Structure Based Partial Solution Search for the Examination Timetabling Problem

UNIVERSITY OF
KWAZULU-NATAL

Christopher Bradley Rajah

Supervisor: Professor Nelishia Pillay

Submitted in fulfilment of the academic requirements of

Doctor of Philosophy

Computer Science

School of Mathematics, Statistics and Computer Science

University of KwaZulu-Natal

Pietermaritzburg

South Africa

January 2021

PREFACE

The research contained in this thesis was completed by the candidate while based in the Discipline of Computer Science, School of Mathematics, Statistics and Computer Science of the College of Agriculture, Engineering and Science, University of KwaZulu-Natal, Pietermaritzburg, South Africa.

The contents of this work have not been submitted in any form to another university and, except where the work of others is acknowledged in the text, the results reported are due to investigations by the candidate.

Signed: Professor Nelishia Pillay

Date:

DECLARATION 1: PLAGIARISM

I, **Christopher Bradley Rajah**, declare that:

(i) the research reported in this dissertation, except where otherwise indicated or acknowledged, is my original work;

(ii) this dissertation has not been submitted in full or in part for any degree or examination to any other university;

(iii) this dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;

(iv) this dissertation 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, their writing has been placed inside quotation marks, and referenced;

(v) where I have used material for which publications followed, I have indicated in detail my role in the work;

(vi) this dissertation is primarily a collection of material, prepared by myself, published as journal articles or presented as a poster and oral presentations at conferences. In some cases, additional material has been included;

(vii) this dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

[redacted]

____

Signed: Christopher Bradley Rajah

Date: 29 January 2021

DECLARATION 2: PUBLICATIONS

The following publications are associated with the research presented in this thesis:

[1] C. Rajah and N. Pillay, "A Study of Cell Depletion in the Developmental Approach for the Uncapacitated Examination Timetabling Problem," 2013. Annual Conference of the Operations Research Society of South Africa (ORSSA 2013), pp. 102-111.

[2] C. Rajah and N. Pillay, "Combining development and evolution: Case study: One dimensional bin-packing," in IJCCI 2015 - Proceedings of the 7th International Joint Conference on Computational Intelligence, 2015, vol. 1, pp. 188–195.

[3] C. Rajah and N. Pillay, "A Structure-Based Partial Solution Search for the Examination Timetabling Problem," in 2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings, 2019, pp. 81–86.

[4] C. Rajah and N. Pillay, "A Revised Structure-Based Partial Solution Search for the Examination Timetabling Problem," in Journal of the Operational Research Society, under review.

ABSTRACT

The aim of this work is to present a new approach, namely, Structure Based Partial Solution Search (SBPSS) to solve the Examination Timetabling Problem. The success of the Developmental Approach in this problem domain suggested that the strategy of searching the spaces of partial timetables whilst constructing them is promising and worth pursuing. This work adopts a similar strategy. Multiple timetables are incrementally constructed at the same time. The quality of the partial timetables is improved upon by searching their partial solution spaces at every iteration during construction. Another key finding from the literature survey revealed that although timetables may exhibit the same behaviour in terms of their objective values, their structures or exam schedules may be different. The challenge with this finding is to decide on which regions to pursue because some regions may not be worth investigating due to the difficulty in searching them. These problematic areas may have solutions that are not amenable to change which makes it difficult to improve them. Another reason is that the neighbourhoods of solutions in these areas may be less connected than others which may restrict the ability of the search to move to a better solution in that neighbourhood. By moving to these problematic areas of the search space the search may stagnate and waste expensive computational resources. One way to overcome this challenge is to use both structure and behaviour in the search and not only behaviour alone to guide the search. A search that is guided by structure is able to find new regions by considering the structural components of the candidate solutions which indicate which part of the search space the same candidates occupy. Another benefit to making use of a structure-based search is that it has no objective value bias because it is not guided by only the objective value. This statement is consistent with the literature survey where it is suggested that in order to achieve good performance the search should not be guided by only the objective value. The proposed method has been tested on three popular benchmark sets for examination timetabling, namely, the Carter benchmark set; the benchmark set from the International Timetabling competition in 2007 and the Yeditepe benchmark set. The SBPSS found the best solutions for two of the Carter problem instances. The SBPSS found the best solutions for four of the competition problem instances. Lastly, the SBPSS improved on the best results for all the Yeditepe problem instances.

