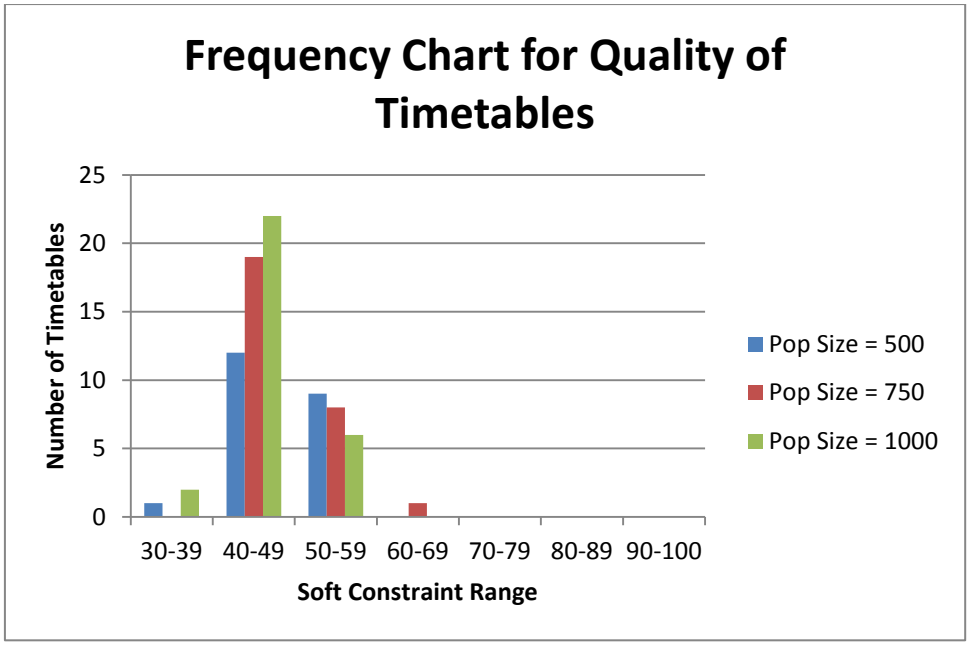


Figure 9.24: Frequency chart showing quality for various population sizes

The frequency chart illustrates that the DGA using a population size of 1000 produces more feasible timetables that contain between 30 and 50 soft constraint violations than when using any of the other tested population sizes. A population size of 1000 will therefore be used due to the DGA producing the highest success rate when using this population size. Better quality timetables are also produced when using this population size.

9.3.2.3 Fine-tuning the tournament size

The DGA is run with three tournament sizes of 5, 10 and 15. The following table (Table 9.84) displays the processes and parameter values used to test for the best tournament size.

Table 9.84: Processes and parameter values to test best tournament size (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	2 Violation
Constant Parameter Values	
SCM Size	50
Population Size	1000
Swaps per mutation	20
Generations	50

The success rates, average HC cost and average timetable quality found when using different tournament sizes are shown in the table below.

Table 9.85: Results for various tournament sizes

Success Rates		
5	10	15
100%	100%	93.33%
Average HC Cost (and standard deviations)		
5	10	15
0 (0)	0 (0)	0.07 (0.25)
Average Quality (Feasible timetables)		
5	10	15
46.03	45.53	46.68

The DGA produces high success rates when using all tested tournament sizes. The difference in timetable quality when using different tournament sizes is very small. A tournament size of 10 produces timetables with the best average quality of approximately 45 soft constraint violations per feasible timetable. The timetable with the best quality is produced when using a tournament size of 15 (37 soft constraint violations). The frequency chart (Figure 9.25) shows the distribution of timetables with regard to quality.

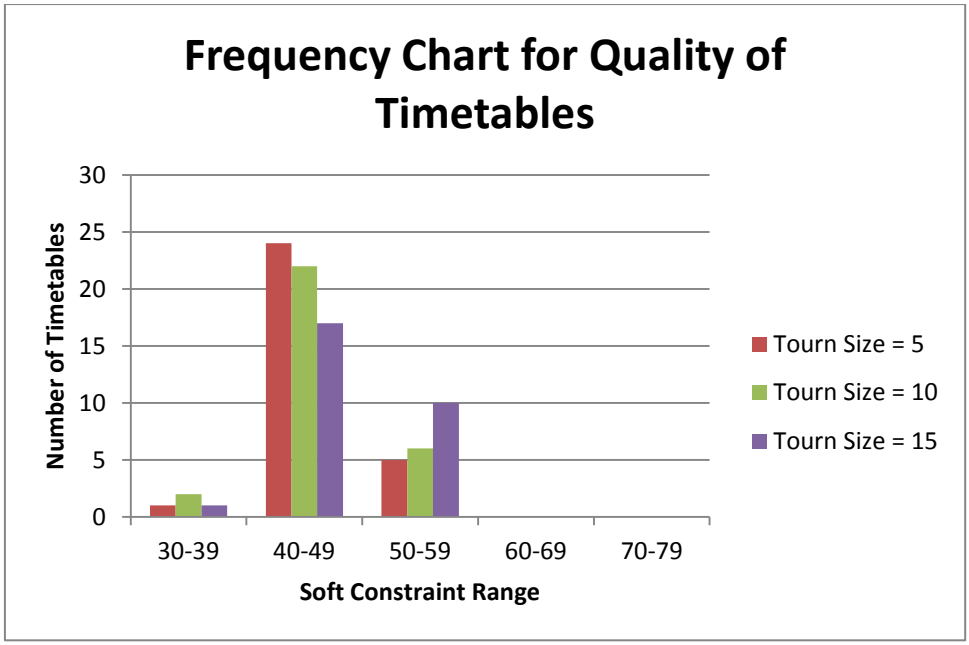


Figure 9.25: Frequency chart showing quality for various tournament sizes

For all runs, the DGA using all tournament sizes produces at least one timetable with soft constraint costs of between 30 and 39. The majority of the timetables generated contain between 40 and 49 soft constraint violations. In conclusion, a tournament size of 10 produces the best performance in terms of average timetable quality (although the difference is small). The frequency chart also shows that the DGA produces better quality timetables more frequently when using a tournament size of 10.

9.3.2.4 Fine-tuning the number of swaps

The DGA is tested using four different swap parameter values. The objective is to determine the best swap parameter value to use. Table 9.86 lists the processes and parameter values used when testing the DGA with each of the swap parameter values.

Table 9.86: Processes and parameter values to test best swap parameter value (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	2 Violation
Constant Parameter Values	
SCM Size	50
Population Size	1000
Tournament Size	10
Generations	50

Table 9.87 shows the success rates and average timetable quality found when the DGA is run with different swap parameter values.

Table 9.87: Results for different number of swaps per mutation

Success Rate			
Swaps = 20	Swaps = 50	Swaps = 100	Swaps = 200
100%	100%	100%	100%
Average Quality (Feasible Timetables)			
Swaps = 20	Swaps = 50	Swaps = 100	Swaps = 200
44.63	42.47	41.97	41.03

The DGA produces feasible timetables when using any swap parameter value. Based on the average quality of the timetables produced, a trend is shown where increasing the number of swaps results in an improvement of the quality of the timetable. This also affects the runtime as more swaps result in the algorithm taking longer to reach the generation limit. The best timetable found contains 35 soft constraint violations and is found when using a swap parameter value of 100. The frequency diagram (Figure 9.26) below illustrates the distribution of timetables in terms of quality when using different swap parameter values.

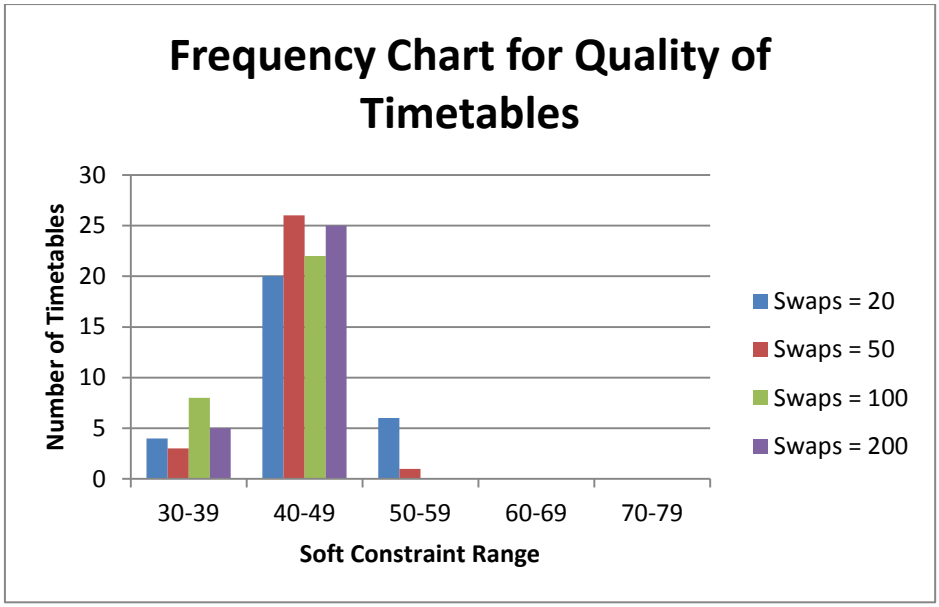


Figure 9.26: Frequency chart showing quality for various swap values

Figure 9.26 clearly shows that the majority of timetables created contain between 40 and 49 soft constraint violations. For all runs, eight timetables are found that have between 30 and 39 soft constraint violations when running the genetic algorithm with a swap parameter value of 100. A swap value of 100 will therefore be used because of the high success rate obtained as well as having produced the most high quality timetables.

9.3.2.5 Maximum number of generations

In order to determine the best number of generations to use, the convergence point of the algorithm must be determined. Three generation parameter values are tested with the following processes and parameter values being constant.

Table 9.88: Processes and parameter values to test best number of generations (Valouxis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Variant
Mutation	1VH
Phase 2	
Selection	Variant
Mutation	2 Violation
Constant Parameter Values	
SCM Size	50
Population Size	1000
Tournament Size	10
Swaps per mutation	100

The success rates listed in Table 9.89 indicate that feasible timetables are produced for all three tested generation values. In terms of quality, generation values of 50 and 75 produce the same quality timetables for all runs. This means that before this point, the algorithm has converged between generations 20 and 50. For the Valouxis problem, a generation parameter value of 50 is used. The timetables do not improve in quality after 50 generations.

Table 9.89: Results for different number of generations per phase

Success Rates		
20	50	75
100%	100%	100%
Average Quality		
20	50	100
43.43	41.97	41.97

9.3.3 The Beligiannis Greek school timetabling problem

The following sections outline the fine-tuning process for control parameter values when applying the DGA to the Beligiannis high school timetabling problem.

9.3.3.1 Fine-tuning the SCM size

The sequential construction method creates a set of timetables with the best timetable being added to the initial population. The SCM is called X times where X is the size of the

population. Table 9.90 lists the processes and parameter values used to test the DGA using the different SCM values.

Table 9.90: Processes and parameter values to test SCM size (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for HS5)
Selection	Variant
Mutation	1VH (1VNH for HS5)
Phase 2	
Selection	Standard
Mutation	1 Violation (Random swap for HS5)
Constant Parameter Values	
Population Size	750
Tournament Size	10
Swaps per mutation	20
Number of generations	50

The DGA is tested using four SCM values and the success rates are listed below.

Table 9.91: Success rates for various SCM parameter values

Success Rates				
	SCM = 1	SCM = 10	SCM = 25	SCM = 50
HS1	100.00%	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%	100.00%
HS5	100.00%	100.00%	100.00%	100.00%
HS7	100.00%	100.00%	100.00%	100.00%

An SCM parameter value of 1 indicates that no SCM is used. Table 9.91 shows that for all runs, all SCM values produce feasible timetables. In order to determine the best SCM value, the average quality of the timetables produced must be compared. The average timetable quality found for each data set by applying the DGA with different SCM parameter values is shown in the Table 9.92.

Table 9.92: Average quality found per data set using different SCM parameter values

Average Quality (and standard deviation)				
	SCM = 1	SCM = 10	SCM = 25	SCM = 50
HS1	117.67 (8.1)	115.10 (7.02)	114.03 (5.83)	115.77 (8)
HS2	120.27 (7.24)	122.87 (8.34)	123.20 (7.03)	122.53 (7.32)
HS3	49.27 (4.83)	48.00 (3.99)	48.20 (5.23)	48.20 (4.06)
HS4	75.83 (4.59)	74.27 (4.28)	73.63 (4.30)	74.03 (5.40)
HS5	57.40 (7.83)	55.53 (7.97)	57.47 (8.88)	54.80 (8.56)
HS7	139.80 (6.12)	138.13 (7.44)	137.33 (5.65)	137.53 (6.13)

For the data sets HS1, HS4 and HS7, an SCM value of 25 produces the best quality timetables. An SCM value of 1 and 10 produce the best quality timetables for data sets HS2 and HS3 respectively. When applied to data set HS5, the DGA produces the best quality timetables when using an SCM value of 50. When using this SCM value, the number of average soft constraint violations is reduced by at least one.

The success rates and average quality vary between data sets. Any of the tested SCM values could be used and feasible timetables are produced. In terms of quality, the ideal SCM parameter value varies between data sets. An SCM size of 25 is used when conducting the remaining tests for this problem.

9.3.3.2 Fine-tuning the population size

Three population sizes of 200, 500 and 750 are tested and 100% success rates are achieved when applying the DGA to all of the data sets, meaning that the DGA using any of the three population sizes tested manages to produce feasible timetables for every run (see Table 9.95). Trial runs using smaller population sizes were also attempted. While feasible timetables were found, timetable quality was poor when compared to larger population sizes (see Table 9.93).

Table 9.93: Trial runs for smaller populations sizes (Beligiannis problem)

Populations size = 100						
	HS1	HS2	HS3	HS4	HS5	HS7
Average SC cost	125.78	136.78	53.56	79.11	75.13	145.44
Best SC cost	117	128	44	73	62	133
Population size = 50						
	HS1	HS2	HS3	HS4	HS5	HS7
Average SC cost	128	140.78	54.89	83.89	73.38	150.67
Best SC cost	117	131	52	79	63	145

Table 9.94 below shows the processes and parameter values that are kept constant when testing each population size.

Table 9.94: Processes and parameter values to test population size (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for HS5)
Selection	Variant
Mutation	1VH (1VNH for HS5)
Phase 2	
Selection	Standard
Mutation	1 Violation (Random swap for HS5)
Constant Parameter Values	
SCM size	25 (50 for HS5)
Tournament Size	10
Swaps per mutation	20
Number of generations	50

Table 9.95: Success rates for different population sizes

Success Rates			
	Pop Size = 200	Pop Size = 500	Pop Size = 750
HS1	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%
HS5	100.00%	100.00%	100.00%
HS7	100.00%	100.00%	100.00%

In order to determine the best population size, timetable quality must also be compared. The average timetable quality found when the DGA approach is applied to each data set using different population sizes is listed in Table 9.96.

Table 9.96: Average quality produced for different population sizes

Average Soft Constraint Violations (and standard deviations)			
	Pop Size = 200	Pop Size = 500	Pop Size = 750
HS1	120.03 (7.91)	117.40 (6.09)	114.03 (5.83)
HS2	129.40 (8.27)	125.83 (8.79)	123.20 (7.03)
HS3	51.57 (5.67)	49.80 (3.97)	48.20 (5.23)
HS4	76.60 (5.12)	74.07 (4.76)	73.63 (4.30)
HS5	61.70 (10.82)	59.30 (10.55)	54.80 (8.56)
HS7	143.07 (8.41)	138.00 (6.51)	137.33 (5.65)

The table shows that for all data sets, a population size of 750 produces the best quality timetable on average. The column chart in Figure 9.27 illustrates the results listed in Table 9.96. Figure 9.27 shows that the best quality timetables are produced when using the largest population size of 750.

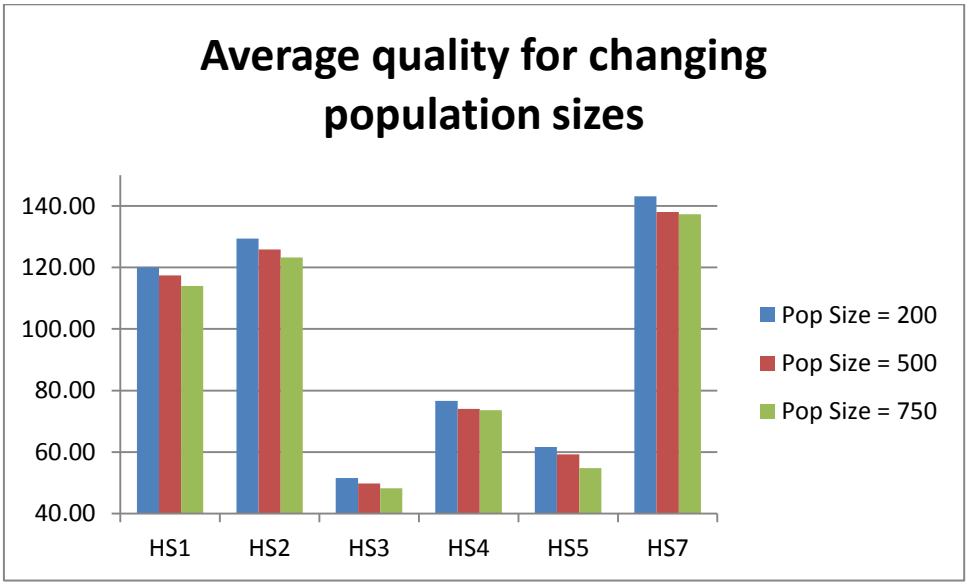


Figure 9.27: Column chart showing quality of different population sizes

The population size to be used is 750 as the DGA approach found better quality timetables when using this parameter value.

9.3.3.3 Fine-tuning the tournament size

The DGA is run using different tournament sizes of 5, 10 and 15. The processes and parameter values used to test the tournament sizes are listed in the table below (Table 9.97). The results in terms of success rate when using different tournament size values are shown in the Table 9.98.

Table 9.97: Processes and parameter values to test best tournament size (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for HS5)
Selection	Variant
Mutation	1VH (1VNH for HS5)
Phase 2	
Selection	Standard
Mutation	1 Violation (Random swap for HS5)
Constant Parameter Values	
SCM size	25 (50 for HS5)
Population size	750
Swaps per mutation	20
Number of generations	50

Table 9.98: Success rates produced for various tournament sizes

Success Rates			
	Tourn Size = 5	Tourn Size = 10	Tourn Size = 15
HS1	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%
HS5	100.00%	100.00%	100.00%
HS7	100.00%	100.00%	100.00%

In order to determine the tournament size, the average quality of timetables produced also needs to be analyzed. The quality of timetables produced using the DGA approach with different tournament sizes are listed in Table 9.99.

Table 9.99: Average quality produced using different tournament sizes

Average SC Cost (and standard deviations)			
	Tourn Size = 5	Tourn Size = 10	Tourn Size = 15
HS1	115.37 (5.13)	114.03 (5.83)	114.30 (7.19)
HS2	122.13 (6.64)	123.20 (7.03)	121.87 (8.02)
HS3	46.20 (4.53)	48.20 (5.23)	47.43 (4.64)
HS4	71.70 (3.94)	73.63 (4.3)	72.07 (4.56)
HS5	52.63 (7.76)	54.80 (8.56)	57.90 (7.34)
HS7	137.60 (6.31)	137.33 (5.65)	136.90 (6.75)

Table 9.99 shows that for each data set, the DGA approach could use different tournament sizes in order to produce better quality timetables. For the larger data sets with more classes and teachers (HS1, HS2 and HS7), a tournament size of between 10 and 15 performs best (higher selection pressure) while for the smaller size data sets (HS3, HS4 and HS5), a tournament size of 5 produces the best quality timetables (lower selection pressure). With the exception of data set HS5, the tournament size is set to 15. For data set HS5, a tournament size of 5 will be used as the average quality of timetables produced using this tournament size are found to be better than when using other tournament sizes.