ACKNOWLEDGMENTS

A number of special acknowledgements deserve specific mention:

My wife, Mrs Dolores Rajah, who helped me to persevere and not to quit throughout this journey;

My parents, Mr Dhuravasan Rajah and Mrs Dhanaluchmee Rajah, for their continued support and encouragement;

My supervisor, Professor Nelishia Pillay, for her guidance and who helped me reach the end goal;

The University of KwaZulu-Natal for allowing me to pursue my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Purpose of the Study

The purpose of this work is to propose a new approach, namely, Structure Based Partial Solution Search (SBPSS) to solve the Examination Timetabling Problem (ETP). This problem has been well researched over the years and many approaches have been proposed to solve it [1]. A survey of the literature revealed that searching the spaces of partial timetables does result in good quality complete timetables. The same strategy was adopted by the Developmental Approach (DA) [2]. It was successfully applied to the uncapacitated version of the ETP and it obtained results comparable to other best performing nature-inspired methods for this problem domain. Later work by Rajah and Pillay [3] revealed that the performance of the DA may be improved by removing random exams from the partially complete timetable and reassigning them during construction. The approach proposed in this study adopts some of the same strategies. Multiple timetables are incrementally constructed at the same time. The quality of the partial timetables are improved upon by searching their partial solution spaces at every iteration after the assignment of a new examination during construction. The SBPSS has a deconstruction phase where exams are removed from each of the timetables. The exams that are removed are then reassigned to each of the partial timetable solutions in the next construction phase of the SBPSS.

Another key finding from the literature survey revealed that some timetables may exhibit the same behaviour in terms of the objective value although they have different structures. The search may not be able to decide on which regions to pursue if the same regions have similar behaviour. Some regions are difficult to search and are not worth investigating because the likelihood of finding better quality solutions in these areas is minimal. One reason is that these problematic areas may have solutions that are not amenable to change and cannot be easily improved upon. Another reason is that that these areas may contain neighbourhoods that are less connected and not easy to traverse [4]. As a result the ability of the search to move to a better solution in such neighbourhoods is restricted. By moving to these problematic regions of the search space the search may stagnate and waste expensive computational resources. The proposed approach then addresses this challenge by adopting a search that is guided by not only behaviour but also by structure when moving through the solution space towards more promising regions. This reduces any objective value bias in the search process. This approach is consistent with the literature survey where it is suggested that in order to achieve good performance the search should not be guided by the objective value alone [5]. The approach finds new regions by considering the structural components of the candidate solutions which indicate which part of the search space the same candidates occupy. The proposed approach has been tested on three popular benchmark sets for the ETP, namely, the Carter benchmark set; the benchmark set from the International Timetabling competition in 2007 (ITC2007) and the Yeditepe benchmark set. The SBPSS found the best solutions for two of the Carter problem instances. The SBPSS found the best solutions for four of the competition problem instances. Lastly, the SBPSS improved on the best results for all the Yeditepe problem instances.

1.2 Aims

The aim of this work is to present a new approach, namely, Structure Based Partial Solution Search (SBPSS) to solve the Examination Timetabling Problem (ETP).

1.3 Objectives

The objectives of this work are as follows:

- To demonstrate that the use of structure combined with behaviour in terms of the objective value to guide the search is a worthwhile and promising approach to solving the ETP.
- To demonstrate that searching partial solution spaces whilst constructing solutions leads to good quality solutions for the ETP.
- To combine the structure-based search together with partial solution space in a structure-based partial solution space to solve the ETP.

1.4 Outline of dissertation/thesis structure

The rest of this thesis is structured as follows:

Chapter 2 is devoted to defining metaheuristics. The chapter presents some popular trajectory-based and population-based approaches. The remainder of the chapter focuses on the diversification and intensification capabilities of the considered approaches.

Chapter 3 introduces the ETP and the two main versions of the problem, namely, the capacity and uncapacitated versions. The benchmark problem sets that represent both versions that are used for assessment in this study are then presented. This is followed by a review of the best performing approaches for each benchmark problem set.

Chapter 4 outlines the different research methodologies and the methodology adopted in this study. The chosen methodology steps are explained and the technical specifications for this study are also given.

Chapter 5 introduces the new approach proposed to solve the Examination Timetabling Problem. The approach and the parameters used by the approach are explained in detail. This is followed by a discussion on how the approach is implemented to solve the problems from the benchmarks sets for both the capacitated and uncapacitated version of the Examination Timetabling Problem.

Chapter 6 presents the results for all simulations and discusses the reasons for the results obtained.

Chapter 7 concludes this thesis by providing a summary of the work done and any conclusions that are drawn. Also included in the chapter is the future work that should be considered in extending this work.

CHAPTER 2: METAHEURISTICS

2.1. Introduction

NP-hard combinatorial optimization problems such as Timetabling Problems cannot be solved completely in polynomial time. In such cases it may be more desirable to find near-optimal solutions in a reasonable amount of time. Methods used to solve NP-hard problems in this manner are referred to as approximate methods. One popular class of approximate methods are metaheuristics. The focus in this chapter is to provide background information on metaheuristics. Subsection 2.2 defines metaheuristics. Subsection 2.3 explains the two broad categories of metaheuristics, namely, trajectory-based methods and population-based methods together with an overview of some of the popular approaches found in both these categories. Subsection 2.4 provides an overview of the search mechanisms used by metaheuristics. The chapter is concluded in subsection 2.5.

2.2. Definition and Characteristics of Metaheuristics

Glover [6] first introduced the term *metaheuristics* to refer to a category of approximate algorithms used to find near-optimal solutions to complex problems. Many definitions for metaheuristics have been proposed over the years. Osman and Laporte [7] define a metaheuristic as a higher level process which guides lower level heuristics to effectively explore the solution space. Similarly, Lodi et al. [8] describe a metaheuristic as a master process that guides subordinate heuristic processes to find good solutions. In general, metaheuristics refer to those methods that employ higher level strategies to overcome the limitations of problem-specific search heuristics in order to efficiently and effectively explore the search space to find better solutions. The performance of a metaheuristic is dependent on two key processes;

- Diversification [6] – this process describes the ability of the search to find new areas in the search space which may potentially have better solutions in pursuit of the global optimum. The search may also need to return to areas already found to be promising for further investigation.
- Intensification [6] – this process describes the ability of the search to efficiently and effectively investigate a region of interest in order to find better quality solutions.

An effective search needs to strike a fine balance between the level of intensification and diversification that is performed [9]. There is a trade-off between these two competing processes because as one is increased the other is decreased [10]. Too much intensification leads to the search spending too much time investigating areas that are suboptimal and wasting expensive resources. The search is more likely to converge prematurely and become trapped in a local optimum. On the other hand, too much diversification can lead to the search spending insufficient time investigating good performing areas and missing the opportunity to find better solutions in these areas. In the next subsection the different categories of metaheuristics are presented.

2.3. Categories of Metaheuristics

Metaheuristics may be classified in many different ways. One such classification makes use of the number of solutions that are considered at any one time by the approach. Based on this definition, there are two broad categories of metaheuristics, namely, trajectory-based metaheuristics and population-based metaheuristics [5]. Trajectory-based methods operate on a single solution at any one time and follow a single path or trajectory in the solution space. Population-based metaheuristics operate on multiple candidate solutions at the same time resulting in multiple paths being followed in the solution space at the same time. In the next few subsections each category is discussed and examples of each category are provided.