9.3.3.4 Fine-tuning the number of swaps

This fine-tuning test determines the number of swaps that the mutation operator must perform when applied to each individual. Swap values of 20, 50, 100 and 200 were tested with the following processes and parameter values (Table 9.100):

Table 9.100: Processes and parameter values to test best number of swaps (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for HS5)
Selection	Variant
Mutation	1VH (1VNH for HS5)
Phase 2	
Selection	Standard
Mutation	1 Violation (Random swap for HS5)
Constant Parameter Values	
SCM size	25 (50 for HS5)
Population size	750
Tournament size	15 (5 for HS5)
Number of generations	50

The success rate and average quality found for the DGA approach using four different swap parameter values is shown in Tables 9.101 and 9.102.

Table 9.101: Success rates produced using different swap parameter values

Success Rates				
	Swaps = 20	Swaps = 50	Swaps = 100	Swaps = 200
HS1	100.00%	100.00%	100.00%	100.00%
HS2	100.00%	100.00%	100.00%	100.00%
HS3	100.00%	100.00%	100.00%	100.00%
HS4	100.00%	100.00%	100.00%	100.00%
HS5	100.00%	56.67%	13.33%	0.00%
HS7	100.00%	100.00%	100.00%	100.00%

Table 9.102: Average quality produced using different swap parameter values

Average SC Cost (and standard deviations)				
	Swaps = 20	Swaps = 50	Swaps = 100	Swaps = 200
HS1	114.30 (7.19)	112.60 (6.63)	109.33 (6.72)	107.20 (6.21)
HS2	121.87 (8.02)	118.20 (7.90)	117.77 (5.82)	113.23 (7.83)
HS3	47.43 (4.64)	45.13 (3.14)	45.00 (3.95)	42.47 (3.69)
HS4	72.07 (4.56)	71.77 (4.87)	69.70 (4.15)	68.80 (4.25)
HS5	52.63 (7.76)	51.94 (7.39)	48.50 (9.88)	NA
HS7	136.90 (6.75)	135.70 (5.64)	135.17 (5.81)	130.40 (6.08)

Tables 9.101 and 9.102 show that, with the exception of data set HS5, 100% success rates are found when using the DGA approach with any swap parameter value. The average quality must be used to determine the best swap parameter value. With the exception of data set HS5, the DGA produces the best quality timetables when a swap parameter value of 200 is used. This conclusion is also be made when observing the column chart below (Figure 9.28).

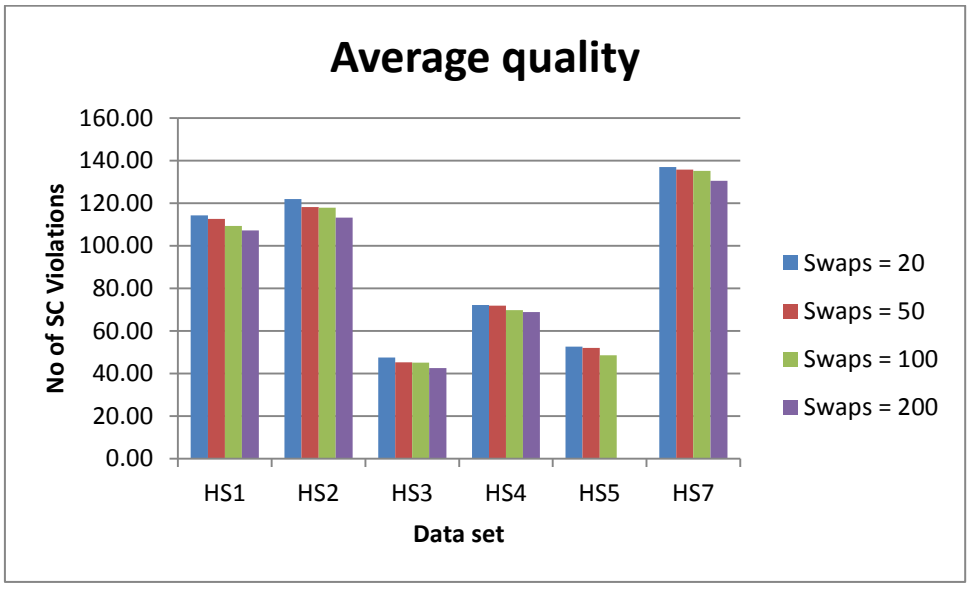


Figure 9.28: Average quality found for different swap parameter values

The column chart also shows the effect on quality of timetables produced when increasing the swap parameter value. The table also indicates an inverse relationship between the number of swaps and the resultant number of soft constraint violations i.e. as the number of swaps increase, the number of soft constraint violations decrease. This trend is observed when the DGA is applied to any of the data sets.

Therefore, the best value for the number of swaps parameter is 200. For the HS5 data set, the best number of swaps is set to 20 as this swap parameter value produces the most number of feasible timetables.

9.3.3.5 Maximum number of generations

Generation parameter values of 20, 50 and 75 are used and the performance of the DGA is evaluated for each generation parameter value.

Table 9.103: Processes and parameter values to test best number of generations (Beligiannis problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree (Largest degree for HS5)
Selection	Variant
Mutation	1VH (1VNH for HS5)
Phase 2	
Selection	Standard
Mutation	1 Violation (Random swap for HS5)
Constant Parameter Values	
SCM size	25 (50 for HS5)
Population size	750
Tournament size	15 (5 for HS5)
Number of swaps	200 (20 for HS5)

The results are displayed below and show the success rates and average quality of the timetables obtained.

Table 9.104: Success rates for varying number of generations

Success Rates			
	Gens = 20	Gens = 50	Gens = 75
HS1	100%	100%	100%
HS2	100%	100%	100%
HS3	100%	100%	100%
HS4	100%	100%	100%
HS5	0%	100%	100%
HS7	100%	100%	100%

Table 9.105: Average quality produced for different generation parameter values

Average SC Cost			
	Gens = 20	Gens = 50	Gens = 75
HS1	107.73	107.20	107.17
HS2	113.50	113.23	113.23
HS3	42.47	42.47	42.47
HS4	68.80	68.80	68.80
HS5	NA	52.63	52.33
HS7	130.80	130.40	130.40

100% success rates are found for all data sets except when using 20 generations for data set HS5. This indicates that for this data set, the DGA has not yet converged. In terms of timetable quality, the DGA converges before generation 20 for data sets HS3 and HS4 (the smaller data sets in terms of requirements, teachers and classes). For the data sets HS1 and HS5, timetable quality stops improving after generation 50 while for data sets HS2 and HS7, timetables stop improving after generation 20. The best number of generations (from the values tested) for all data sets is 75 as the algorithm would have converged at this point.

9.3.4 The W.A. Lewitt primary school timetabling problem

This section describes the fine-tuning process when the DGA approach is applied to the W.A. Lewitt primary school timetabling problem. All parameter values except the SCM size are used for both Phase 1 and Phase 2 of the approach.

9.3.4.1 Fine-tuning the SCM size

The performance of the DGA is compared using four SCM values. These values are 1, 10, 20 and 50. The following processes and parameter values were kept constant when testing these SCM values.

Table 9.106: Processes and parameter values to test best SCM size (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
Population size	500
Tournament size	10
Number of swaps	200
Number of generations	50

The success rates and average quality of timetables produced using the DGA approach with each SCM value is shown in Table 9.107.

Table 9.107: Success rates and average quality obtained using different SCM parameter values

Success Rates			
SCM = 1	SCM = 10	SCM = 20	SCM = 50
26.67%	43.33%	60.00%	33.33%
Average HC Cost (and standard deviations)			
SCM = 1	SCM = 10	SCM = 20	SCM = 50
1.10 (0.8)	0.77 (0.82)	0.63 (0.89)	1.03 (1.00)
Average Quality (and standard deviations)			
SCM = 1	SCM = 10	SCM = 20	SCM = 50
10.50 (2.93)	10.54 (3.36)	10.83 (2.55)	10.70 (2.87)

The best success rate is obtained when using a SCM value of 20. In terms of average quality, there is very little difference found when using any of the SCM values tested. The frequency diagram (Figure 9.29) also indicates that 6 out of the 18 (33%) timetables produced using an SCM value of 20 contain between 0 and 9 soft constraint violations. The DGA with an SCM value of 10 produces 38% of the timetables in this range.

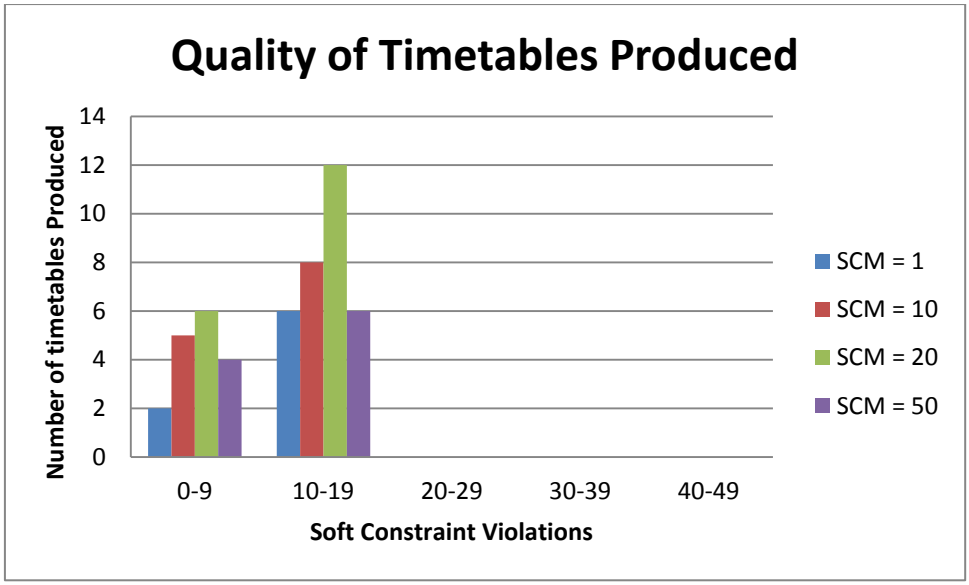


Figure 9.29: Frequency chart showing quality for different SCM values

Based on the success rates and the frequency chart, the DGA approach with an SCM of 20 performs better than any other of the SCM values tested. The DGA approach also produces the best quality timetable when using this parameter value (6 soft constraint violations). An SCM size of 20 will therefore be used.

9.3.4.2 Fine-tuning the population size

Initially, a lower population size of 50 was attempted, but only one feasible timetable was found from the initial runs and each run had an average of two hard constraint violations. The genetic algorithm is then run using population sizes of 100, 200 and 500 respectively. The performance of the algorithm using each population size is then compared. The processes and parameter values used when testing the different population sizes are listed in Table 9.108.

Table 9.108: Processes and parameter values to test best population size (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM size	20
Tournament size	10
Number of swaps	200
Number of generations	50

The success rates, average HC cost and average quality for the DGA using each population size is shown in Table 9.109.

Table 9.109: Results obtained for different population sizes

Success Rates		
Pop Size = 100 16.67%	Pop Size = 200 36.67%	Pop Size = 500 60.00%
Average HC Cost (and standard deviations)		
Pop Size = 100 1.6 (1.19)	Pop Size = 200 1.03 (1.13)	Pop Size = 500 0.63 (0.89)
Average Quality (and standard deviations)		
Pop Size = 100 11.80 (1.92)	Pop Size = 200 9.82 (2.23)	Pop Size = 500 10.83 (2.55)

Table 9.109 shows that there is a relationship between an increase in the success rate and an increase in the population size. The DGA approach with a population size of 500 produces the highest success rate followed by a population size of 200 and a population size of 100. In terms of timetable quality, the average number of soft constraint violations is lowest when using a population size of 200. The difference in quality when comparing population sizes of 200 and 500 is less than one soft constraint violation. The frequency chart below (Figure 9.30) provides more details in terms of timetable quality.

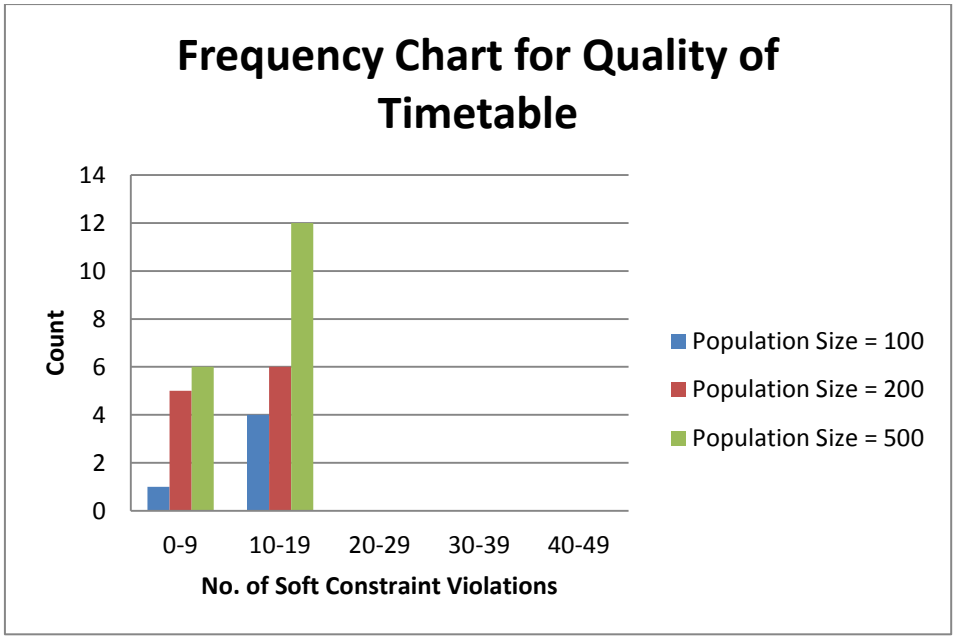


Figure 9.30: Frequency chart showing quality for various population sizes

The DGA approach with a population size of 500 produces six timetables that have between 4 and 9 soft constraint violations. The frequency chart also shows that the DGA approach with a population size of 500 produces more timetables of a higher quality than when using any other population size. For the Lewitt problem, a population size of 500 will be used.

9.3.4.3 Fine-tuning the tournament size

The DGA approach is tested using tournament sizes of 10 and 20. When testing these two values, the following processes and parameter values were kept constant:

Table 9.110: Processes and parameter values to test best tournament size (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM size	20
Population size	500
Number of swaps	200
Number of generations	50

The success rates and average quality of the timetables produced is shown in Table 9.111.

Table 9.111: GA approach performance when using two tournament sizes

Success Rates	
Tournament Size = 10	Tournament Size = 20
60.00%	30.00%
Average Quality (Feasible timetables)	
Tournament Size = 10	Tournament Size = 20
10.83	11.22

By observing the success rates, it is concluded that the DGA using a tournament size of 10 performs the best. The DGA approach using this tournament size produces the most feasible timetables. The low success rate when using a tournament size of 20 indicates that the selection pressure was too high. This resulted in the algorithm converging prematurely.

The frequency chart below (Figure 9.31) also shows that a sufficient number of high quality timetables are produced when using a tournament size of 10. The tournament size that will be used is therefore 10.

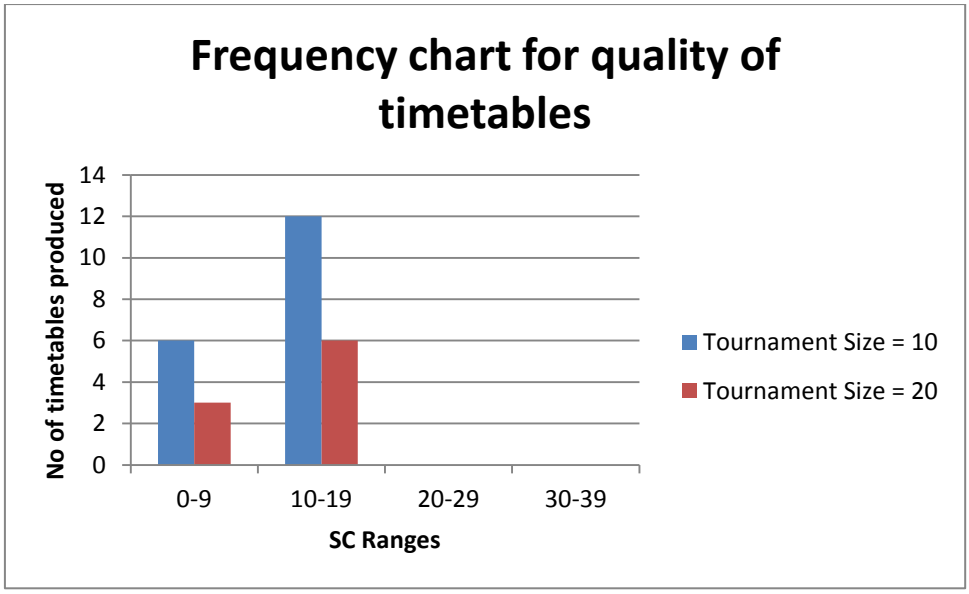


Figure 9.31: Frequency chart showing quality for various tournament sizes

9.3.4.4 Fine-tuning the number of swaps

This parameter sets the number of swaps performed by the mutation operator. The DGA approach is tested using four parameter values while other processes and parameter values were kept constant (see Table 9.112).

Table 9.112: Processes and parameter values to test best number of swaps (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM size	20
Population size	500
Tournament size	10
Number of generations	50

The success rates, average HC cost and average quality of the timetables produced when applying each of these parameter values are shown in the table below (Table 9.113).

Table 9.113: Results produced using various swap parameter values

Success Rates			
Swaps = 50 26.67%	Swaps = 75 16.67%	Swaps = 100 23.33%	Swaps = 200 60.00%
Average HC Cost (and standard deviations)			
Swaps = 50 1.27 (1.01)	Swaps = 75 1.23 (0.82)	Swaps = 100 1.2 (0.92)	Swaps = 200 0.63 (0.89)
Average Quality (and standard deviations)			
Swaps = 50 11.63 (4.00)	Swaps = 75 11.60 (2.3)	Swaps = 100 10.86 (1.57)	Swaps = 200 10.83 (2.55)

From Table 9.113, it can be seen that the highest success rate is obtained when the genetic algorithm is run using a swap parameter value of 200. The number of swaps also affects the quality of the timetables produced where the average soft constraint cost decreases as the swap parameter value increases. The frequency chart below (Figure 9.32) shows the distribution of timetables with regard to soft constraint violations.

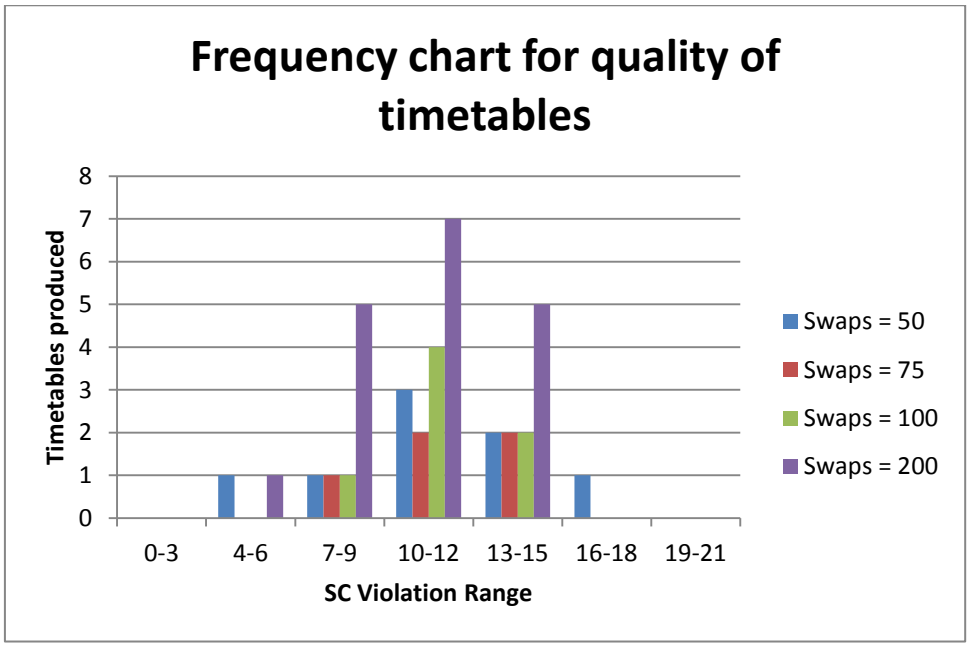


Figure 9.32: Frequency chart showing quality using various swap parameter values

Figure 9.32 shows a frequency diagram with timetables that range between 4 and 18 soft constraint violations. When using a DGA approach with a swap parameter value of 200, six out of the 18 timetables produced contain between 6 and 9 soft constraint violations. The timetable with the fewest soft constraint violations was found when using a swap parameter value of 50. Despite this, the swap parameter value to be used is 200 due to the high

success rate in producing feasible timetables and its ability to produce timetables of a high quality.

9.3.4.5 Maximum number of generations

This parameter indicates the number of generations for each phase. The performance of the DGA approach using each generation parameter value is shown in Table 9.114.

Table 9.114: Processes and parameter values to test best number of generations (Lewitt problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation Degree
Selection	Standard
Mutation	Hybrid
Phase 2	
Selection	Variant
Mutation	Random swap
Constant Parameter Values	
SCM size	20
Population size	500
Tournament size	10
Number of swaps	200

Table 9.115: Results produced using different generation parameter values

Success Rate		
Generations = 20 40.00%	Generations = 50 60.00%	Generations = 75 60.00%
Average Quality		
Generations = 20 11.17	Generations = 50 10.83	Generations = 75 10.83

When testing the DGA approach using 20 generations, the success rate and average quality calculated over the thirty runs is found to be 40% and 11.17 respectively. After 50 generations, the success rate and average quality is found to be 60% and 10.83 respectively. After 75 generations, the success rate and average quality of timetables are found to be the same as the success rate and average timetable quality of generation 50. This indicates that the algorithm converges between generations 20 and 50. The number of

generations is therefore set to 50 as timetable quality could not be improved any further after 50 generations.

9.3.5 The Woodlands secondary school timetabling problem

This section describes the results of fine-tuning the parameters of the DGA for the Woodlands secondary school timetabling problem. All parameter values except the SCM size are used for both Phase 1 and Phase 2 of the approach.

9.3.5.1 Fine-tuning the SCM size

The DGA approach is tested using four SCM sizes. While testing the different SCM values, all processes of the DGA and other parameter values are kept constant (see Table 9.116)

Table 9.116: Processes and parameter values to test best SCM Size (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation degree
Selection	Standard
Mutation	1VH
Phase 2	
Selection	Standard
Mutation	1 Violation swap with combination row swap
Constant Parameter Values	
Population size	500
Tournament size	10
Number of swaps	75
Number of generations	50

The success rates, average HC costs and average quality of timetables found when using each SCM value is shown in the table below (Table 9.117).

Table 9.117: Results using various SCM values

Success Rate			
SCM = 1 16.67%	SCM = 5 26.67%	SCM = 10 40.00%	SCM = 20 40.00%
Average HC Cost (and standard deviation)			
SCM = 1 2.8 (2.14)	SCM = 5 2.13 (1.66)	SCM = 10 1.8 (1.85)	SCM = 20 1.6 (1.61)
Average Quality (Feasible timetables)			
SCM = 1 4.60	SCM = 5 5.50	SCM = 10 5.00	SCM = 20 4.75

The DGA approach using SCM values of 10 and 20 produce the highest success rates. In terms of quality, the DGA using an SCM value of 20 produces slightly better quality timetables on average. The best possible timetable is also found when the DGA uses an SCM value of 20 (two soft constraint violations). The frequency chart (Figure 9.33) below shows the distribution of the quality of the timetables found.

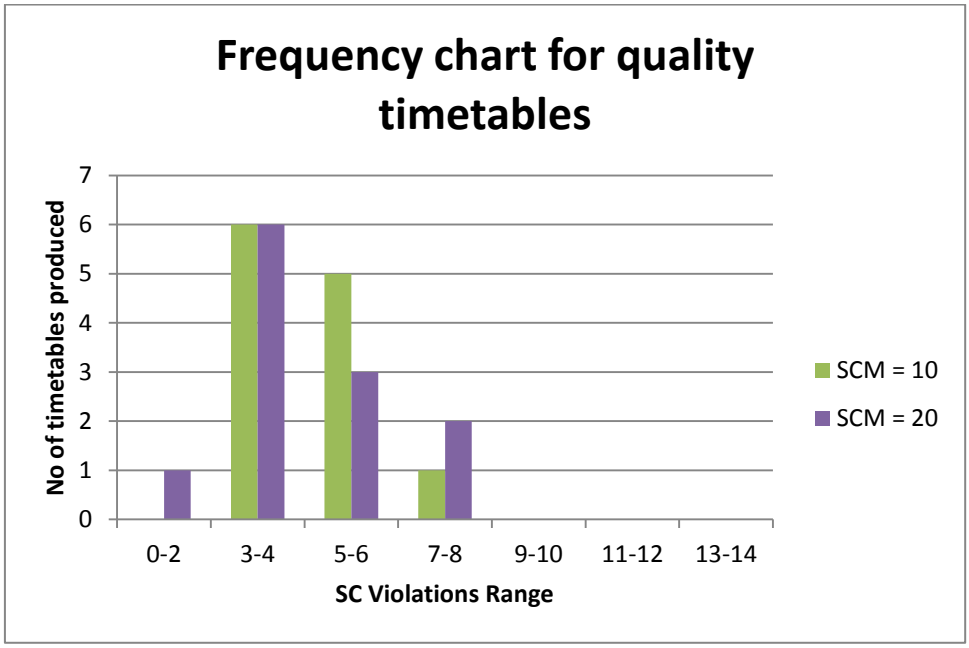


Figure 9.33: Frequency chart for the two best SCM values

As can be seen in the frequency chart, the DGA with an SCM value of 20 produces slightly more timetables of a higher quality. Only one timetable with two soft constraint violations is found. The best SCM parameter value from the tested values is 20 as the DGA approach using this parameter values produces the most number of feasible timetables. The quality of the timetables when the DGA uses this parameter value is also competitive when compared to the other SCM values.

9.3.5.2 Fine-tuning the population size

The algorithm is run using population sizes of 200, 500 and 750 respectively. Lower population sizes of 50 and 100 were initially tested and only two feasible timetables were induced (from ten runs) and each run had an average of 3 hard constraint violations. Table 9.118 lists the processes and parameter values that were kept constant when testing each of the population sizes of 200, 500 and 750.

Table 9.118: Processes and parameter values to test best population Size (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation degree
Selection	Standard
Mutation	1VH
Phase 2	
Selection	Standard
Mutation	1 Violation swap with combination row swap
Constant Parameter Values	
SCM size	20
Tournament size	10
Number of swaps	75
Number of generations	50

The success rates, average HC costs and average timetable quality found when running the DGA with each population size is shown in the table below (Table 9.119).

Table 9.119: Results using various population sizes

Success Rates		
Population Size = 200 26.67%	Population Size = 500 40.00%	Population Size = 750 46.67%
Average HC Cost (and standard deviations)		
Population Size = 200 2.47 (2.08)	Population Size = 500 1.6 (1.61)	Population Size = 750 1.33 (1.42)
Average Quality (and standard deviations) from feasible timetables		
Population Size = 200 5.00 (1.93)	Population Size = 500 4.75 (1.71)	Population Size = 750 4.36 (0.93)

As with the other problems addressed, Table 9.119 shows that an increase in the population size produces better success rates. An increase in the population size also shows an

improvement in the quality of the feasible timetables. The DGA using a population size of 750 produces the best quality timetables on average. Many of these timetables had between 3 and 4 soft constraint violations. While no optimal solutions (containing two soft constraint violations) are found, the increase in the success rate and the reduction of the average soft constraint cost results in a population size of 750 being chosen as the population size.

9.3.5.3 Fine-tuning the tournament size

To determine the best tournament size to use, the genetic algorithm is run using various tournament sizes.

Table 9.120: Processes and parameter values to test best tournament Size (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation degree
Selection	Standard
Mutation	1VH
Phase 2	
Selection	Standard
Mutation	1 Violation swap with combination row swap
Constant Parameter Values	
SCM size	20
Population size	750
Number of swaps	75
Number of generations	50

The following table shows the success rates, average HC costs and average timetable quality when applying each tournament size (Table 9.121).

Table 9.121: Results for various tournament sizes

Success Rates		
Tournament Size = 5 43.33%	Tournament Size = 10 46.67%	Tournament Size = 20 36.67%
Average HC Cost (and standard deviations)		
Tournament Size = 5 1.4 (1.5)	Tournament Size = 10 1.33 (1.42)	Tournament Size = 20 1.6 (1.52)
Average Quality (and standard deviations)		
Tournament Size = 5 4.46 (1.45)	Tournament Size = 10 4.36 (0.93)	Tournament Size = 20 4.45 (1.21)

The DGA produces more feasible timetables when using a tournament size of 10. The best quality timetables are also found when using this tournament size. The best timetable is found when using a tournament size of 5 (two soft constraint violations). The frequency chart showing the quality of timetables produced for each tournament size is displayed below in Figure 9.34.

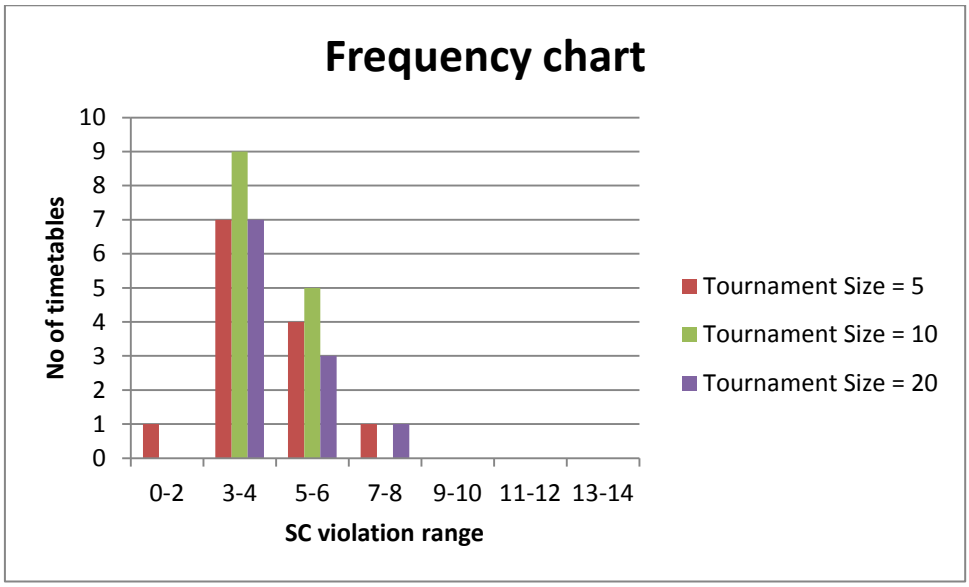


Figure 9.34: Frequency chart for various tournament sizes

The frequency chart shows that nine timetables containing between 3 and 4 soft constraint violations are found when using a tournament size of 10. This is higher than any of the other tournament sizes tested. In conclusion, a tournament size of 10 is found to be the best tournament size from the parameter values tested.

9.3.5.4 Fine-tuning the number of swaps

This parameter specifies the number of swaps that occur during mutation. The DGA is run using four different swap parameter values. The success rate, average hard constraint cost

and average timetable quality found for the DGA approach using each of the swap parameter values is shown in the table below (Table 9.123).

Table 9.122: Processes and parameter values to test best number of swaps (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation degree
Selection	Standard
Mutation	1VH
Phase 2	
Selection	Standard
Mutation	1 Violation swap with combination row swap
Constant Parameter Values	
SCM size	20
Population size	750
Tournament size	1
Number of generations	50

Table 9.123: Results using various swap parameter values

Success Rate			
Swaps = 50	Swaps = 75	Swaps = 100	Swaps = 150
46.67%	50.00%	63.33%	66.67%
Average HC Cost (and standard deviations)			
Swaps = 50	Swaps = 75	Swaps = 100	Swaps = 150
1.2 (1.24)	1.4 (1.59)	0.8 (1.13)	0.73 (1.11)
Average Quality (and standard deviations)			
Swaps = 50	Swaps = 75	Swaps = 100	Swaps = 150
4.36 (1.22)	5.47 (2.50)	4.58 (1.39)	4.60 (1.39)

The above table shows that an increase in the swap parameter value results in an increase in the number of feasible timetables produced. The DGA approach using a swap parameter value of 50 produces the best timetables in terms of average quality. The best possible timetable is also produced when using this parameter value i.e. a feasible timetable with two soft constraint violations. A similar timetable is also produced when the DGA is run using a parameter value of 100. When using a parameter value of 150, the DGA manages to produce two optimal timetables from the thirty runs conducted. The frequency diagram for the results produced is shown below (Figure 9.35).

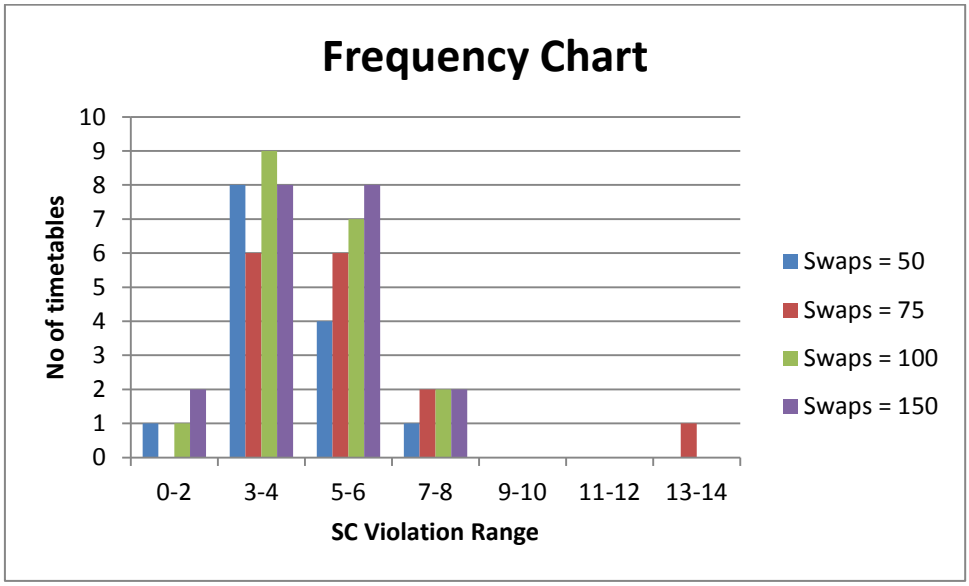


Figure 9.35: Frequency chart for varying number of swaps

The frequency chart shows that the DGA with 150 swaps produces the most number of timetables with soft constraint violations between 2 and 4. Based on the success rate and the frequency chart, the best number of swaps to use is 150.

9.3.5.5 Maximum number of generations

In order to determine the number of generations to use, the DGA approach is run using three different generation parameter values. Table 9.124 lists the processes and parameter values of the DGA that were constant when testing the maximum number of generations to use.

Table 9.124: Processes and parameter values to test best number of generations (Woodlands problem)

Constant Methods and Operators	
Phase 1	
Heuristic	Saturation degree
Selection	Standard
Mutation	1VH
Phase 2	
Selection	Standard
Mutation	1 Violation swap with combination row swap
Constant Parameter Values	
SCM size	20
Population size	750
Tournament size	1
Number of swaps	150

The performance of the DGA using each generation parameter value tested is shown in the table below (Table 9.125).

Table 9.125: Results for varying number of generations

Success Rates		
Gens = 20	Gens = 50	Gens = 75
43.33%	66.67%	66.67%
Average Quality		
Gens = 20	Gens = 50	Gens = 75
4.23	4.60	4.50

When the algorithm reaches 20 generations, the algorithm has still not converged as the success rate shows an increase in the number of feasible timetables after 20 generations. The success rates at generations 50 and 75 are the same, indicating that at some point after generation 50, the maximum number of feasible timetables have been produced. The average quality produced at 75 generations is slightly better than that of 50. This indicates that timetable quality may be improved after 50 generations. The number of generations is therefore set to 75.

9.3.6 Summary of fine-tuning

Table 9.126 shows the parameter values that were selected based on the tests conducted in section 9.3.

Table 9.126: Parameter values for each data set

Data Set	SCM	Population Size	Tournament Size	Swaps per Mutation	Generations
HDTT4	50	1000	10	200	50
HDTT5	50	1000	10	200	50
HDTT6	50	1000	10	200	50
HDTT7	50	1000	10	200	50
HDTT8	50	1000	10	200	50
Valouxis	50	1000	10	100	50
HS1 – HS7 (excluding HS5)	25	750	15	200	50
HS5	50	750	10	20	75
Lewitt	20	500	10	200	50
Woodlands	20	750	10	150	75

The sequential construction method is found to be useful in reducing the number of hard constraint violations of the initial population. An SCM value of at least 20 is recommended.

From the population sizes tested, a trend is observed where an increase in population size results in an increase in the success rate. Smaller population sizes of 100 and 50 were initially tested, but very few feasible timetables were induced. In addition, these timetables were of a poor quality. The advantages of an increase in the success rate for finding feasible timetables, combined with an improvement in timetables quality, conclude that a population size of at least 500 must be used.

The best tournament size to use will vary depending on the problem. For all problems, the DGA performs best when using a tournament size of either 10 or 15. In many cases, there was only a small difference in quality and success rate when comparing the DGA with different tournament sizes.

For most data sets, a trend is found where increasing the number of swaps results in an improvement in the success rate and the average quality of the timetables produced. From the tests conducted, a swap parameter value of at least 150 swaps per mutation is required. The only exception was data set HS5 where reducing the number of swaps improved the feasibility and quality of timetables produced.

For the generations limit, it was found that at least 50 generations are required before the algorithm converges. This value is also adequate for Phase 1 as feasible timetables are found before generation 50.

9.4 IGA – Results and discussion

This section reports on the performance of the IGA (described in Chapter 8) when applied to the five school timetabling problems. The performance of the IGA is compared to that of the DGA in order to determine the better approach and this was tested for statistical significance. In order to make the comparison of the two approaches as fair as possible, the parameter values used for the IGA are the same parameter values used by the DGA (see section 9.3).

Section 9.4.1 describes the IGA when applied to the Abramson school timetabling problem. Section 9.4.2 discusses the performance of the IGA approach when applied to the Valouxis Greek school timetabling problem. The Beligiannis Greek school timetabling problem is then covered in Section 9.4.3. Sections 9.4.4 and 9.4.5 respectively describe the results of applying the IGA to the W. A. Lewitt Primary and Woodlands Secondary school timetabling problems.

9.4.1 The Abramson School Timetabling Problem

The following table lists the parameter values and the instructions used to build timetables for the HDTT problem.

Table 9.127: Parameter values and instruction set used for Abramson problem

SCM	50
Population Size	1000
Tournament Size	10
Maximum Generations	50
Crossover Rate	80%
Mutation Rate	20%
Swaps per mutation	200
Instruction set	A, D, 3 (1VH), 4 (2VH)

Initially, the IGA contained non-hill climbing operators in the instruction set. However, the IGA did not perform well and the non-hill climbing operators were removed from the instruction set. The line chart below (Figure 9.36) illustrates how each instruction in a sample string affects the hard constraint cost of a partially complete timetable. In this line chart, the x-axis represents the instructions of the sample string and the y-axis is the hard constraint cost.