2.3.1. Overview of Popular Trajectory-Based Metaheuristics

Over the years many trajectory-based approaches have been proposed. Some approaches have gained in popularity because new ways have been found to improve performance. In some cases new variants to existing approaches have been introduced which outperform their constituents. In this section some of the early methods that are still widely adopted today either as a variant or as an enhancement are considered. Simulated Annealing, Tabu Search, Greedy Randomised Adaptive Search and Variable Neighbourhood Search are presented in that order.

2.3.1.1. Simulated Annealing

Simulated Annealing (SA) is based on the annealing process of metals and is employed to improve initial complete solutions [11]. The search accepts a neighbouring candidate solution only if it is better than the current solution or with a probability p (referred to as the transition probability). This neighbouring candidate solution then becomes the current solution. The transition probability follows the Boltzmann distribution and is calculated using a temperature T which is initialised at the beginning to some user defined maximum value. As the algorithm progresses T is reduced according to a predefined cooling schedule. A bigger difference in cost between the neighbouring candidate solution and the current candidate solution decreases the probability of acceptance of the neighbouring candidate solution whilst a higher temperature increases the probability of acceptance of the neighbouring candidate solution. Therefore, at the beginning of the algorithm the probability of accepting neighbours that are worse than the current candidate solution is high but the probability of acceptance decreases as the temperature

T approaches zero. At this stage the algorithm becomes a simple iterative improvement search method.

The cooling schedule may be implemented in different ways in SA. The study by Elmohammed et al. [12] compared two different types of cooling schedules. The first type considered was geometric cooling which allows for a faster reduction in temperature. The new temperature is calculated using the equation; $T_{new} = \propto T_{current}$ and $0<\propto<1$. The other type considered was adaptive cooling where T_{new} is determined by considering all the costs obtained at $T_{current}$. The lowest cost is then used to determine the new temperature. Adaptive cooling allows for the system to be cooled much slower than geometric cooling. Kusumawardani et al. [13] point out that besides the cooling schedule, the initial temperature and the number of iterations and evaluations performed by the approach at a specific temperature also affect performance. The initial temperature should be high enough to increase the acceptance of new solutions. The number of iterations at a certain temperature improves the likelihood of accepting a worse solution because of the inherent randomness in the search. The FASTSA method by Leite et al. [14] showed that the performance of SA may be improved by reducing the number of evaluations performed by the search in a single iteration. The temperature value was divided into temperature intervals. If there was no permissible move for a solution component in the preceding interval then it was ignored by the search in the current interval thereby, reducing the number of evaluations.

A popular variant to SA is the Great Deluge algorithm (GD) [15]. The GD is based on the analogy that a person will attempt to stay above rising waters in a deluge to avoid drowning. As the algorithm progresses the water level which represents the acceptance criterion for neighbouring solutions also rises. Gradually the acceptance of neighbouring solutions is reduced based on the water level. The rate at which the water rises is determined by a decay function which is based on a user-defined parameter known as the decay rate.

2.3.1.2. Tabu Search

Similar to SA, the Tabu Search method (TS) [6] is also used to improve initial solutions. TS selects the best move from the list of available moves in a single iteration which is similar to a best improvement local search. However, TS makes use of a memory structure known as a tabu list to record past moves made by the search. The tabu list is consulted before any move is made. If the move features in the list then it is not carried out. The tabu list is therefore used to prevent the search from returning to candidate solutions already visited and may even cause the search to move to solutions that are worse than the current candidate solution. Sometimes it may be necessary to allow the search to perform moves that do feature in the tabu list. This may be the case when there is a need to allow the search to revisit areas in the search space found to be promising. In this case aspiration criteria are used. Aspiration criteria enable the search to make moves that are currently not permitted because they feature in the tabu list for a limited period.