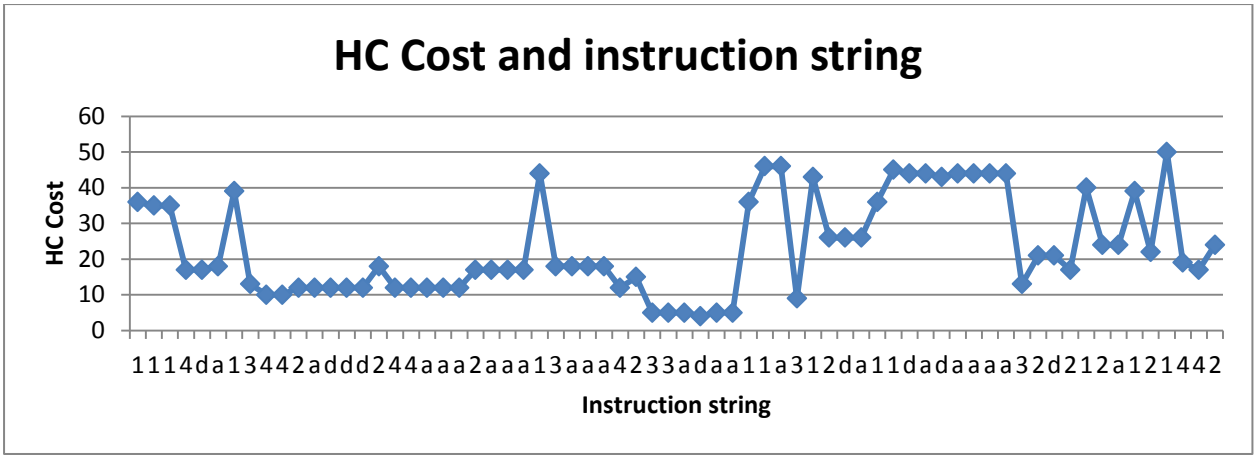


Figure 9.36: Effect of each instruction on the hard constraint cost (HDTT8)

In Figure 9.36, instructions 1 and 2 represent non-hill climbing mutation operators and instructions 3 and 4 represent hill climbing mutation operators. As can be seen, whenever instruction 1 (1VNH) is executed, the cost of the individual increases in most cases. In the case of the 2VNH operator, if the cost is above 30, then the operator improves the cost of the timetable. In cases where the cost is low (below 20), the 2VNH operator is found to increase the cost. Whenever the hill climbing operators (instructions 3 and 4) are applied, it was found that the cost was reduced. The above diagram represents only one sample string. Several strings were tested and similar results were found. It is also interesting to note that the DGA performed best when using a non-hill climbing operator (2VNH) while the IGA performs best when using hill climbing operators.

The success rates for the IGA and DGA (from section 9.2) when applied to each data set are listed in Table 9.128 below. The IGA induces feasible timetables for all runs.

Table 9.128: Success rates comparison – DGA vs IGA

Success Rates – DGA				
HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
100%	100%	100%	46.67%	13.33%
Success Rates – IGA				
HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
100%	100%	100%	100%	100%

While both algorithms perform well for data sets HDTT4, HDTT5 and HDTT6, the IGA approach outperforms the DGA for data sets HDTT7 and HDTT8. Hypothesis tests were

performed to test the statistical significance of the statement. The following table lists the hypotheses and Z-values:

Table 9.129: Hypothesis tests for HDTT7 and HDTT8

Data set	Hypothesis	Z-value
HDTT7	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	5.76
HDTT8	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	13.73

From the hypothesis tests conducted, it is concluded that the performance of the IGA is better than that of the DGA for data sets HDTT7 and HDTT8. This result is confirmed at all levels of significance.

9.4.2 The Valouxis School Timetable Problem

The parameter values listed in Table 9.130 were used when applying the IGA to the Valouxis problem. The table also shows the instruction sets that were used for Phase 1 and Phase 2 respectively.

Table 9.130: Parameter values and instruction set used for Valouxis problem

SCM	50
Population Size	1000
Tournament Size	10
Maximum Generations	50
Crossover Rate	80
Mutation Rate	20
Swaps per mutation	100
Instruction set – Phase 1	A, D, 3 (1VH), 4 (2VH)
Instruction set – Phase 2	A, D, 5 (Random swap), 6 (Row Swap), 7 (1V), 8 (2V)

Similar to the instruction set used for the Abramson problem, the non-hill climbing operators were not included in the Phase 1 instruction set as they performed poorly. This poor performance was found to be consistent with the performance of the DGA where the non-hill climbing operators performed poorly (see section 9.2.2.3). Figure 9.37 shows a line chart depicting the effect of each instruction (in a sample instruction string) on the hard constraint cost of the timetable.

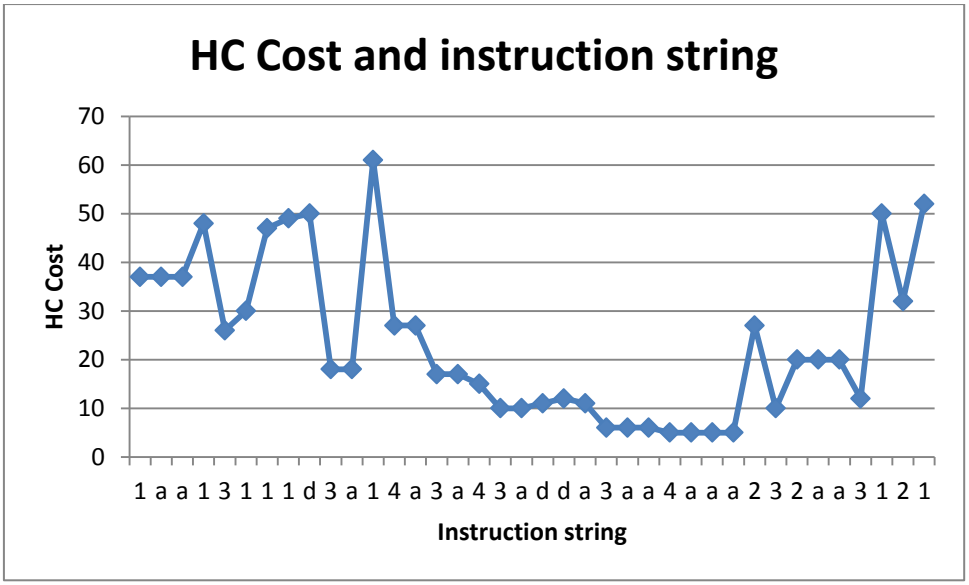


Figure 9.37: Effect of each instruction on the hard constraint cost (Valouxis problem)

In Figure 9.37, the cost of the timetable is reduced when using the hill climbing operators (instructions 3 and 4). This is shown in the middle portion of the instruction string that consists only of allocation, de-allocation and hill climbing mutation operators. The application of the non-hill climbing operators results in an increase in cost. Similar to the IGA, the 2VNH operator appears to improve the cost only when it is above 30 while the application of the operator results in an increase in the cost if it is below 10. The 1VNH operator performs poorly, resulting in an increase in the cost of the timetable whenever it is applied. Similar results were found for other strings that were tested. As a result of their poor performance, the non-hill climbing operators are removed from the instruction set. For Phase 2, no mutation operators were omitted as the quality of the timetables produced were found to be competitive.

Table 9.131 compares the performance of the IGA and DGA when applied to the Valouxis school timetabling problem. Thirty runs were conducted for each approach.

Table 9.131: Performance Comparison for the Valouxis Problem

	IGA	DGA
Success Rate	100%	100%
Average Quality	34	41.97
Standard Deviation	0	3.29
Best SC Cost	34	35

Both the IGA and DGA are able to induce feasible timetables for every run conducted. The difference in performance is found in terms of the quality of the timetables produced. The IGA is able to find an instruction string capable of reducing the soft constraint cost (the quality of the timetable) to 34 while the timetables produced by the DGA are found to average 41.97. The best timetable produced using the DGA contains 35 soft constraint violations.

A hypothesis test is conducted to determine the statistical significance of the above conclusion. A Z-value of 13.28 indicates that the IGA outperforms the DGA at all levels of significance.

9.4.3 The Beligiannis Greek School Timetabling Problem

Table 9.132 lists the parameter values and instruction sets that were used when the IGA is applied to the problem.

Table 9.132: Parameter values and instruction set used for Beligiannis problem

SCM	25
Population Size	750
Tournament Size	15
Maximum Generations	50
Crossover Rate	80
Mutation Rate	20
Swaps per mutation	200
Instruction set – Phase 1	A, D, 1 (1VNH), 2 (2VNH), 3 (1VH), 4 (2VH)
Instruction set – Phase 2	A, D, 5 (Random Swap), 6 (Row Swap), 7 (1V), 8 (2V)

Table 9.133 provides a summary of the performance of the IGA in terms of success rate, average quality and standard deviations (soft constraint cost). Similar to the DGA, the IGA was able to induce feasible timetables for all thirty runs conducted for each data set.

Table 9.133: Results summary for IGA applied to Beligiannis Problem

Success rates					
HS1	HS2	HS3	HS4	HS5	HS7
100%	100%	100%	100%	100%	100%
Average Quality (Average SC Cost)					
HS1	HS2	HS3	HS4	HS5	HS7
77.37	79.00	18.47	46.03	19.30	105.40
Standard deviations (SC Cost)					
HS1	HS2	HS3	HS4	HS5	HS7
6.51	8.12	0.86	1.52	4.11	4.95

Figure 9.38 displays a bar graph comparing the average quality of the timetables produced by both the IGA and the DGA when applied to the Beligiannis school timetabling problem.

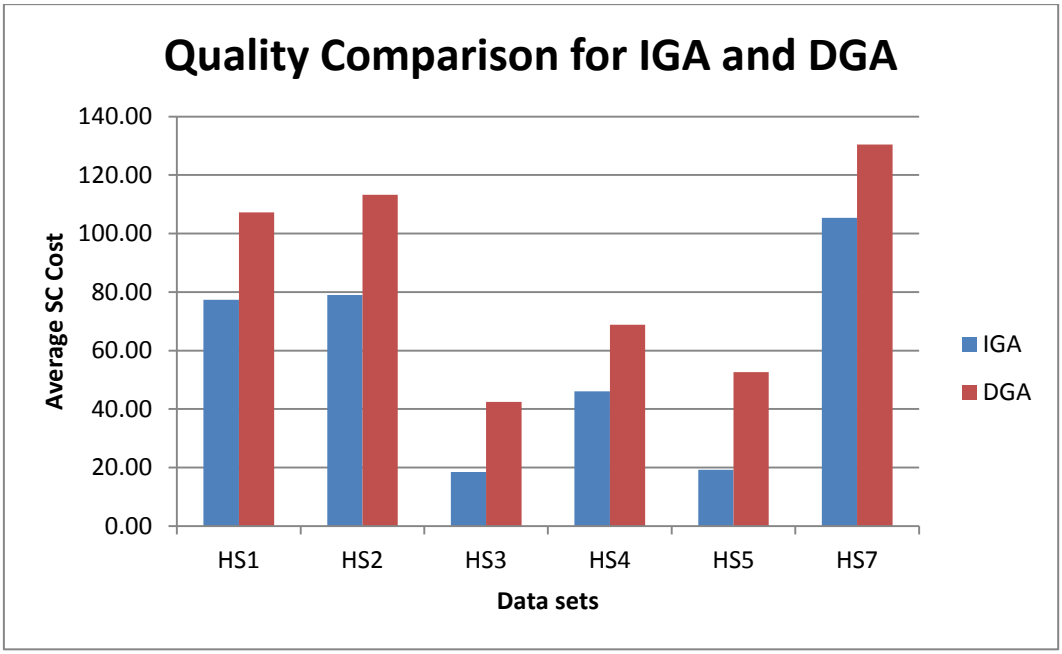


Figure 9.38: Average Timetable Quality induced by DGA and IGA

Figure 9.38 illustrates that, for all the data sets, the timetables produced by IGA have far fewer soft constraint violations on average than the timetables produced by the DGA approach. Hypothesis tests were conducted to test for the significance of this conclusion. Table 9.134 lists the hypotheses and associated Z-values.

Table 9.134: Hypothesis tests for quality

Data set	Hypothesis	Z-value
HS1	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	18.16
HS2	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	16.62
HS3	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	34.67
HS4	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	27.60
HS5	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	20.79
HS7	H_0: $\mu_{DGA} = \mu_{IGA}$; H_A: $\mu_{DGA} > \mu_{IGA}$	17.47

From the Z-values listed in the table above, it is concluded that the IGA induces better quality timetables than the DGA. This is confirmed at all levels of significance.

9.4.4 The Lewitt Primary School Timetabling Probem

The IGA was applied to the W.A. Lewitt Primary school timetabling problem using the following parameter values and instruction sets.

Table 9.135: Parameter values and instruction set used for Lewitt problem

SCM	20
Population Size	500
Tournament Size	10
Maximum Generations	50
Crossover Rate	80%
Mutation Rate	20%
Swaps per mutation	200
Instruction set – Phase 1	A, D, 2 (2VNH), 3 (1VH), 4 (2VH)
Instruction set – Phase 2	A, D, 5 (Random Swap), 7 (1V), 8 (2V)

The 1VNH operator was not included in the instruction set as it had performed poorly. This was found to be consistent with the tests conducted using the DGA (see section 9.2.4.3) where the 1VNH operator was the worst performing operator. Figure 9.39 shows the effect of each instruction (in a sample instruction string) on the hard constraint cost of a partially complete timetable.

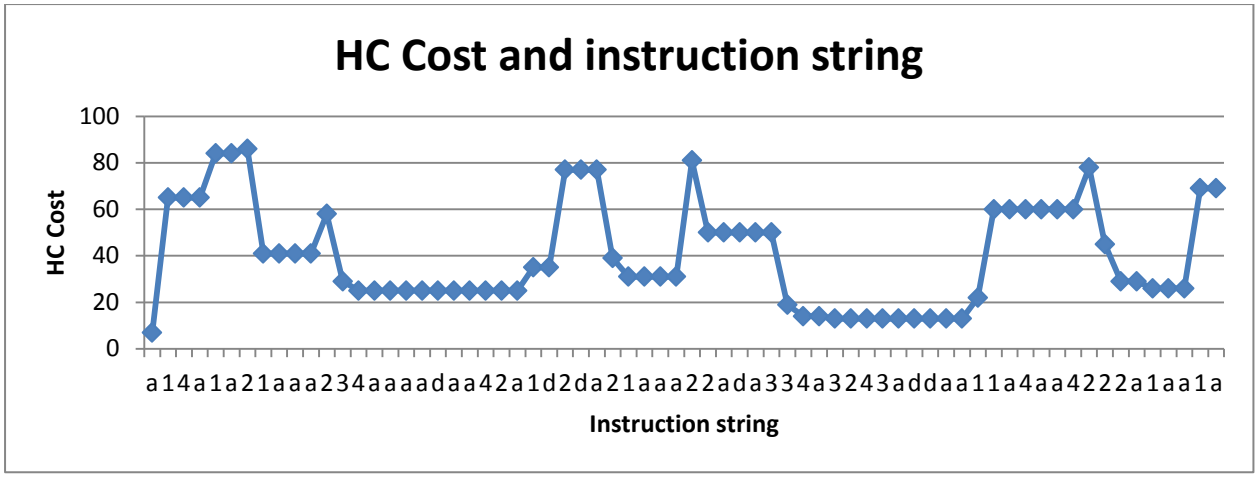


Figure 9.39: Effect of each instruction on hard constraint cost (Lewitt problem)

The hill climbing operators perform the best and reduce the hard constraint cost in most cases. The 1VNH operator seems to only improve the hard constraint cost when it is above 60. When applied at a point when the hard constraint cost is low, the 1VNH operator tends to increase the cost of the timetable. The 1VNH operator was therefore not included in the Phase 1 instruction set. The 2VNH operator was found to be effective when the constraint cost was high but was not able to further reduce constraint violations when the cost was below 20.

From the thirty runs conducted, 28 feasible timetables are found compared to the 18 feasible timetables found by the DGA. The two timetables that are not feasible have two hard constraint violations and one hard constraint violation respectively. In terms of quality, the feasible timetables average 7.61 soft constraint violations. This is better than the DGA which averages 10.83 soft constraint violations per timetable. A hypothesis test was conducted for feasibility. The hypothesis tested is that the IGA produces fewer hard constraint violations than the DGA. A Z-value of 2.90 is calculated, indicating that the performance of the IGA is significantly better than the DGA. A hypothesis test was not conducted for quality since there were not enough feasible timetables produced by the DGA.

9.4.5 The Woodlands secondary school timetabling problem

The IGA uses the following parameter values and instruction sets to solve the Woodlands Secondary school timetabling problem.

Table 9.136: Parameter values and instruction set used for Woodlands problem

SCM	20
Population Size	750
Tournament Size	10
Maximum Generations	50
Crossover Rate	80%
Mutation Rate	20%
Swaps per mutation	150
Instruction set – Phase 1	A, D, 1 (1VNH), 2 (2VNH), 3 (1VH), 4 (2VH)
Instruction set – Phase 2	A, D, 5 (Random Swap), 6 (Row Swap), 7 (1V), 8 (1V Row Swap)

Despite the poor performance of the non-hill climbing operators reported in section 9.2.5.3, the IGA managed to induce feasible timetables using an instruction set (Phase 1) consisting of all the mutation operators. The Phase 2 instruction set also incorporated all the soft constraint mutation operators tested with the DGA.

Table 9.137 summarizes the results that were found after conducting thirty runs using different seed values. This table also compares the success rates and average quality with that of the results obtained using the DGA approach.

Table 9.137: Results comparison for IGA and DGA

	IGA	DGA
Success Rate	100%	66.67%
Average quality	2.37	4.5
Standard deviation	0.56	1.36

The IGA achieves a 100% success rate and from the thirty feasible timetables, twenty of these have the minimum number of constraint violations of two. The IGA approach is far more effective than the DGA where only twenty out of the thirty runs produce feasible timetables and from these twenty feasible timetables, only two have the minimum number of soft constraint violations.

Hypothesis tests were conducted to evaluate the significance of these results. In terms of feasibility, a Z-value of 3.61 is obtained, allowing for the conclusion that the IGA approach performs better than the DGA approach.

9.5 Discussion of IGA versus DGA Results

From the results in section 9.4, it is concluded that the IGA is a far more effective approach for solving the school timetabling problem than the DGA and this was found to be statistically significant. Table 9.138 lists the average runtime, success rate, and average quality obtained when the DGA and IGA are applied to each problem.

Table 9.138: Summary of DGA and IGA performance for each problem

	DGA				IGA			
	Average Time	Best Time	Success %	Average Quality	Average Time	Best Time	Success %	Average Quality
HDTT4	10 mins	9mins	100%	NA	11 mins	10 mins	100%	NA
HDTT5	16 mins	15 mins	100%	NA	20 mins	26 mins	100%	NA
HDTT6	25 mins	25 mins	100%	NA	68 mins	1 hour	100%	NA
HDTT7	44 mins	42 mins	47%	NA	235 mins	4 hrs	100%	NA
HDTT8	65 mins	64 mins	13%	NA	678 mins	5 hrs	100%	NA
Valouxis	58 mins	58 mins	100%	41.97	669 mins	2 hrs	100%	34.00
HS1	133 mins	133 mins	100%	107.20	3 days	3 days	100%	77.37
HS2	131 mins	131 mins	100%	113.23	3 days	10 hrs	100%	79.00
HS3	46 mins	46 mins	100%	42.47	3 days	3 days	100%	18.47
HS4	53 mins	53 mins	100%	68.80	3 days	3 days	100%	46.03
HS5	18 mins	18 mins	100%	52.63	2 days	1.5 hrs	100%	19.30
HS7	202 mins	202 mins	100%	130.40	3 days	1 day	100%	105.40
Lewitt	269 mins	269 mins	60%	10.83	7 days	1 day	93%	7.61
Woodlands	1 day	1 day	67%	4.50	1.5 days	3 mins	100%	2.37

With the IGA, feasible solutions were quickly induced for the real world problems (Greek and South African schools) and the remaining time was spent in improving timetable quality. The increased runtimes for the IGA are mainly due to the evaluation process where a timetable is created or improved for every string (individual). The longer runtimes are justified due to the high success rates (when comparing HDTT7, HDTT8 and the Lewitt and Woodlands problem) as well as the vast improvement in timetable quality (for all real world problems).

One possible reason for the performance of the IGA is that the search space was reduced. The number of possible permutations of timetables for each problem is far greater than the number of permutations of different instruction strings available. Kazarlis et al. [KAZA07], Aickelin [AICK04] and Ross [ROSS94] also observed a reduction in the search space when using an indirect representation in other problem domains. Kazarlis et al. attributed this to a less constrained and complicated search space since the building of the schedule is not directly performed by the genetic algorithm. Further research will also involve investigating the search areas of the DGA and IGA respectively.

9.6 Comparison with Other Studies

This section compares the timetables produced by the DGA and the IGA approaches with timetables produced using other methods. Similar to studies by [SMIT03] and [BELI12], the comparisons made are empirical in nature. A thorough comparison is not possible as it is not known as to whether the methods that are being compared have been fine-tuned or not. Statistical tests cannot be performed for these comparisons as the studies have not provided sufficient data (averages and standard deviations) to perform these tests. The purpose of the comparison is to simply show that the IGA and DGA approaches are capable of inducing acceptable, high quality timetables.

For the Abramson problem, the performance of the IGA and DGA are compared to methods used in other studies (see section 9.6.1). For the Greek school timetabling problems, timetables were made available by Beligiannis et al. [BELI08] and Valouxis [VALO09] respectively and are compared to the timetables induced using the IGA and DGA. For the South African school timetabling problems, the timetables induced using the IGA and DGA are compared to the actual school timetables used by each school. In order to fairly assess the timetables, a common fitness function was used to evaluate the hard constraint and soft constraint cost.

9.6.1 Abramson benchmark problem set comparison

Table 9.139 lists the best cost (BC) and average cost (AC) of timetables created using different timetable construction techniques. The best cost indicates the lowest hard constraint cost and the average cost indicates the average number of hard constraint violations over twenty runs conducted for each technique. The last two rows in the table lists the performance of the approaches used in this study. The techniques listed in the table are:

- SA1 – A simulated annealing method implemented by [ABRA93].
- SA2 – A simulated annealing algorithm implemented by [RAND00].
- TS – A tabu search tested by [RAND00].
- GS – The greedy search method by [RAND00].
- NN-T2 – A neural network employed by [SMIT03].
- NN-T3 – A neural network employed by [SMIT03].
- Hybrid – A hybrid approach incorporating simulated annealing and a VLSN by [AVEL07].
- SA3 – A simulated annealing method implemented by [LIU09].
- DGA – Direct representation genetic algorithm used in this study.
- IGA – Indirect representation genetic algorithm used in this study.

Table 9.139: Results comparison for Abramson problem

Method	HDTT4	HDTT5	HDTT6	HDTT7	HDTT8
SA1	BC: Unknown AC: Unknown	BC: 0 AC: 0.67	BC: 0 AC: 2.5	BC: 2 AC: 2.5	BC: 2 AC: 8.23
SA2	BC: 0 AC: 0	BC: 0 AC: 0.3	BC: 0 AC: 0.8	BC: 0 AC: 1.2	BC: 0 AC: 1.9
TS	BC: 0 AC: 0.2	BC: 0 AC: 2.2	BC: 3 AC: 5.6	BC: 4 AC: 10.9	BC: 13 AC: 17.2
GS	BC: 5 AC: 8.5	BC: 11 AC: 16.2	BC: 19 AC: 22.2	BC: 26 AC: 30.9	BC: 29 AC: 35.4
HNN1	BC: 0 AC: 0.1	BC: 0 AC: 0.5	BC: 0 AC: 0.8	BC: 0 AC: 1.1	BC: 0 AC: 1.4
HNN2	BC: 0 AC: 0.5	BC: 0 AC: 0.5	BC: 0 AC: 0.7	BC: 0 AC: 1	BC: 0 AC: 1.2
Hybrid	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0.1	BC: 0 AC: 0.6
SA3	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0.4
DGA	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 1.06	BC: 0 AC: 1.73
IGA	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0	BC: 0 AC: 0

Based on the best cost and average cost obtained, the IGA performs best as this approach found feasible timetables for all runs conducted. The DGA approach is also very competitive, finding solutions for all data sets and performing better than the simulated annealing, tabu search and genetic search approaches. The neural network approaches perform slightly better than the DGA. Avella's Hybrid approach and the simulated annealing method used by [LIU09] produce timetables with fewer clashes (on average) than the DGA.

9.6.2 The Valouxis Greek school timetabling problem

A timetable created using a constraint programming approach is obtained from a paper by Valouxis et al. [VALO03]. No other techniques applied to this problem could be found. The first column of Table 9.140 lists each of the hard and soft constraints of the problem. The second, third and fourth columns indicate a count of the number of violations for each study.

Table 9.140: Comparison of timetables from GA approach and constraint programming approach

Hard Constraint	Constraint programming	DGA	IGA
Clashes	37	0	0
Free periods	0	0	0
Illegal teacher placements	4	0	0
Teacher average violations	9	0	0
Teacher class average violations	6	0	0
Total	**56**	**0**	**0**
Soft Constraints			
AM-PM preference violations	34	34	34
Teacher free periods	11	1	0
Total	**45**	**35**	**34**

As can be seed in Table 9.140, the timetable created using the constraint programming approach is unfeasible with 56 hard constraint violations. The majority of these violations are teacher clashes. The best timetable produced by both the DGA and the IGA contains no hard constraint violations and is thus feasible. Upon closer analysis of the timetable produced by [VALO03], it was found that the published timetable may be incorrect. The schedules of two of the classes are exactly the same resulting in 35 of the 37 clashes. In terms of quality, both the IGA and DGA produce several better quality timetables than the timetable induced using constraint programming. This was the only timetable that was available and attempts to contact the author to request other timetables were unsuccessful.

9.6.3 The Beligiannis Greek school timetabling problem

Beligiannis et al. [BELI08] used an evolutionary algorithm approach to solve the Greek high school timetabling problem and introduces seven data sets. The best timetable obtained from tests conducted and described in section 9.3.3.5 (DGA) and section 9.4 (IGA) is compared with the sample timetables that were made available by [BELI08].

9.6.3.1 Data set HS1

Table 9.141 lists the number of hard and soft constraint violations for the timetables produced using each method.

Table 9.141: Comparison of GA approaches and Beligiannis results (HS1)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	0	0	0
Co-teaching and Subclasses	0	0	0
Total	**0**	**0**	**0**
Soft Constraints			
Gaps	31	11	4
Teacher gap distribution	24	8	0
Teacher daily gap distribution	5	1	3
Daily average	66	57	47
Repeats	13	19	9
Total	**139**	**96**	**63**

The timetables produced using all three methods are feasible (no hard constraint violations). The better timetable is determined by evaluating the number of soft constraint violations and as can be seen, the timetable produced using the IGA has the fewest number of soft constraints. This timetable has 33 fewer soft constraint violations than the DGA and 76 fewer soft constraint violations than the evolutionary algorithm [BELI08].

9.6.3.2 Data set HS2

Table 9.142 shows the comparison of the timetables produced using the evolutionary algorithm by [BELI08] and the two approaches used in the current study.

Table 9.142: Comparison of GA approaches and Beligiannis results (HS2)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	0	0	0
Co-teaching and Subclasses	0	0	0
Total	**0**	**0**	**0**
Soft Constraints			
Gaps	42	12	0
Teacher gap distribution	21	7	0
Teacher daily gap distribution	11	2	0
Daily average	83	65	56
Repeats	18	13	10
Total	**175**	**99**	**66**

Once again, the evolutionary algorithm and both the DGA and IGA produce feasible timetables with no hard constraint violations. In terms of quality, the DGA approach produces 76 fewer soft constraint violations than the evolutionary algorithm. The IGA produced the best of the three timetables with 66 soft constraint violations. From these 66 violations, there were no free periods i.e. all teacher timetables were compact. This timetable (found by the IGA) also had the fewest daily average violations and the fewest repeat lessons.

9.6.3.3 Data set HS3

Table 9.143 lists the number of violations for each timetable produced by the three different approaches. This data set contained no co-teaching and subclass requirements.

Table 9.143: Comparison of GA approach and Beligiannis results (HS3)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	0	0	0
Co-teaching and Subclasses	NA	NA	NA
Total	**0**	**0**	**0**
Soft Constraints			
Gaps	17	2	0
Teacher gap distribution	13	2	0
Teacher daily gap distribution	4	0	0
Daily average	24	19	13
Repeats	3	11	4
Total	**61**	**34**	**17**

All three approaches found feasible timetables when applied to data set HS3. In terms of quality, the IGA produced the best timetable with half the number of violations as the DGA. Similar to the HS2 data set, the IGA finds a timetable with no free periods. The timetable induced by the DGA contained the most number of repeat lesson violations. The evolutionary algorithm by [BELI08] produced a timetable with the fewest number of repeats.

9.6.3.4 Data set HS4

This section compares the performance of the three approaches when applied to the HS4 data set.

Table 9.144: Comparison of GA approach and Beligiannis results (HS4)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	0	0	0
Co-teaching and Subclasses	0	0	0
Total	**0**	**0**	**0**
Soft Constraints			
Gaps	32	5	0
Teacher gap distribution	14	3	0
Teacher daily gap distribution	10	1	0
Daily average	46	45	38
Repeats	0	5	5
Total	**102**	**59**	**43**

The timetables produced by all three approaches are feasible and can be used by the schools. When observing the soft constraint violations, the IGA finds compact timetables for all teachers (no free periods). The timetable found using the evolutionary algorithm contains no repeats, but the most number of free periods. Overall, the timetable induced by the IGA was of the best quality.

9.6.3.5 Data set HS5

This section compares the timetables produced when applying the approaches to data set HS5. This data set is similar to data set HS3, as there are no co-teaching and subclass requirements.

Table 9.145: Comparison of GA approach and Beligiannis results (HS5)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	0	0	0
Coteaching and Subclasses	NA	NA	NA
Total	**0**	**0**	**0**
Soft Constraints			
Gaps	8	5	0
Teacher gap distribution	8	3	0
Teacher daily gap distribution	0	1	0
Daily average	27	24	14
Repeats	0	7	1
Total	**43**	**40**	**15**

Once again, the timetables produced by all three approaches are found to be feasible with zero hard constraint violations. The difference in quality between the timetable produced by the evolutionary algorithm ([BELI08]) and the timetable produced by the DGA is just three soft constraint violations in favour of the DGA. For this data set, the timetable produced using the DGA contains seven repeat violations while the evolutionary algorithm by [BELI08] manages to prevent any repeat violations and provides an even distribution of teacher daily free periods. Overall, the timetable produced by the IGA has the best quality. This timetable contains no free periods, 14 daily average violations and only one repeat lesson violation.

9.6.3.6 Data set HS7

This final section compares the timetables produced by the three approaches when applied to the HS7 data set. This data set was the largest of the data sets, containing the most number of classes and teachers.

Table 9.146: Comparison of GA approach and Beligiannis results (HS7)

	Beligiannis	DGA	IGA
Hard Constraint			
Total clashes	0	0	0
Last period violations	0	0	0
Unavailability violations	52	0	0
Co-teaching and Subclasses	18	0	0
Total	**70**	**0**	**0**
Soft Constraints			
Gaps	58	10	35
Teacher gap distribution	25	7	1
Teacher daily gap distribution	18	1	5
Daily average	99	81	51
Repeats	26	18	6
Total	**226**	**117**	**98**

The timetable produced using the DGA and IGA approaches were feasible while the sample timetable provided by [BELI08] was found to be unfeasible due to several unavailability violations as well as co-teaching and subclasses requirements that were not fulfilled. The timetable produced by the DGA approach was found to be feasible and was found to have 117 soft constraint violations. The timetable found by the IGA was again found to be of the highest quality with 98 soft constraint violations. The timetable provided by the author may be incorrect as [BELI08] states that feasible timetables are found for all data sets.

9.6.4 W.A. Lewitt primary school problem

A comparison is made between the timetable produced by the DGA and IGA approaches in this study and the actual timetable (developed using commercial software) used by the school. The actual timetable used by Lewitt has no clashes but does have one double period violation where the double period has to be split into two separate single periods during the day. Despite this violation, the school still chose to use the timetable. The timetable produced by the DGA and the IGA approaches have no clashes and all double period lessons are correctly allocated. In terms of quality, the actual timetable and the timetable produced using the DGA are of a similar quality with six constraint violations. The timetable produced using the IGA was of a better quality and had 3 soft constraint violations.

The main problem that faces the staff of W. A. Lewitt School is that the system currently employed by the school does not cater for double periods. These double periods have to be placed manually by the staff member. As a result, the staff member is also required to manually move tuples around in order to accommodate the double periods and to prevent resultant clashes. Both the DGA and the IGA developed in this study address all hard constraint requirements and the quality of the timetables produced is similar, if not better than that of the timetable used by the school.

9.6.5 Woodlands data set

A comparison is made between a timetables produced by the DGA, the IGA and the actual timetable (developed using commercial software) used by Woodlands secondary school. All three timetables were feasible. The timetables were also of an equivalent quality with only two soft constraint violations.

9.7 Chapter summary

This chapter firstly covers the performance of the DGA and its effect on each of the school timetabling problems when changing the different processes (initial population generation, selection methods and genetic operators). The chapter then describes the process of fine-tuning the DGA. The performance of the IGA is then covered and the performance of this approach is compared to that of the DGA. Finally, timetables obtained using the two genetic algorithm approaches are compared to timetables created using other techniques from the literature or to actual school timetables.

Chapter 10 - Conclusions and Future Research

10.1 Introduction

This chapter provides the overall conclusions based on the findings of the study. Two objectives were outlined in Chapter 1 of this thesis. Section 10.2 presents each objective and conclusions based on the results presented in Chapter 9. The chapter ends by discussing future extensions to the research presented in this thesis (section 10.3).

10.2 Objectives and Conclusions

Objective: Based on the analysis of the literature, implement a genetic algorithm approach for solving the school timetabling problem and evaluate it on more than one type of school timetabling problem.

Conclusion:

A genetic algorithm approach was developed based on an analysis of the literature. This genetic algorithm approach, referred to in this study as a DGA due to using a direct representation, was able to induce feasible timetables for five different school timetabling problems. The timetables were of a high quality when compared to timetables produced using other methods.

The study showed that the genetic algorithm approach performed better when construction heuristics are used to guide the search when creating the initial population. A sequential construction method (SCM) was also used and in some cases, it was found that the use of the SCM not only assisted in finding feasible solutions but also improved the quality of the timetables produced.

A variant tournament selection (VTS) was introduced and was based on standard tournament selection, but gave an opportunity for weaker individuals in the tournament to be selected as parents. In many cases, this selection method was found to have performed better with the DGA than standard tournament selection, especially during Phase 1 of the DGA.

When solving the Woodlands, Lewitt, Valouxis and Beligiannis school timetabling problems, it was found that the use of the mutation operators with hill climbing was necessary in order to produce feasible timetables. Similar results were reported in the literature where hill climbing improved the performance of the genetic algorithm. The type of mutation operator to use varied for each problem.

A conclusion can be made that different construction heuristics (primary and secondary), different selection methods and different mutation operators were required to produce the best results for each problem.

Objective: To evaluate a genetic algorithm that uses an indirect representation when solving the school timetabling problem.

Conclusion:

It was found that the IGA performed well and produced feasible solutions. It also outperformed the DGA in terms of both finding feasible timetables and producing better quality timetables. One possible reason that was hypothesized was that the IGA had a smaller search space to explore than the DGA as the IGA had fewer combinations to create an instruction string than when considering the different combinations of tuple allocations when creating a timetable.

10.3 Future research

Based on the research, future extensions of this work will include the following:

- Previous studies (discussed in Chapter 2) have classified the examination timetabling problem as a multi-objective problem. This involved the grouping of soft constraints in order to reduce the complexity of the problem. Future research will look at addressing the school timetabling problem as a multi-objective problem.
- From the results described in Chapter 9, it was found that for the DGA, each problem requires different sets of heuristics, selection methods, genetic operators and control parameter values in order to produce feasible, high quality timetables. Determining the best processes and the fine-tuning of the DGA control parameters was performed using a “by hand” (manual) approach. Future work would look at automating this process using a meta-genetic algorithm as described by Eiben [EIBE99] in order to find the best possible combination of initial population generation method, selection

method, mutation operators and parameter values. Future work would also investigate the use of parameter tuning tools such as F-Race.

- From the results in Chapter 9, it was concluded that the IGA performed better than the DGA. One possible reason identified for the better performance was the size of the search space explored by the two approaches. Another area of investigation would be the difference in the fitness landscape when using each of the approaches.
- The GA approaches developed during this study is for research purposes only. The staff of primary or secondary school may have difficulty using the program as they would require a basic background in genetic algorithms in order to run the program and configure the parameter settings. Future development will include the development of a user interface and an interactive timetabling system that allows the user to play a role in the development of the timetable. Examples of user roles would include:
    - specifying the data set,
    - choosing from a list of constraints that the algorithm must consider,
    - applying weights or priorities to the constraints,
    - pre-allocation of specific tuples to the timetable,
    - the adjustment of a generated timetable to suit their preferences,
    - the choice of timetables from a population of feasible timetables,
    - printing the chosen solution as a class, teacher, venue and/or student timetable.
- The Abramson benchmark problem is a common problem that has been solved using a variety of techniques such as tabu search, simulated annealing and neural networks. Future research would include using these techniques to solve the other school timetabling problems discussed in this study. The performance of these techniques can then be compared to the DGA and IGA approaches.

Bibliography

[1]	[ABDE10]	T. F. Abdelmaguid. Representations in Genetic Algorithm for the Job Shop Scheduling: A Computational Study. In *Software Engineering and Applications*. Volume 3, Pages 1155-1162. 2010.
[2]	[ABRA91a]	D. Abramson, J. Abela. A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In *Proceedings of the 15th Australian Conference: Division of Information Technology.* Pages 1-11. 1991.
[3]	[ABRA91b]	D. Abramson. Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms. In *Management Science*. Vol. 37 No 1. Pages 98-113. 1991.
[4]	[ABRA93]	D. Abramson, H. Dang. School Timetable: a case study in simulated annealing. In *Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems.* Chapter 5. Pages 103-124. 1993.
[5]	[AFFE09]	M. Affenzeller, S. Winkler, S. Wagner, A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall/CRC. Boca Raton, Florida. 2009. ISBN 978-1-58488-629-7.
[6]	[ALVA96]	R. Alvarez-Valdes, G. Martin, J. M. Tamarit. Constructing Good Solutions for the Spanish School Timetabling Problem. In *The Journal of the Operational Research Society, Vol. 47, No. 10.* Pages 1203-1215. October 1996
[7]	[AVEL07]	P. Avella, B. D'Auria, S. Salerno, I. Vasil'ev. A computational study of local search algorithms for Italian high-school timetabling. In *The Journal of Heuristics, Vol. 13, No. 6*. Pages 543-556. December 2007
[8]	[BANZ98]	W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone. *Genetic Programming: An introduction on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc. 1998. ISBN 1-55860-510-X.
[9]	[BART99]	R. Bartak. Constraint Programming: In pursuit of the Holy Grail. In *Proceedings of the Week of Doctoral Students (WDS99)*. Part IV. MatFyz Press. Prague. Pages 555-564. June 1999.
[10]	[BEAS93]	D. Beasley, D. R. Bull, R. R. Martin. An Overview of Genetic Algorithms: Part 1 and Part 2, Research Topics. In *University*

		Computing 15 (4). Pages 170-181. UCISA. 1993.
[11]	[BEDO04]	C. F. Bedoya, M. Santos. A Non-Standard Genetic Algorithm Approach to Solve Constrained School Timetabling Problems. In *Computer Aided Systems Theory – Eurocast 2003, Lecture notes in Computer Science.* Vol. 2809. Pages 26-37. 2004
[12]	[BELI08]	G. N. Beligiannis, C. N. Moschopoulos, G. P. Kaperonis, S. D. Likothanassis. Applying evolutionary computation to the school timetabling problem: The Greek Case. In *Computers and Operations Research.* Vol. 35. Pages 1265-1280. 2008.
[13]	[BELI12]	I. X. Tassopoulos, G. N. Beligiannis. Solving Effectively the School Timetabling Problem Using Particle Swarm Optimization. In *Expert Systems with Applications 39*. Pages 6029-6040. 2012.
[14]	[BELL08]	G. S. Bello, M. C. Rangel, M. C. S. Boeres. An Approach for the Class/Teacher Timetabling Problem using Graph Coloring. In *The Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT).* 2008.
[15]	[BIRB97]	T. Birbas, S. Daskalaki, E. Housos. Timetabling for Greek High Schools. In *The Journal of the Operational Research Society*, *Vol. 48, No. 12*. Pages 1191-1200. December 1997.
[16]	[BLIC95]	T. Blickle, L. Thiele. A Comparison of Selection Schemes used in Genetic Algorithms. In TIK-Report, No 11. December 1995.
[17]	[BRUC95]	W. S. Bruce. The Application of Genetic Programming to the Automatic Generation of Object-Oriented Programs, Phd Dissertation, School of Computer and Information Sciences, Nova Southeastern University, 1995.
[18]	[BUFE01]	M. Bufe, T. Fisher, H. Gubbels, C. Hacker, O. Hasprich, C. Scheibel, K. Weicker, N. Weicker, M. Wenig, C. Wolfangel. Automated Solution of a highly constrained school timetabling problem – preliminary results. In *Applications of Evolutionary Computing, Lecture notes in Computer Science*. Vol. 2037. Pages 431-440. 2001
[19]	[BURK08]	E. K. Burke, B. McCollum, P. McMullan, A. J. Parkes. Multi-objective Aspects of the Examination Timetabling Competition Track. In *Proceedings of PATAT 2008*. Montreal, Canada. 2008.
[20]	[BUSE03]	F. Busetti. *Simulated Annealing overview*. 2003
[21]	[CALD97]	J. P. Caldeira, A. C. Rosa. School Timetabling using Genetic Search. In *The proceedings of the International Conference on the Practice and*