The most basic implementation of a tabu list is as a queue of fixed size [16]. As a new move is added to the list the oldest move is removed. A dynamic list was used in the study by Gaspero and Schaerf [16] where a move was kept in the list for *k* moves. The value of *k* was a random number chosen between a user-defined upper and lower bound. Lawal et al. [17] extended the TS approach by applying a weighting system to the hard and soft constraints. White and Xie [18] showed that using a longer tabu list in longer-term memory may also improve performance. Instead of using the tabu list to record moves, Kendall and Hussain [19] recorded low-level heuristics already applied in their hyper-heuristic approach. Hyper-heuristics search the heuristic space to determine which low-level heuristic to apply next. The hyper-heuristic used considered heuristics that were not in the tabu list and chose one to apply next. The study showed that keeping good performing heuristics too long in the tabu list decreased overall performance. On the other hand, if the tabu duration is too short the search is limited to a small solution space because of the increased likelihood of reusing the same heuristic.

2.3.1.3. Greedy Randomised Adaptive Search Procedure

The Greedy Randomised Adaptive Search Procedure (GRASP) [20] [21] is another popular metaheuristic. The basic approach consists of two phases. In the first phase a solution is constructed and in the second phase it is improved upon using local search. The construction phase is a multi-start process that is similar to the semi-greedy heuristic proposed by Hart and Shogan [22]. Solution elements are incorporated one at a time into the partial candidate solution until it is complete. The next solution element to be added is chosen at random from a list of solution elements referred to as a Restricted Candidate List (RCL). The RCL is initialised at the beginning of the algorithm by including all the candidate solution components. The RCL is rebuilt at each iteration and consists of those solution elements which when incorporated into the current candidate partial solution results in the least increase in cost in the candidate solution. Some search strategies that may be followed in the second phase include the best-improving strategy and the first-improving strategy [23]. The latter is faster because it returns the first solution found to be better whilst the former considers all the neighbouring solutions before returning the best one found.

There are two approaches that may be followed when deciding on the composition of the RCL. The first approach is the use of a cardinality-based criterion whereby the RCL is limited to a specific number of solution elements [20]. The number of elements is provided upfront before the start of the construction process and the solution elements considered for the RCL are the ones which provide the least incremental cost to the partial solution for that iteration step. The second approach makes use of a value-based criterion. In this case the RCL is made up of all solution elements x whose incremental cost is $cost(x) \in \left[cost^{min}, cost^{min} + \propto (cost^{max} - cost^{min})\right]$ and $\propto \in [0,1]$. If $\propto = 0$ then the strategy is completely greedy because only the best performing solution element at that iteration is chosen for addition into the partial solution. On the other hand, if $\propto = 1$ then the strategy is completely random because any solution element from those available at that iteration may be chosen for addition to the partial solution.

GRASP is easily implemented and hybridised with other local search algorithms like TS for the solution improvement phase [4]. Souza et al. [24] and Prais and Ribeiro [25] make use of TS in the local search phase of GRASP. GRASP was also successfully hybridised with population-based methods. Ahuja et al. [26] made use of GRASP to generate the initial population for a genetic algorithm when solving the Quadratic Assignment Problem. Rocha et al. [27] also implemented GRASP for the Course Timetabling Problem. The classes that were the most difficult in terms of the hard constraint were scheduled first in the construction phase. The soft constraint cost of each class was considered in the construction of the RCL. Local search was used in the improvement phase of GRASP to improve the solution from the initial phase.

2.3.1.4. Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) [28] makes use of a set of neighbourhoods. The set of neighbourhoods are defined upfront. The neighbourhoods may be arbitrarily chosen or may be a set of nested neighbourhoods with increasing size. A set of nested neighbourhoods is one where the first neighbourhood is a subset of the second one which is turn is a subset of the third and so on. Once VNS finds a local optimum in one neighbourhood, it escapes that optimum by changing the neighbourhood. VNS consists of three phases, namely, a shaking phase, a local search phase and a move phase. In the shaking phase a candidate solution in the k^{th} neighbourhood is randomly selected to be the local search starting point. Local search is performed to improve the current solution. After the local search, the new candidate solution replaces the current candidate solution if it is better and a new neighbourhood structure is considered.

Burke et al. [29] investigated two variants of VNS to solve the Examination Timetabling Problem. In the first variation the search moved to a new neighbourhood only when it was unable to improve the solution any further in the current neighbourhood. This is different from the standard VNS where the neighbourhoods are recycled independent of the behaviour of the search. The second variant made use of a Genetic Algorithm (GA) to evolve a fixed set of neighbourhoods. Each individual in the population represented an ordered list of neighbourhoods which was used by VNS to determine the order in which the neighbourhoods were searched.

More recently, Alegfragis et al. [30] proposed a VNS framework for the Uncapacitated Examination Timetabling Problem. The framework allows for the use of multiple metaheuristic algorithms such as SA and Great Deluge. Associated with each metaheuristic is a set of defined local search moves and neighbourhoods that are implemented to improve the solution. In the beginning VNS randomly choses a metaheuristic and applies to an initial solution in an attempt to improve it. If the solution cannot be further improved upon then VNS considers a different metaheuristic to improve the same solution

2.3.2. Overview of Popular Population-Based Metaheuristics

There have been many population-based methods developed over the years. Evolutionary Algorithms is a popular class of methods that are based on the natural selection and evolutionary processes found in nature [31]. Genetic Algorithms belong to this class of methods and these methods are discussed in this work. Another class of population-based methods that is also inspired by nature are Swarm Intelligence approaches. Swarms such as bees tend to overcome their limited capabilities to find food by collectively collaborating with each other and interacting with their environment [32]. Swarm Intelligence approaches model the collective behaviour of swarms and their interactions with their environment and how they use this global information to solve complex problems. The Ant Colony Optimisation method and Particle Swarm Optimisation method from this class are discussed in this work.

2.3.2.1. Genetic Algorithms

In Genetic Algorithms (GA) a population of individuals is evolved from one generation to the next until convergence occurs or the termination criteria have been met. Each individual represents a candidate solution to the problem at hand [31]. The process begins with an initial population which is then evaluated using a fitness function. Based on the selection method adopted, individuals from the current population are chosen as parents for the individuals of the next generation. The parents are subjected to crossover and mutation to create offspring that make up the new generation. The new generation then replaces the current generation. This process continues until the termination criteria have been met. On completion, only the fittest solution in the current generation is returned.

Yadav and Sohal [33] present a study of the various selection techniques adopted by GA for the selection of individuals to act as parents from the current generation. One of the most commonly used methods is tournament selection. In tournament selection a fixed number of randomly chosen individuals are first selected from the population. The fittest individual in that selection is then returned. The crossover operator performs local search and promotes convergence of the algorithm. Lim et al. [34] provide a survey of the more commonly used crossover and mutation operators in GA. The most basic is the single-point crossover. In this crossover operator, a random point is chosen in the chromosomes of each parent and the information after that point is exchanged between the two parents to create the offspring. The mutation operator maintains diversity in the population. The rate at which mutation occurs is lower than the rate at which crossover occurs. Mutation involves making small random alterations to the chromosomes of the offspring. A wide selection of mutation operators has been used. For example, timetables in timetabling problems may be mutated by moving a randomly selected exam from one randomly chosen period to another in the same timetable.

Genetic Algorithms (GAs) has been implemented in various ways to improve performance. Hong et al. [35] point out that the performance of a GA is directly linked to the choice of crossover operator and mutation operator used. Sastry et al. [36] also noted this observation and proposed the Dynamic GA (DGA) which implements multiple crossover and mutation operators instead of the traditional GA which makes use of a single crossover operator and a single mutation operator. The GA can be applied in more than one phase in an approach. The two-phased approach by Pillay and Banzhaf [37] made use of a GA in the first phase to find feasible solutions and in the second phase a different GA was used to improve the individuals evolved from the first phase.