		Theory of Automated Timetabling (PATAT). Pages 115-122. 1997.
[22]	[CART96]	M. W. Carter, G. Laporte. Recent Developments in Practical Examination Timetabling. In *Practice and Theory of Automated Timetabling: Lecture Notes in Computer Science*. Pages 1-21. 1996.
[23]	[CERD08]	A. Cerdeira-Pena, L. Carpente, A. Farina, D. Seco. New approaches for the school timetabling problem. In *Proceedings of the 7th Mexican International Conference on Artificial Intelligence*. Pages 261-267. 2008.
[24]	[COEL07]	C. A. Coelho, G. B. Lamont, D. A. Van Veldhuisen. Evolutionary Algorithms for Solving Multi-Objective Problems. Pages 556 – 557. Springer. ISBN: 9780387367972. 2007.
[25]	[COLO98]	A. Colorni, M. Dorigo, V. Maniezzo. In Metaheuristics for High School Timetabling. In *Computational Optimization and Applications.* Vol 9. Pages 275-298. 1998.
[26]	[DALT11]	J. Dalton. Roulette Wheel Selection. Available: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/ Accessed: 2010 Last Updated: May 2011
[27]	[DAVI91]	L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, New York. 1991. ISBN 0-442-00173-8.
[28]	[DEHA07]	P. de Haan, R. Landman, G. Post, H. Ruizenaar. A Four-Phase Approach to a Timetabling Problem in Secondary Schools. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. 2007.
[29]	[DESE06]	T. Desef, A. Bortfeldt, H. Gehring. A Tabu Search Algorithm for Solving the Timetabling-Problem for German Primary Schools (Abstract). In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT).* 2006
[30]	[DIAZ07]	P. A. Diaz-Gomez, D. F. Hougen. Initial Population for Genetic Algorithms: A Metric Approach. In *Proceedings of the International Conference on Genetic and Evolutionary Methods*. Pages 43-49. 2007.
[31]	[DIST01]	C. Di Stefano, A. G. B. Tettamanzi. An Evolutionary Algorithm for solving the School Time-Tabling Problem. In *Applications of Evolutionary Computing, Lecture notes in Computer Science*. Vol. 2037. Pages 452-462. 2001.
[32]	[DYER08]	D. W. Dyer. Evolutionary Computation in JAVA: A Practical Guide to

		the Watchmaker Framework. Available: http://watchmaker.uncommons.org/manual/index.html. Accessed: 2012 Last Updated: 2008
[33]	[EIBE94]	A. E. Eiben, P-E. Raue, Zs. Ruttkey. Genetic Algorithms with multi-parent recombination. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature. LNCS 866*. Springer-Verlag. Pages 78-87. 1994.
[34]	[EIBE95]	A. E. Eiben, C. H. M. van Kemenade. Performance of multi-parent crossover operators on numerical function optimization problems. In *Technical Report TR-95-33*. Leiden University. 1995.
[35]	[EIBE99]	A. Eiben, R. Hinterding, Z. Michalewicz. Parameter Control in Evolutionary Algorithms. In *Evolutionary Computation IEEE Transactions on 3.2*. Pages 124-141. 1999.
[36]	[EIBE03]	Introduction to evolutionary computing. Augusto Eiben and James Smith. Springer. ISBN: 9783540401841. 2003
[37]	[ELMO98]	S. Elmohamed, G. Fox, P. Coddington. A Comparison of Annealing Techniques for Academic Course Scheduling. In *Proceedings of the 2nd International Conference on the Practice and Theory of Aumtomated Timetabling*. Pages 146-166. Syracuse, New York. 1998.
[38]	[FERN99]	C. Fernandes, J. P. Caldeira, F. Melicio, A. Rosa. High School Weekly Timetabling by Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Algorithm.* Page 1777. 1999.
[39]	[FILH01]	G. R. Filho, L. A. N. Lorena. A Constructive Evolutionary Approach to School Timetabling. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing, Lecture Notes in Computer Science.* Vol 2037. Pages 130-139. 2001.
[40]	[FROS94]	D. Frost, R. Dechter. Dead-End Driven Learning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.* Pages 294-300. Seattle. 1994.
[41]	[GHAN03]	R. Ghanea-Hercock. *Applied Evolutionary Algorithms in Java*. Springer-Verlag. New York. 2003. ISBN 0-387-95568-2.
[42]	[GOLD89]	D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Westley. 1989. ISB 0-201-15767-5.
[43]	[GOLE89]	D. E. Goldberg, B. Korb, K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. In *Complex Systems* 3 (5).

		Pages 493-530. 1989. ISSN 0891-2513.
[44]	[GLOV89]	F. Glover. Tabu Search. Part 1. *ORSA Journal of Computing*. Pages 190-206. 1989.
[45]	[GLOV97]	F. Glover, M. Laguna. *Tabu Search*. Kluwer Academic Publishers. Netherlands. 1997. ISBN 0-7923-8187-4.
[46]	[GREF86]	J. J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. In IEEE Transactions on Systems, Man and Cybernetics. Vol SMC16 No. 1. Pages 122-128. January 1986.
[47]	[GROB02]	M. Grobner, P. Wilke, S. Buttcher. A Standard Framework for Timetabling Problems. In *Springer-Verlag Lecture Notes in Computer Science.* Vol 2740. Page 25. 2002.
[48]	[GUTI02]	G. Gutierrez, J. M. Molina, I. Galvan, A. Sanchiz. An Objective Measure to Compare some Automatic Generation Methods of NN Architectures. In *Systems, Man and Cybernetics*. Volume 3. 2002.
[49]	[HILL99]	R. R. Hill. A Monte Carlo study of Genetic Algorithm Initial Population Generation Methods
[50]	[HINT96]	R. Hinterding, Z. Michalewicz, T. C. Peacher. Self-adaptive genetic algorithms for numeric functions. In *Parallel Problem Solving from Nature, PPSN IV*. Pages 420-429. 1996.
[51]	[JACO06]	F. Jacobsen, A. Bortfeldt, H. Gehring. Timetabling at German Secondary Schools: Tabu Search verses Constraint Programming. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Pages 439-442. 2006.
[52]	[JOHN11]	C. Johnson. Basic Research skills in Computer Science. Available: http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/basics.html Retrieved: 2011
[53]	[KAZA07]	S. Kazarlis, V. Petridis, P. Adamidis, P. Fragkou. Evolutionary Timetabling with a Priority-Based Indirect Representation. In *Proceedings of the 22nd European Conference on Operational Research.* EURO XXII. Prague. July 8-11 2007.
[54]	[KENN95]	J. Kennedy, R. Eberhart. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks.* IV. Pages 1942-1948. 1995
[55]	[KING05]	J. H. Kingston. A Tiling Algorithm for High School Timetabling. In *Practice and Theory of Automated Timetabling V, Lecture Notes in*

		Computer Science. Vol. 3616. Pages 208-225. 2005.
[56]	[KIRK83]	S. Kirkpatrick, C. D. Gelatt Jr., M. Vecchi. Optimization by Simulated Annealing. *Science*. 220(4598). Pages 671-680. 1983.
[57]	[LUKE12]	S. Luke. *Essentials of Metaheuristics*. Lulu. Available at http://cs.gmu.edu/~sean/book/metaheuristics/. 2012. ISBN: 978-0-557-14859-2.
[58]	[LIU09]	Y. Liu, D. Zhang, S. C. H. Leung. A Simulated Annealing Approach with a new Neighbourhood Structure for the Timetabling Problem. In *Proceedings of GEC 2009, First ACM/SIGEVO Summit on Genetic and Evolutionary Computing.* Pages 381-386. 2009.
[59]	[MAHF95]	S. W. Mahfoud. Niching Methods for Genetic Algorithms. Phd Dissertation. Department of General Engineering, University of Illinois. 1995.
[59]	[MART07]	M. Marte. Towards constraint-based school timetabling. In *Annuals of Research*. Vol. 155. No.1. Pages 207-225. November 2007
[60]	[MEFF10]	K. Meffert. A Brief Introduction to Genetic Algorithms. Available: http://jgap.sourceforge.net/doc/gaintro.html Retrieved: 2011. Last Updated: 2010
[61]	[MICH98]	M. Michell. *An Introduction to Genetic Algorithms*. MIT Press. 1998. ISBN: 9780262631853.
[62]	[MILL95]	B. L. Miller, D. E. Goldberg. Genetic Algorithms, Tournament Selection and the Effects of Noise. In *Complex Systems*. Pages 193-212. 1995.
[63]	[NAID09]	A. Naidoo. Evolving Automata using Genetic Programmings. Masters Thesis. School of Computer Science, University of Kwazulu Natal. 2009
[64]	[NELS91]	M. M. Nelson, W. T. Illingsworth. *A practical guide to neural nets*. Addison-Wesley Publishing Company Inc. 1991. ISBN: 0-201-56309-6
[65]	[NEUR02]	Neurodimension Inc. Mutation. Available: http://www.nd.com/products/genetic/mutation.htm Retrieved: 2010. Last Updated: 2002.
[66]	[NURM07]	K. Nurmi, J. Kyngas. A Framework for School Timetabling Problem. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference*. Paris, France. 2007

[67]	[OBIT98]	M. Obitko. Introduction to Genetic Algorithms. Available: http://www.obitko.com/tutorials/genetic-algorithms/index.php Retrieved: 2010. Last Updated: 1998
[68]	[OXFO11]	Concise Oxford English Dictionary. Oxford University Press. Available: http://oxforddictionaries.com/definition/methodology?view=get Retrieved: 2011. Last updated: 2011.
[69]	[PAIS09]	T. C. Pais, P. Amaral. Weight Aggregation in a Multi-objective Approach for Examination Timetabling Problems. 2009.
[70]	[PETT93]	E. Pettit, K. M. Swigger. An Analysis of genetic-based pattern tracking. In Proceedings of National Conference of Artificial Intelligence. AAAI 83. Pages 327-332. 1983.
[71]	[RAHO06]	M. Rahoual, R. Saad. Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search. In *The Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT).* 2006.
[72]	[RAMS93]	C. L. Ramsey, J. J. Grefenstette. Case-based Initialization of Genetic Algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms.* San Mateo, California. Morgan Kaufmann. Pages 84-91. 1993.
[73]	[RAND00]	M. Randall. A General Meta-Heuristic Based Solver for Combinatorial Optimization Problems. In *Computational Optimization and Applications.* Vol 20 No 2. Pages 185-210. 2000.
[74]	[RAGH08]	R. Raghavjee, N. Pillay. An Application of Genetic Algorithms to the School Timetabling Problem. In *Proceedings of SAICSIT 2008*. Pages 193-199. ACM Press. 2008.
[75]	[RAGH10]	R. Raghavjee, N. Pillay. Using Genetic Algorithms to Solve the South African School Timetabling Problem. In *Proceedings of Conference for Nature and Biologically Inspired Computing 2010 (NaBIC).* Pages 286-292. 2010.
[76]	[RAZA11]	N. M. Razali, J. Geraghty. Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. Proceedings of the World Congress on Engineering 2011. Vol. 2. July 2011.
[77]	[ROJA96]	R. Rojas. Neural Networks: A Systematic Introduction. Springer. Berlin, Germany. 1996.
[78]	[ROSS06]	F. Rossi, P. van Beek, T. Walsh. Handbook of Constraint

		Programming. Elsevier Science. ISBN: 9780444527264. 2006.
[79]	[SAGI06]	M. Sagir. Multi-objective Course Scheduling with Mathematical Programming and Analytic Network Process. 2006.
[80]	[SANT06]	B. Santosa. *Tutorial Particle Swarm Optimization*. 2006
[81]	[SARM07]	S. Siamak. An investigation on Genetic Algorithm Parameters. Report. Penang: Universiti Sains Malaysia. 2007.
[82]	[SAST05]	K. Sastry, D. Goldberg, G. Kendall. Genetic Algorithms. In *Search Methodologies*. Pages 97 – 125. Springer. 2005.
[83]	[SCHA96]	A. Schaerf. Tabu Search Techniques for Large High-School Timetabling Problems. In *IEEE Transactions on Systems, Management and Cybernetics: Part A.* 1996.
[84]	[SCHA99]	A. Schaerf. A Survey of Automated Timetabling. In *Artificial Intelligence Review 13*. Pages 87-127. Kluwer. 1999.
[85]	[SCHA01]	A. Schaerf, L. Di Gaspero. Local Search Techniques for Educational Timetabling Problems. In *Proceedings of the 6th International Symposium on Operations Research in Slovenia*. Pages 13-23. 2001
[86]	[SIGL03]	B. Sigl, M. Golub, V. Mornar. Solving Timetabling Scheduling Problem using Genetic Algorithms. In *Information Technology Interfaces, 2003*. ITI 2003. Proceedings of the 25th International Conference in IEEE. 2003.
[87]	[SMIT03]	K. A. Smith, D. Abramson, D. Duke. Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparitive Results. In *Computers and Industrial Engineering*. Vol. 44. Pages 283-305. 2003.
[88]	[SPEA91]	W. M. Spears, V. Anand. A study of Crossover Operators in Genetic Programming. *Methodologies for Intelligent Systems*. Pages 409-418. 1991.
[89]	[SPEA93]	W. M. Spears. Crossover or Mutation? In *Foundation of Genetic Algorithms,* Volume 2, Pages 221-237. Kaufmann. 1993.
[90]	[SPED95]	W. M. Spears, K. De Jong. On the Virtues of Parameterized Uniform Crossover. In *Naval Research Lab*. Washington D.C. 1995.
[91]	[SRIN94]	M. Srinivas, L. M. Patnaik. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. In IEEE Transactions on Systems, Man and Cybernetics, Vol. 24 No. 4. April 1994.
[92]	[SUMA08]	Evolutionary Intelligence: An introduction to theory and applications

		with Matlab: S. Sumathi and T. Hamsapriya and P. Surekha. 2008
[93]	[SYSW89]	G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schafer (Ed.) Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, California. Morgan Kaufmann Publishers Inc.
[94]	[TERA99]	H. Terashima-Marin, P. Ross. Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99).* 1999.
[95]	[TONG99]	S. Tongchim. Coarse-Grained Parallel Genetic Algorithm for Solving the Timetable Problem. In *Proceedings of 3rd Annual National Symposium on Computational Science and Engineering.* 1999.
[96]	[VALO03]	C. Valouxis, E. Housos. Constraint programming approach for school timetabling. In *Computers and Operations Research*. Vol 30. Pages 1555-1572. 2003.
[97]	[WILK02]	P. Wilke, M. Grobner, N Oster. A Hybrid Genetic Algorithm for School Timetabling. In *AI 2002. Advances in Artificial Intelligence, Lecture Notes in Computer Science*. Volume 2557/2002. Pages 455-464. 2002
[98]	[WILK08]	P. Wilke, J. Ostler. Solving the School Timetabling Problem Using Tabu Search, Simulated Annealing, Genetic and Branch & Bound Algorithms. In *The Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT).* 2008
[99]	[WOLS98]	L. Wolsey. *Integer Programming*. Wiley-Interscience. ISBN: 9780471283669. 1998.
[100]	[XIE09]	H. Xie, M. Zhang. Tuning Selection Pressure in Tournament Selection. In *Technical Report Series*. School of Engineering and Computer Science, Victoria University of Wellington. New Zealand. 2009.
[101]	[ZDAN02]	M. Zdansky, J. Pozivil. Combination Genetic/Tabu Search Algorithm for Hybrid Flowshops Optimization. *Proceedings of ALGORITMY 2002*. Conference on Scientific Computing. Pages 230-236. 2002.
[102]	[ZITZ99]	E. Zitzler. Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Volume 63. Shaker. 1999.
[103]	[ZUTE07]	J. Zuters. Neural Networks to Enrich Fitness Function in a GA-based School Timetabling Model. In *WSEAS Transactions on Information Science and Applications.* Issue 2 Vol. 4. Pages 346-353. February 2007.

Appendix A – School Timetabling Problem data

A.1 Abramson data sets

```
NUMBER OF TEACHERS =         4
NUMBER OF SUBJECTS =        20
NUMBER OF CLASSES =         4
NUMBER OF ROOM AVAILABLE =         4
NUMBER OF REQUIREMENTS =       120
```

Figure A.1: HDTT4 Details

```
2  2  1  2
1  1  1  2
1  1  1  6
2  2  3  2
2  5  1  2
0  4  3  2
1  2  1  0
2  2  1  2
2  1  1  2
0  0  5  1
2  1  4  1
6  1  2  1
3  1  2  1
1  4  1  4
3  3  2  1
2  0  1  1
```

Figure A.2: HDTT4 Requirements

```
NUMBER OF TEACHERS =         5
NUMBER OF SUBJECTS =        20
NUMBER OF CLASSES =         5
NUMBER OF ROOM AVAILABLE =         5
NUMBER OF REQUIREMENTS =       150
```

Figure A.3: HDTT5 Details

1	0	0	1	2
2	0	1	3	1
0	0	1	2	1
1	2	3	1	0
0	2	1	4	1
4	0	3	1	3
0	1	0	1	3
0	1	2	1	1
1	0	3	0	0
2	1	0	2	0
2	2	0	0	2
3	2	1	2	1
1	0	3	1	1
3	0	0	0	3
1	0	0	1	1
1	2	0	0	1
0	2	3	1	0
2	0	2	0	4
1	3	0	2	1
1	1	0	2	1
1	1	1	1	1
0	3	0	0	0
1	2	3	1	0
1	2	1	1	1
1	3	2	2	1

Figure A.4: HDTT5 Requirements

NUMBER OF TEACHERS = 6
NUMBER OF SUBJECTS = 20
NUMBER OF CLASSES = 6
NUMBER OF ROOM AVAILABLE = 6
NUMBER OF REQUIREMENTS = 180

Figure A.5: HDTT6 Details

1	0	1	1	0	3
0	2	0	0	0	0
0	1	0	1	0	0
1	0	2	1	0	1
2	1	1	2	2	1
0	1	1	1	1	2
2	1	1	0	2	0
0	0	1	0	2	3
0	0	0	0	2	1
1	0	0	0	0	0
0	2	1	2	2	1
3	0	1	1	1	0
2	1	1	1	0	0
1	0	3	0	1	0
0	0	1	1	2	1
1	1	1	1	1	2
1	1	1	0	0	0
1	1	2	0	1	0
0	2	0	0	2	1
0	1	1	2	0	2
0	2	0	0	3	0

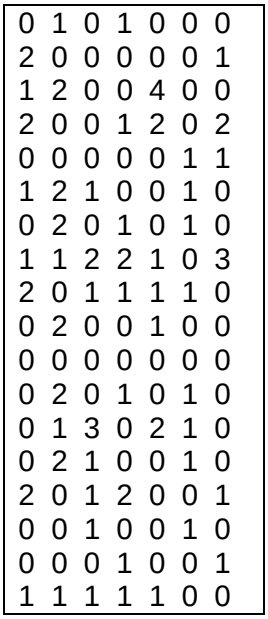
1	1	2	1	0	0
1	0	1	0	1	2
1	0	0	1	1	1
0	1	0	0	0	3
0	1	0	3	1	0
3	0	1	4	0	2
2	0	1	0	0	1
2	1	0	0	0	0
1	1	0	0	1	1
0	1	1	0	2	0
0	3	2	1	0	0
0	1	0	2	1	1
1	2	2	3	0	0
1	0	0	0	1	0
1	1	1	1	0	1

Figure A.6: HDTT6 Requirements

NUMBER OF TEACHERS = 7
NUMBER OF SUBJECTS = 20
NUMBER OF CLASSES = 7
NUMBER OF ROOM AVAILABLE = 7
NUMBER OF REQUIREMENTS = 210

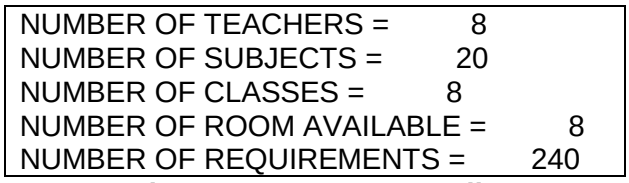
Figure A.7: HDTT7 Details

1	0	0	0	2	1	2
0	1	1	0	0	1	1
0	1	0	1	1	0	0
0	0	0	1	1	1	0
1	0	3	0	0	1	1
0	1	0	1	0	0	1
0	0	1	1	1	1	1
2	0	0	1	2	1	0
0	1	2	1	0	0	0
0	0	0	0	2	1	0
1	0	0	0	1	0	0
1	0	0	0	0	0	1
4	1	1	3	1	0	0
1	0	0	1	0	1	0
0	0	1	1	0	2	0
1	1	0	0	2	1	0
1	0	1	0	1	0	0
0	0	2	1	0	1	0
0	1	1	0	0	1	2
3	0	0	0	0	0	1
0	1	2	0	2	0	0
0	0	0	0	0	1	4
0	1	0	1	0	0	0
0	1	1	2	1	1	2
0	0	1	1	0	0	1
1	0	2	0	0	1	2
1	0	0	0	0	0	0
0	2	0	1	1	0	1
0	2	0	0	0	1	0
0	0	0	1	0	1	1
0	0	0	1	0	4	0



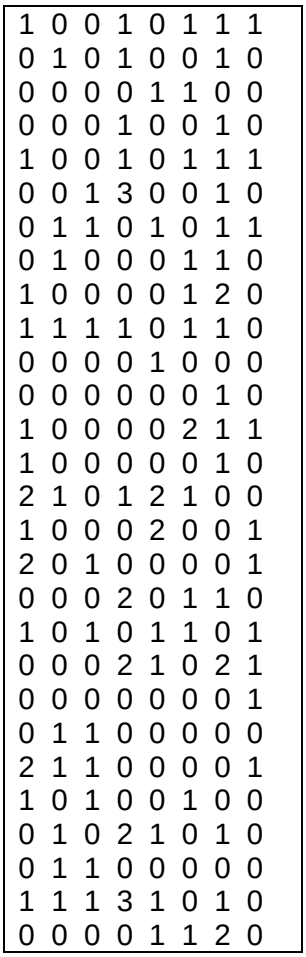
0	1	0	1	0	0	0
2	0	0	0	0	0	1
1	2	0	0	4	0	0
2	0	0	1	2	0	2
0	0	0	0	0	1	1
1	2	1	0	0	1	0
0	2	0	1	0	1	0
1	1	2	2	1	0	3
2	0	1	1	1	1	0
0	2	0	0	1	0	0
0	0	0	0	0	0	0
0	2	0	1	0	1	0
0	1	3	0	2	1	0
0	2	1	0	0	1	0
2	0	1	2	0	0	1
0	0	1	0	0	1	0
0	0	0	1	0	0	1
1	1	1	1	1	0	0

Figure A.8: HDTT 7 Requirements



NUMBER OF TEACHERS = 8
NUMBER OF SUBJECTS = 20
NUMBER OF CLASSES = 8
NUMBER OF ROOM AVAILABLE = 8
NUMBER OF REQUIREMENTS = 240

Figure A.9: HDTT8 Details



1	0	0	1	0	1	1	1
0	1	0	1	0	0	1	0
0	0	0	0	1	1	0	0
0	0	0	1	0	0	1	0
1	0	0	1	0	1	1	1
0	0	1	3	0	0	1	0
0	1	1	0	1	0	1	1
0	1	0	0	0	1	1	0
1	0	0	0	0	1	2	0
1	1	1	1	0	1	1	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0
1	0	0	0	0	2	1	1
1	0	0	0	0	0	1	0
2	1	0	1	2	1	0	0
1	0	0	0	2	0	0	1
2	0	1	0	0	0	0	1
0	0	0	2	0	1	1	0
1	0	1	0	1	1	0	1
0	0	0	2	1	0	2	1
0	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0
2	1	1	0	0	0	0	1
1	0	1	0	0	1	0	0
0	1	0	2	1	0	1	0
0	1	1	0	0	0	0	0
1	1	1	3	1	0	1	0
0	0	0	0	1	1	2	0

1	0	0	1	0	1	0	0
0	0	0	0	0	2	0	0
0	0	0	1	1	0	0	1
0	2	0	1	0	0	0	0
0	0	0	0	0	0	0	2
0	0	1	0	0	0	0	1
0	0	0	0	1	1	0	0
0	1	0	0	1	1	1	1
1	0	0	1	2	0	0	0
1	0	2	0	0	0	1	1
0	0	0	0	2	0	0	0
1	0	1	1	0	2	0	3
0	2	0	0	0	0	0	1
1	1	0	0	0	0	1	0
1	0	3	0	0	0	0	0
0	0	1	1	1	0	2	0
1	0	1	0	1	1	1	0
1	1	0	0	0	1	0	0
1	0	2	1	0	0	0	0
0	0	1	0	0	1	0	1
0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	2
0	2	0	0	1	0	1	0
0	0	2	0	1	0	0	1
0	1	0	1	0	0	1	2
1	1	0	2	1	1	0	1
1	0	0	0	2	0	0	0
0	0	1	0	0	0	1	0
0	2	1	0	0	1	1	0
0	1	2	0	1	2	0	1
0	2	0	0	0	2	0	0
1	0	0	0	1	0	0	1
0	1	0	0	1	0	0	0
0	0	1	2	0	1	0	0
0	0	1	0	0	0	0	0
2	1	0	0	1	0	0	0

Figure A.10: HDTT8 Requirements

A.2 Valouxis data set

Teachers:	15
Classes:	6
Sessions:	2

Figure A.11: Valouxis Details

Teacher	Class Section						Total Hours	Days					Desired Shift	
ID	C1	C2	C3	C4	C5	C6		Mo	Tu	We	Th	Fr	Early	Late
T1	3	3	3	3	2	2	16	1	1	1	1	1	0	1
T2	9	0	8	0	0	0	17	1	1	1	1	1	1	0
T3	0	9	0	8	0	0	17	1	1	1	1	1	1	0
T4	2	0	0	0	8	8	18	1	1	1	1	1	1	0
T5	0	2	2	2	4	4	14	1	1	1	1	0	0	1
T6	2	2	2	2	2	2	12	1	1	1	0	0	0	1
T7	2	2	2	2	2	2	12	1	1	1	1	1	0	1
T8	3	3	3	3	2	2	16	1	1	1	1	1	0	1
T9	1	1	1	1	1	1	6	0	0	0	1	1	0	1
T10	0	0	1	1	1	1	4	0	0	0	0	1	1	0
T11	4	4	4	2	2	2	18	1	1	1	1	1	0	1
T12	4	4	4	4	0	0	16	1	1	1	1	1	1	0
T13	0	0	0	2	6	6	14	1	1	1	1	0	1	0
T14	3	3	3	3	2	2	16	1	1	1	1	1	0	1
T15	2	2	1	1	0	0	6	1	0	0	0	1	0	1
Total	35	35	34	34	32	32	202							

Figure A.12: Valouxis requirements

A.3 Beligiannis data sets

Number of Classes	11
Number of Teachers	34
Number of Days	5
Hours per Day	7
MaxNo of Splits	18

T	Days					1		2		3		4		5		6		7		8		9		10		11		Classes													
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	2	1	2	1	0	0	11	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	7	2	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	0	0	0	0	0	0	0	0	8	3	0	0	0	0	3	1	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	8	3	0	0	3	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	2	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	3	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	0	0	9	3	9	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	2	1	0	0	2	1	0	0	2	1	10	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0	4	1	0	0	4	1	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	0	0	4	1	0	0	4	1	0	0	4	1	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	1	1	4	1	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	4	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	1	3	2	3	2	2	1	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	3	2	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	1	0	0	0	0	0	0	0	0	0	0	3	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	1	1	1	1	1	0	0	3	1	3	1	3	1	0	0	0	0	0	0	3	1	0	0	0	0	3	1	0	0	2	0	0	0	0	0	0	0	0	8	0	0
20	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	3	1	3	1	0	0	3	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	1	1	1	1	1	3	1	0	0	3	1	0	0	2	1	0	0	4	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	7	0	0	9	0	0	0
22	1	0	0	1	1	0	0	3	1	0	0	3	1	0	0	2	1	0	0	0	0	2	1	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	1	1	0	1	3	1	3	1	0	0	0	0	3	1	0	0	0	0	3	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	8	0	0
24	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	1	1	1	1	1	0	0	0	0	3	1	3	1	0	0	0	0	3	1	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	1	1	1	1	1	3	1	3	1	0	0	0	0	3	1	3	1	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	2	1	4	2	2	1	0	0	0	0	0	0	0	0	0	8	0	0	10	0
29	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	2	1	1	2	0	3	4	5	6	0	7	0	0	0	0	11
30	0	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	1	1	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	1	1	1	1	1	0	0	2	1	0	0	0	0	2	1	2	1	2	1	4	2	2	1	4	2	2	1	0	2	0	0	0	5	6	0	7	8	9	0	10	11
34	1	1	1	0	0	2	1	0	0	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	4	0	0	0	0	0	0	0	0	0

Figure A.13: High_school_01 details and requirements

Number of Classes	11
Number of Teachers	35
Number of Days	5
Hours per Day	7
Maxno of Splits	24

T	Days					1		2		3		4		5		6		7		8		9		10		11		Classes																						
1	1	1	1	1	1	0	0	0	0	0	0	9	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	2	1	0	0	0	0	2	1	0	0	0	0	0	0	8	3	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	0	0	6	2	0	0	9	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	0	0	0	0	9	3	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	1	1	1	0	0	0	0	0	0	0	0	8	3	0	0	0	0	0	0	3	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	9	3	5	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	8	3	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	3	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	3	0	0	0	0	11	4	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0	4	1	4	1	0	0	0	0	0	0	0	0	0	0	4	1	4	1	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	4	1	0	0	0	0	4	1	4	1	0	0	0	0	4	1	0	0	0	0	0	0	0	0	4	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	4	1	4	1	4	1	4	1	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	1	1	1	1	1	0	0	0	0	0	0	0	0	2	1	3	2	3	2	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	2	1	2	1	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	1	1	1	1	4	2	4	2	4	2	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	1	0	1	0	1	0	0	0	0	0	0	0	0	3	2	0	0	0	0	3	1	1	1	1	1	1	1	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	1	1	1	1	1	0	0	3	1	0	0	0	0	0	0	3	1	0	0	3	1	0	0	3	1	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	1	1	1	1	0	0	0	0	3	1	3	1	0	0	0	0	3	1	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	1	1	1	1	1	3	1	0	0	3	1	0	0	2	1	0	0	2	1	0	0	2	1	0	0	2	1	0	0	0	0	0	0	1	3	5	7	9	11	0	0	0	0	0	0	0	0	0	0	0
23	1	1	1	1	1	0	0	3	1	0	0	3	1	0	0	2	1	0	0	2	1	0	0	2	1	2	1	0	0	0	0	0	0	2	4	6	8	10	11	0	0	0	0	0	0	0	0	0	0	0
24	1	1	1	1	1	3	1	3	1	3	1	3	1	0	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	1	1	1	1	1	0	0	0	0	0	0	0	0	3	1	3	1	3	1	3	1	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	1	1	1	1	1	2	1	0	0	2	1	2	1	2	1	2	1	2	1	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3	4	5	6	7	9	10	11	0	0
27	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	4	5	6	7	0	0	0	2	8
28	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	10	11	0	0
29	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	1	0	1	1	1	1	1	1	1	1	1	1	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	1	0	0	2	1	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	8
34	1	1	1	1	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	1	1	1	1	0	3	1	3	1	0	0	0	0	3	1	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.14: High_School_02 details and requirements

Number of Classes	6
Number of Teachers	19
Number of Days	5
Hours per Day	7
MaxNo of Splits	0

	Days					A1		A2		B1		B2		G1		G2	
1_T	1	1	1	1	1	0	0	0	0	0	0	0	0	5	1	0	0
2_T	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1
3_T	1	1	1	1	1	0	0	0	0	6	3	2	1	0	0	0	0
4_T	1	1	1	1	1	9	2	9	2	0	0	0	0	0	0	0	0
5_T	1	1	1	1	1	0	0	0	0	4	2	4	2	4	2	6	2
6_T	1	1	1	1	1	2	1	2	1	0	0	4	2	4	2	7	3
7_T	1	1	1	1	1	0	0	0	0	4	1	4	1	4	1	4	1
8_T	1	1	1	0	1	4	1	4	1	0	0	0	0	0	0	0	0
9_T	1	1	1	1	1	2	1	2	1	5	2	5	2	0	0	1	1
10_T	1	1	1	1	1	2	1	2	1	0	0	0	0	6	3	5	2
11_T	0	1	1	0	1	3	1	3	1	0	0	3	1	0	0	0	0
12_T	0	1	0	1	0	0	0	0	0	3	1	0	0	3	1	3	1
13_T	1	1	1	1	1	3	1	3	1	2	1	2	1	2	1	2	1
14_T	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
15_T	1	1	1	1	1	3	1	3	1	3	1	3	1	2	1	2	1
16_T	0	0	1	0	1	1	1	1	1	2	1	2	1	0	0	0	0
17_T	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
18_T	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0
19_T	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure A.15: High_School_03 details and requirements

Number of Classes	7
Number of Teachers	19
Number of Days	5
Hours per Day	7
Maxno of Splits	12

	Days					1		2		3		4		5		6		7		**Classes**									
1_T	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
2_T	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
3_T	1	1	1	1	1	2	1	2	1	2	1	3	2	2	1	5	2	2	1	0	0	0	0	0	0	0	0	0	0	0
4_T	1	1	1	1	1	4	1	0	0	6	2	4	2	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5_T	1	1	1	1	1	7	3	6	2	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6_T	1	1	1	1	1	0	0	0	0	0	0	4	1	4	1	6	3	5	2	0	0	0	0	0	0	0	0	0	0	0
7_T	1	1	1	1	1	0	0	5	2	0	0	0	0	2	1	4	1	8	3	0	0	0	0	0	0	0	0	0	0	0
8_T	1	1	1	1	1	0	0	0	0	0	0	4	1	4	1	4	1	4	1	0	0	0	0	0	0	0	0	0	0	0
9_T	1	1	1	1	1	4	1	4	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10_T	1	1	1	1	1	2	1	2	1	2	1	1	1	0	0	5	3	5	3	0	0	0	0	0	0	0	0	0	0	0
11_T	1	1	1	1	1	2	1	2	1	2	1	4	2	5	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12_T	1	1	1	1	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	0	0	0	0	0	0	0	0	0	0	0
13_T	1	1	1	1	1	6	2	0	0	6	2	5	2	0	0	0	0	0	0	1	3	4	0	0	0	0	0	0	0	0
14_T	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	4	2	0	0	0	0	0	6	0	0	0	0	0	0	0
15_T	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
16_T	1	1	1	1	1	0	0	6	2	6	2	1	1	5	2	0	0	4	2	2	3	5	7	0	0	0	0	0	0	4
17_T	0	1	0	1	0	1	1	1	1	1	1	2	1	2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
18_T	1	0	1	0	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	1	2	3	4	5	0	
19_T	1	0	1	0	1	2	1	2	1	2	1	2	1	2	1	1	1	1	1	0	0	0	0	1	2	3	4	5	0	

Figure A.16: High_School_04 details and requirements

Number of Classes	6
Number of Teachers	18
Number of Days	5
Hours per Day	7
MaxNo of Splits	0

	Days					1		2		3		4		5		6	
1_T	1	1	1	0	1	4	1	0	0	0	0	0	0	0	0	0	0
2_T	1	1	1	1	1	4	2	2	1	4	2	4	2	2	1	2	1
3_T	1	1	1	1	1	0	0	0	0	4	1	4	1	4	2	4	2
4_T	1	1	1	1	1	5	2	5	2	0	0	0	0	4	1	4	1
5_T	1	1	1	1	1	0	0	4	1	4	2	4	2	3	1	3	1
6_T	1	1	1	1	1	0	0	0	0	4	1	4	1	0	0	0	0
7_T	1	1	1	1	1	4	1	4	1	0	0	0	0	4	1	4	1
8_T	1	1	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1
9_T	1	1	1	1	1	0	0	0	0	5	3	3	2	3	2	3	2
10_T	1	0	1	1	1	2	1	2	1	0	0	2	1	0	0	0	0
11_T	0	1	1	1	0	3	1	3	1	0	0	0	0	0	0	0	0
12_T	0	0	1	1	1	0	0	0	0	3	1	3	1	3	1	3	1
13_T	1	1	1	1	1	3	1	5	2	2	1	2	1	2	1	2	1
14_T	1	1	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1
15_T	0	1	0	1	1	3	2	3	2	2	1	2	1	0	0	0	0
16_T	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
17_T	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
18_T	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure A.17: High_School_05 details and requirements

Number of Classes	13
Number of Teachers	35
Number of Days	5
Hours per Day	7
Maxno of Splits	20

T	Days					1		2		3		4		5		6		7		8		9		10		11		12		13		Classes																			
1	1	1	1	1	1	0	0	9	3	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	2	10	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	4	1	9	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	2	1	10	3	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	9	3	0	0	0	0	0	0	9	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	3	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	1	1	1	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	8	2	0	0	0	0	8	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	2	1	2	1	0	0	2	1	2	1	4	1	0	0	0	0	0	0	3	1	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	4	2	4	2	4	2	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	3	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0	0	0	2	1	2	1	0	0	0	0	0	0	0	0	3	2	3	2	3	2	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	2	1	2	1	3	2	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	1	1	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	1	1	1	1	1	4	1	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	10	11	12	13	0	0	0	0	0	0
16	1	1	1	1	1	0	0	0	0	4	1	4	1	0	0	0	0	0	0	4	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	4	1	4	1	0	0	0	0	4	1	4	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	1	1	1	1	3	1	3	1	0	0	3	1	0	0	0	0	0	0	2	1	0	0	0	0	0	0	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	0	8	0	12
19	1	1	1	1	1	3	1	0	0	3	1	0	0	3	1	2	1	0	0	0	0	0	0	2	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	6	0	10	0
20	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	2	1	0	0	2	1	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	9	11	13
21	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	3	1	3	1	3	1	3	1	0	0	3	1	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	1	1	1	1	1	0	0	3	1	3	1	3	1	3	1	0	0	0	0	0	0	0	0	3	1	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	1	1	1	1	1	0	0	0	0	0	0	3	1	3	1	0	0	0	0	3	1	3	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	1	1	1	1	1	3	1	3	1	3	1	0	0	0	0	3	1	2	1	0	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	1	1	1	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	1	1	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	1	1	1	1	1	2	1	2	1	0	0	0	0	0	0	2	1	2	1	2	1	2	1	0	0	2	1	0	0	2	1	0	1	2	0	0	0	6	7	8	9	0	11	0	13	0	0	0	0	0	0
33	0	0	1	1	1	0	0	0	0	2	1	2	1	2	1	0	0	0	0	0	0	0	0	2	1	0	0	2	1	0	0	0	0	0	3	4	5	0	0	0	0	10	0	12	0	0	0	0	0	0	0
34	1	1	1	1	1	2	1	2	1	2	1	2	1	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	1	2	3	4	0	6	7	8	9	0	0	0	0	0	0	0	0	0	0
35	0	0	0	1	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.18: High_School_07 details and requirements

A.4 Lewitt data set

1	9	9	9	9	9
2	9	9	9	9	9
3	9	9	9	9	9
4	9	9	9	9	9
5	9	9	9	9	9
6	11	10	10	10	9
7	11	10	10	10	9
8	11	11	11	11	9
9	11	11	11	11	9
10	11	11	11	11	9
11	11	11	11	11	9
12	11	11	11	11	9
13	11	11	11	11	9
14	11	11	11	11	9
15	11	11	11	11	9
16	11	11	11	11	9

Figure A.19: Lewitt Periods for each class labeled 1 to 16

School Details
Number of Classes 16
Number of Teachers 19
Number of Subjects 14
Number of Days 5
Periods Per day 11