2.3.2.2. Ant Colony Optimisation

Ant Colony Optimisation (ACO) is based on the foraging behaviour of real ants where ants use pheromone deposits to find the shortest path to their food [38]. In the beginning each ant follows a different path to find food. As each ant returns to the nest after finding food it deposits pheromone on its trail. As more ants follow the same path more pheromone is deposited leading to more ants picking up the same trail. This leads to a path reinforcement loop which has been experimentally proven to enable ants to find the shortest path [39]. The pheromone update process in ACO consists of two processes, namely, a pheromone deposit process and a pheromone evaporation process. In the pheromone deposit process, solution components that are associated with high quality solutions have their pheromone values increased making these solution components more desirable to ants in the following iterations. The pheromone deposit process takes place after ants have finished with solution construction. The pheromone evaporation process takes place during solution construction and the process involves reducing the pheromone values on solution components. The aim is to make some solution components less desirable to ants during construction. Once a solution has been completed the entire process is restarted. This continues until no new solutions can be found or the termination criterion has been met.

ACO was first applied to the Traveling Salesman Problem and then adapted later on to solve other problems like the ETP [40] and the Job Scheduling Problem [41]. ACO has many variants [42]. A popular variant to ACO is the Min-Max Ant System (MMAS) [43]. In MMAS only the ant which returns the best candidate solution in a cycle is allowed to deposit pheromone. Another variant is the Rank-Based Ant System (ASrank) [44] where ants are ranked according to trail lengths. Only a fixed number of the best ants are allowed to make deposits. The deposits are weighted by allowing the ants with shorter paths to make larger deposits. Eley [40] noted that a large difference in pheromone deposit values may result in search stagnation due to the likelihood of all the ants returning to the same solution. The MMAS also introduces a range of possible pheromone values to limit search stagnation. Djannaty and Mirzaei [45] improved the performance of MMAS by incorporating local search in the form of the Great Deluge, a variant of SA. Local search was used to improve the best solution in each cycle before the pheromone update process. An improved local search incorporating memory was also used in the hybrid ACO approach by Abounacer et al. [46]. The local search method was similar to TS in that it kept a record of explored solutions to avoid recycling and made use of backtracking to improve the search process.

2.3.2.3. Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) [47] is based on the flocking behaviour of birds. PSO makes use of a number of particles to move through the search space to find the global best position [32] [47]. Each particle represents a candidate solution to the problem and has a velocity and a position in the search space. The particle also keeps record of the best position found in its path so far. After each evolutionary cycle the velocity of each particle is calculated by adding its current velocity to the influence from its current best position and the swarm's best position. An inertia weight variable *w* is used to control the influence of previous velocities on the particle's new velocity [48]. The new position of each particle is then computed by using the particle's updated velocity. Gradually, the particles move towards the global best position in the swarm. Every particle swarm has a defined topology which describes how the particles are connected. The neighbourhood of a particle is then the set of particles in the swarm to which it is topologically connected. The neighbourhood of a particle may be a subset of the swarm or contain the entire swarm itself.

Fealko [49] used a constraint version of the traditional PSO to ensure that only feasible regions of the solution space are considered. The constrained PSO made use of a low-level construction heuristic to generate an initial population of particles representing only feasible candidate solutions and the modified PSO only accepted feasible solutions during the optimisation process. Chu et al. [50] proposed a self-mutated PSO. The movement of each particle is processed by choosing a random solution component from the particle's best solution found so far and a random solution component from the best solution found by the swarm so far.

2.4. Diversification and Intensification Mechanisms in Metaheuristics

In this section the diversification and intensification capabilities of the metaheuristics presented earlier in this chapter are discussed. Both these capabilities are defined and discussed earlier in section 2.2. The search capabilities of trajectory-based methods are first looked at followed by population-based methods.

2.4.1. Trajectory-Based Methods

Trajectory-based methods have good intensification capabilities [5]. They are able to investigate good performing areas to find better solutions. Once the search finds a local optimum through exploitation it needs to be able to move out of the optimum in pursuit of the global optimum by accepting uphill moves. Otherwise the search will return a solution which is suboptimal. Spending too much time stuck in a local optimum is costly in terms of processing time. Therefore the performance of the search is highly dependent on the effectiveness and efficiency of the mechanism used to escape the local optimum. The escape mechanisms adopted by the trajectory-based methods and their search capabilities are presented in turn.