Class	Subject	Teacher 1	Teacher 2	Times SReq	Clash	Double	Triple	Session	
1A	Fnd	KN	0	45	0	0	0	4	1
1B	Fnd	VP	0	45	0	0	0	4	1
1C	Fnd	AM	0	45	0	0	0	4	1
2A	Fnd	NDG	0	45	0	0	0	4	1
2B	Fnd	SN	0	45	0	0	0	4	1
3A	Afk	NH	0	3	0	1	0	7	0
3A	Fnd	HPB	0	45	0	0	0	7	0
3A	Lfe	AM	0	2	0	1	0	7	0
3B	Afk	NH	0	3	0	1	0	7	0
3B	Fnd	RM	0	37	0	0	0	7	0
3B	Lfe	KN	0	1	0	0	0	7	0
3B	Lfe	NH	0	2	0	1	0	7	0
3B	Lfe	VP	0	2	0	0	0	7	0
3B	Lfe	NDG	0	2	0	0	0	7	0
3B	Zul	HPB	0	3	0	1	0	7	0
4A	ACc	SB	0	3	0	1	0	7	0
4A	Afk	JM	0	5	0	0	0	7	0
4A	EMS	JM	0	3	0	1	0	7	0
4A	Eng	US	0	11	0	1	0	7	0
4A	LOr	KND	0	3	0	1	0	7	0
4A	LRE	NDG	0	1	0	0	0	7	0
4A	Mth	US	0	10	0	1	0	4	1
4A	NSc	US	0	2	0	1	0	7	0
4A	NSc	US	0	2	0	0	0	7	0
4A	PEd	AM	SMK	2	1	1	0	7	0
4A	SSc	JM	0	2	0	1	0	7	0
4A	SSc	JM	0	2	0	0	0	7	0
4A	Tch	KS	0	1	0	0	0	7	0
4A	Tch	KS	VBS	2	1	1	0	7	0
4A	Zul	NFM	0	4	0	0	0	7	0
4B	ACc	SB	0	3	0	1	0	7	0
4B	Afk	JM	0	5	0	0	0	7	0
4B	EMS	JM	0	3	0	1	0	7	0
4B	Eng	US	0	11	0	1	0	7	0
4B	LOr	KND	0	3	0	1	0	7	0
4B	LRE	US	0	1	0	0	0	7	0
4B	Mth	US	0	10	0	1	0	4	1

4B	NSc	US	0	2	0	1	0	7	0
4B	NSc	US	0	2	0	0	0	7	0
4B	PEd	KN	SMK	2	1	1	0	7	0
4B	SSc	JM	0	2	0	1	0	7	0
4B	SSc	JM	0	2	0	0	0	7	0
4B	Tch	KS	0	1	0	0	0	7	0
4B	Tch	KS	VBS	2	1	1	0	7	0
4B	Zul	NFM	0	4	0	0	0	7	0
5A	ACc	NFM	0	3	0	1	0	7	0
5A	Afk	JM	0	5	0	0	0	7	0
5A	EMS	YM	0	3	0	1	0	7	0
5A	Eng	KND	0	11	0	1	0	7	0
5A	LOr	JM	0	3	0	1	0	7	0
5A	LRE	AD	0	1	0	0	0	7	0
5A	Mth	AD	0	10	0	1	0	4	1
5A	NSc	KND	0	2	0	1	0	7	0
5A	NSc	KND	0	2	0	0	0	7	0
5A	PEd	NFM	KS	2	1	1	0	7	0
5A	SSc	AD	0	2	0	1	0	7	0
5A	SSc	AD	0	2	0	0	0	7	0
5A	Tch	KS	0	1	0	0	0	7	0
5A	Tch	KS	DN	2	1	1	0	7	0
5A	Zul	NFM	0	4	0	0	0	7	0
5B	ACc	NFM	0	3	0	1	0	7	0
5B	Afk	JM	0	5	0	0	0	7	0
5B	EMS	YM	0	3	0	1	0	7	0
5B	Eng	KND	0	11	0	1	0	7	0
5B	LOr	JM	0	3	0	1	0	7	0
5B	LRE	KND	0	1	0	0	0	7	0
5B	Mth	AD	0	10	0	1	0	4	1
5B	NSc	KND	0	2	0	1	0	7	0
5B	NSc	KND	0	2	0	0	0	7	0
5B	PEd	VP	AD	2	1	1	0	7	0
5B	SSc	AD	0	2	0	1	0	7	0
5B	SSc	AD	0	2	0	0	0	7	0
5B	Tch	KS	0	1	0	0	0	7	0
5B	Tch	KS	DN	2	1	1	0	7	0
5B	Zul	NFM	0	4	0	0	0	7	0
5C	ACc	NFM	0	3	0	1	0	7	0
5C	Afk	JM	0	5	0	0	0	7	0
5C	EMS	YM	0	3	0	1	0	7	0
5C	Eng	KND	0	11	0	1	0	7	0
5C	LOr	JM	0	3	0	1	0	7	0
5C	LRE	NDG	0	1	0	0	0	7	0
5C	Mth	AD	0	10	0	1	0	4	1
5C	NSc	KND	0	2	0	1	0	7	0
5C	NSc	KND	0	2	0	0	0	7	0
5C	PEd	AD	KS	2	1	1	0	7	0

5C	SSc	AD	0	2	0	1	0	7	0
5C	SSc	AD	0	2	0	0	0	7	0
5C	Tch	KS	0	1	0	0	0	7	0
5C	Tch	KS	DN	2	1	1	0	7	0
5C	Zul	NFM	0	4	0	0	0	7	0
6A	ACc	SB	0	3	0	1	0	7	0
6A	Afk	SMK	0	5	0	0	0	7	0
6A	EMS	DN	0	3	0	1	0	7	0
6A	Eng	SB	0	11	0	1	0	7	0
6A	LOr	KS	0	3	0	1	0	7	0
6A	LRE	SN	0	1	0	0	0	7	0
6A	Mth	VBS	0	10	0	1	0	4	1
6A	NSc	SB	0	2	0	1	0	7	0
6A	NSc	SB	0	2	0	0	0	7	0
6A	PEd	SN	VBS	2	1	1	0	7	0
6A	SSc	SB	0	2	0	1	0	7	0
6A	SSc	SB	0	2	0	0	0	7	0
6A	Tch	KS	0	1	0	0	0	7	0
6A	Tch	KS	NFM	2	1	1	0	7	0
6A	Zul	NFM	0	4	0	0	0	7	0
6B	ACc	SB	0	3	0	1	0	7	0
6B	Afk	SMK	0	5	0	0	0	7	0
6B	EMS	DN	0	3	0	1	0	7	0
6B	Eng	SB	0	11	0	1	0	7	0
6B	LOr	KS	0	3	0	1	0	7	0
6B	LRE	KN	0	1	0	0	0	7	0
6B	Mth	VBS	0	10	0	1	0	4	1
6B	NSc	SB	0	2	0	1	0	7	0
6B	NSc	SB	0	2	0	0	0	7	0
6B	PEd	KS	AD	2	1	1	0	7	0
6B	SSc	SB	0	2	0	1	0	7	0
6B	SSc	SB	0	2	0	0	0	7	0
6B	Tch	KS	0	1	0	0	0	7	0
6B	Tch	KS	DN	2	1	1	0	7	0
6B	Zul	NFM	0	4	0	0	0	7	0
7A	ACc	SMK	0	3	0	1	0	7	0
7A	Afk	SMK	0	5	0	0	0	7	0
7A	EMS	YM	0	3	0	1	0	7	0
7A	Eng	YM	0	11	0	1	0	7	0
7A	LOr	SMK	0	3	0	1	0	7	0
7A	LRE	SN	0	1	0	0	0	7	0
7A	Mth	DN	0	10	0	1	0	4	1
7A	NSc	SMK	0	2	0	1	0	7	0
7A	NSc	SMK	0	2	0	0	0	7	0
7A	PEd	SMK	YM	2	1	1	0	7	0
7A	SSc	DN	0	2	0	1	0	7	0
7A	SSc	DN	0	2	0	0	0	7	0
7A	Tch	KS	0	1	0	0	0	7	0

7A	Tch	KS	VBS	2	1	1	0	7	0
7A	Zul	NFM	0	4	0	0	0	7	0
7B	ACc	SMK	0	3	0	1	0	7	0
7B	Afk	SMK	0	5	0	0	0	7	0
7B	EMS	YM	0	3	0	1	0	7	0
7B	Eng	YM	0	11	0	1	0	7	0
7B	LOr	SMK	0	3	0	1	0	7	0
7B	LRE	YM	0	1	0	0	0	7	0
7B	Mth	DN	0	10	0	1	0	4	1
7B	NSc	SMK	0	2	0	1	0	7	0
7B	NSc	SMK	0	2	0	0	0	7	0
7B	PEd	SMK	YM	2	1	1	0	7	0
7B	SSc	DN	0	2	0	1	0	7	0
7B	SSc	DN	0	2	0	0	0	7	0
7B	Tch	KS	0	1	0	0	0	7	0
7B	Tch	KS	VBS	2	1	1	0	7	0
7B	Zul	NFM	0	4	0	0	0	7	0

Figure A.20: Lewitt details and requirements

Type	Class	Tchr	Subj	FromDay	ToDay	FromPrd	ToPrd	Sched	NumPer
1	-1	US	Mth	1	5	1	4	1	10
1	-1	AD	Mth	1	5	1	6	1	10
1	-1	VBS	Mth	1	5	1	4	1	10
1	-1	DN	Mth	1	5	1	4	1	10
0	1A	-1	Fnd	1	5	1	9	1	45
0	1B	-1	Fnd	1	5	1	9	1	45
0	1C	-1	Fnd	1	5	1	9	1	45
0	2A	-1	Fnd	1	5	1	9	1	45
0	2B	-1	Fnd	1	5	1	9	1	45

Figure A.21: Lewitt Scheduled Requirements

A.5 Woodlands data set

8A
8B
8C
8D
8E
8F
8G
9A
9B
9C
9D
9E
10A
10B

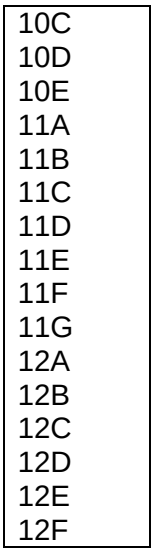


Figure A.22: Class list for Woodlands

Type	Grade	Cl/T	Subj	FromDay	ToDay	FromPrd	ToPrd	Schedule
0	12	C	ENG	1	6	3	3	1
1	-1	AK	-1	1	6	6	6	0
2	10	5	-1	1	6	3	3	1

Figure A.23: Preferences for Woodlands

SCHOOL DETAILS

NUMBER OF CLASSES	30
NUMBER OF TEACHERS	40
NUMBER OF SUBJECTS	44
PERIODS IN DAY	7
DAYS IN WEEK	6

Teacher	Grade	Class	Subject	Meetings
EL	10	1	ML	7
EL	10	2	ML	7
EL	10	B	LO	2
EL	10	C	LO	2
EL	10	D	LO	2
EL	8	A	AC	4
EL	8	B	AC	4
EL	8	C	AC	4
EL	8	D	AC	4
EL	8	G	LO	2
PM	10	B	AFR	7
PM	10	C	AFR	7
PM	10	D	AFR	7
PM	10	E	AFR	7

PM	9	A	AFR	5
PM	9	E	AFR	5
TA	11	E	RP	1
TA	10	5	EGD	6
TA	9	A	TEC	4
TA	9	B	TEC	4
TA	9	C	AC	4
TA	9	C	TEC	4
TA	9	D	AC	4
TA	9	D	TEC	4
TA	9	E	TEC	4
TA	8	F	LO	2
FS	10	A	ENG	7
FS	10	C	ENG	7
FS	10	D	ENG	7
FS	8	B	ENG	5
FS	8	D	ENG	5
FS	8	G	NSC	6
RTS	11	E	BS	6
RTS	9	A	EMS	4
RTS	9	B	EMS	4
RTS	9	B	RP	1
RTS	9	C	EMS	4
RTS	9	D	EMS	4
RTS	9	E	EMS	4
RTS	8	A	LO	2
RTS	8	B	LO	2
RTS	8	C	LO	2
RTS	8	G	EMS	4
SS	10	5	EGD2	6
SS	8	A	TEC	4
SS	8	B	TEC	4
SS	8	C	TEC	4
SS	8	D	LO	2
SS	8	D	TEC	4
SS	8	D	RP	1
SS	8	E	TEC	4
SS	8	F	TEC	4
SS	8	G	TEC	4
MP	10	3	BS	6
MP	10	C	RP	1
MP	10	E	BS	6
MP	8	A	EMS	4

MP	8	B	EMS	4
MP	8	C	EMS	4
MP	8	D	EMS	4
MP	8	E	EMS	4
MP	8	F	EMS	4
AR	12	B	ENG	7
AR	12	B	RP	1
AR	12	D	ENG	7
AR	12	D	RP	1
AR	12	F	ENG	7
AR	10	B	ENG	7
AR	10	E	ENG	7
CG	11	C	ENG	7
CG	11	C	RP	1
CG	11	D	ENG	7
CG	11	D	RP	1
CG	9	A	ENG	5
CG	9	C	ENG	5
CG	9	E	ENG	5
CG	9	E	RP	1
CG	8	F	ENG	5
MVG	11	A	ENG	7
MVG	11	E	ENG	7
MVG	11	G	ENG	7
MVG	11	G	RP	1
MVG	9	B	AFR	5
MVG	9	C	AFR	5
MVG	9	D	AFR	5
PN	12	1	ML2	7
PN	12	6	ML3	7
PN	11	5	ML2	7
PN	10	E	RP	1
PN	9	C	MAT	7
PN	9	D	MAT	7
RT	12	3	LS	6
RT	10	A	LS	6
RT	10	B	LS	6
RT	8	A	NSC	6
RT	8	C	NSC	6
RT	8	E	NSC	6
RLN	12	E	DA	6
RLN	11	2	DA	6

RLN	11	6	DA	6
RLN	10	E	DA	6
RLN	9	A	AC	4
RLN	9	B	AC	4
RLN	9	E	AC	4
TVV	12	A	AFR	7
TVV	12	C	AFR	7
TVV	11	B	AFR	7
TVV	11	E	AFR	7
TVV	10	A	AFR	7
TVV	10	A	RP	1
USM	12	3	PSC	6
USM	12	A	PSC	6
USM	11	4	PSC	6
USM	11	A	PSC	6
USM	10	A	PSC	6
USM	9	D	NSC	6
YH	12	1	MAT	7
YH	12	6	ML	7
YH	12	F	RP	1
YH	11	1	ML	7
YH	10	1	MAT2	7
YH	10	2	MAT	7
TP	9	A	RP	1
TP	8	A	AFR	5
TP	8	B	AFR	5
TP	8	C	AFR	5
TP	8	D	AFR	5
TP	8	F	AFR	5
TP	8	G	ENG	5
TP	8	G	AFR	5
SB	12	2	BS	6
SB	12	3	BS	6
SB	12	4	BS	6
SB	12	C	BS	6
SB	11	6	BS	6
SB	11	F	BS	6
SSM	12	4	TOU	6
SSM	11	3	TOU	6
SSM	11	G	TOU	6
SSM	10	4	TOU	6
SSM	8	C	HSS	4

SSM	8	E	HSS	4
SSM	8	F	HSS	4
TIL	12	B	AFR	7
TIL	12	D	AFR	7
TIL	11	A	AFR	7
TIL	11	A	RP	1
TIL	11	C	AFR	7
TIL	11	F	AFR	7
DPE	12	4	GEO	6
DPE	11	2	GEO	6
DPE	11	7	GEO	6
DPE	10	4	GEO	6
DPE	9	A	HSS	4
DPE	9	B	HSS	4
DPE	9	D	HSS	4
FJ	12	1	ML	7
FJ	12	6	ML2	7
FJ	11	5	ML3	7
FJ	11	E	ML	7
FJ	11	G	ML	7
FJ	10	D	RP	1
GMR	12	B	LO	2
GMR	12	C	LO	2
GMR	12	D	LO	2
GMR	12	E	LO	2
GMR	12	F	LO	2
GMR	11	B	LO	2
GMR	11	C	LO	2
GMR	11	D	LO	2
GMR	11	E	LO	2
GMR	11	F	LO	2
GMR	11	G	LO	2
GMR	9	A	LO	2
GMR	9	B	LO	2
GMR	9	C	LO	2
GMR	9	D	LO	2
GMR	9	E	LO	2
GMR	8	A	RP	1
GMR	8	E	LO	2
GMR	8	E	RP	1
ADK	12	2	IT	6
ADK	12	7	CAT	6
ADK	11	5	ML	7

ADK	11	7	IT	6
ADK	10	B	RP	1
ADK	9	A	MAT	7
ADK	9	C	RP	1
ADK	8	B	RP	1
ADK	8	C	RP	1
ATM	12	3	LS2	6
ATM	11	3	LS	6
ATM	11	B	LS	6
ATM	8	B	NSC	6
ATM	8	D	NSC	6
ATM	8	F	NSC	6
DI	9	B	MAT	7
DI	9	E	MAT	7
DI	8	D	MAT	7
DI	8	E	MAT	7
DI	8	F	MAT	7
DI	8	F	RP	1
LAN	12	3	HOS	6
LAN	12	4	CON	6
LAN	12	7	CON	6
LAN	11	6	CON	6
LAN	11	7	HOS	6
LAN	10	5	CON	6
PG	12	5	GEO	6
PG	11	4	GEO	6
PG	10	B	GEO	6
PG	9	C	HSS	4
PG	8	A	MAT	7
PG	8	C	MAT	7
NS	12	2	ACC	6
NS	12	5	ECO	6
NS	12	C	ECO	6
NS	11	3	ACC	6
NS	11	4	ECO	6
NS	11	6	ECO	6
TCP	9	E	HSS	4
TCP	8	B	MAT	7
TCP	8	E	AC	4
TCP	8	F	AC	4
TCP	8	G	MAT	7
TCP	8	G	AC	4

TCP	8	G	HSS	4
TCP	8	G	RP	1
PH	12	1	MAT2	7
PH	12	6	MAT	7
PH	11	1	MAT	7
PH	11	5	MAT	7
PH	10	1	MAT	7
SMA	12	A	ENG	7
SMA	12	A	RP	1
SMA	12	E	ENG	7
SMA	12	E	RP	1
SMA	11	B	ENG	7
SMA	11	B	RP	1
SMA	11	F	ENG	7
SMA	11	F	RP	1
ZK	9	B	ENG	5
ZK	9	D	ENG	5
ZK	9	D	RP	1
ZK	8	A	ENG	5
ZK	8	C	ENG	5
ZK	8	E	ZUL	5
ZK	8	E	ENG	5
DMO	12	3	EGD	6
DMO	12	5	EGD	6
DMO	11	2	EGD	6
DMO	11	3	EGD	6
DMO	11	7	EGD	6
LMD	12	A	LO	2
LMD	11	4	HIS	6
LMD	11	7	HIS	6
LMD	11	A	LO	2
LMD	10	3	HIS	6
LMD	10	A	LO	2
LMD	10	E	HIS	6
SIT	12	E	AFR	7
SIT	12	F	AFR	7
SIT	11	D	AFR	7
SIT	11	G	AFR	7
SIT	10	E	LO	2
NMG	12	3	ACC	6
NMG	12	C	ACC	6

NMG	11	6	ACC	6
NMG	10	A	ACC	6
NMG	10	B	ACC	6
AK	9	A	NSC	6
AK	9	B	NSC	6
AK	9	C	NSC	6
AK	9	E	NSC	6
SSN	8	A	HSS	4
SSN	8	B	HSS	4
SSN	8	D	HSS	4
HAJ	12	C	ENG	8

Figure A.24: Woodlands details and requirements

1	12	A
1	12	B
1	12	E
2	12	A
2	12	B
3	12	A
3	12	B
3	12	D
3	12	E
3	12	F
4	12	D
4	12	E
4	12	F
5	12	B
5	12	D
6	12	C
6	12	D
6	12	F
7	12	F
1	11	A
1	11	C
2	11	E

2	11	F
3	11	A
3	11	C
3	11	D
4	11	B
4	11	C
4	11	E
5	11	B
5	11	D
5	11	F
6	11	B
6	11	C
6	11	D
6	11	G
7	11	A
7	11	D
7	11	F
7	11	G
1	10	A
1	10	C
1	10	E
2	10	B
2	10	D
3	10	C
3	10	D
4	10	C
4	10	D
5	10	C
5	10	D

Figure A.25: List of Split and Subclasses for Woodlands