Tabu Search (TS) maintains a tabu list which plays a key role in the intensification and diversification capabilities of the search. The search is prohibited from carrying out moves that feature in the list. This may cause the search to carry out worse or uphill moves. In doing so the search is able to go beyond any local optima it is stuck in. The use of aspiration criteria also improves diversification because the search is allowed to perform prohibited moves that feature in the tabu list. The way the tabu list is implemented also determines the performance of the search. There have been many different approaches to the implementation. Blum and Roli [51] state that a smaller sized (tenure) tabu list causes the search to focus on small areas of the search space. A larger tenure causes the search to explore larger search areas because a greater number of moves in the neighbourhood is prohibited. Battiti and Tecchiolli [52] noticed that an increase in the number of moves being repeated is an indication that the search required more diversification. This was done by dynamically increasing the tabu tenure for a short while. White and Xie [18] also used the frequency of moves to improve performance. A frequency table was kept in memory and if a move was repeated the table was updated accordingly. In this way moves with high frequency values were avoided.

Simulated Annealing (SA) makes use of a cooling schedule and a temperature T to control the level of intensification and diversification and to escape from any local optima. SA is more likely to accept uphill moves in the beginning stages when T is relatively high. As the algorithm progresses T is decreased according to a cooling schedule which in turns decreases the probability of accepting uphill moves [53] [54]. In the beginning more diversification takes place. As the algorithm progresses the level of diversification decreases whilst that of intensification increases. Blum and Roli [51] state that convergence to the optima is likely if the cooling schedule is slow but this is not practical in real-world applications where faster cooling rates are required. It is also observed that if the system cools too quickly then the search may be unable to move out of local optima because the probability of accepting uphill moves is too low. One of the approaches adopted to solve this problem is the use of non-monotonic cooling schedules. The traditional SA makes use of a monotonic cooling schedule because the temperature only decreases as the algorithm progresses. Non-monotonic cooling schedules allow for the temperature to be increased in some way during the search. This is referred to as reheating and has been used successfully [51]. Elmohammed et al. [12] used a reheating technique to explicitly increase the diversification capabilities of the search. The authors state that if the best cost so far is high then the temperature required to escape the local minimum must be just as high. Abrahamson et al. [55] also made use of a similar heating technique as a function of cost where the reheating technique considered the heat of the current system and the cost of the current best solution found. Cooling schedules enable the search to balance the amount of intensification and diversification performed. It is also shown by Abdullah et al. [56] that the choice of neighbourhood may lead to better quality solutions. The same conclusion was reached by Thomson and Dowsland [57] when comparing the results from three different move operators. A change in the move operator changes the neighbourhood which affects performance.

The RCL in GRASP plays a key role in the exploratory capabilities of the search because the next solution element to be added to the partial solution is randomly chosen from the RCL. The size of the RCL is another important factor that determines performance. The smaller the size, the more deterministic the approach becomes because the choice of solution elements is limited. Prais and Ribeiro [25] argue that allowing for the size of the RCL to change during the construction phase enabled the approach to construct more diverse solutions. A Reactive GRASP was proposed for the Matrix Composition Problem and it outperformed the standard GRASP [25]. The information gathered about the quality of previously found solutions was used to self-tune the restrictiveness of the RCL by building a discrete list of acceptable values to be used for the size of the RCL. The size of the RCL was then randomly selected from this built list at every iteration in the construction phase. This approach appeared to be more robust because there was no need to manually define the size of the RCL. Burke et al. [4] also acknowledged the size of the RCL affects the quality of results obtained and proposed a hyper-heuristic approach to adaptively determine the size of the RCL at each iteration. The hyper-heuristic operated on two low-level construction heuristics, namely, the saturation degree and the largest weighted degree. Blum and Roli [51] argue that the choice of construction heuristic(s) employed in the construction phase is an important factor in determining the quality of the initial solutions constructed. Using a good quality initial solution enables GRASP to have a good starting point for the second phase leading to better overall performance.

In Variable Neighbourhood Search (VNS) the number of neighbourhood structures used determines the amount of intensification and diversification. If the number is small then more intensification takes place as previously found promising areas in the search space are revisited more often. On the other hand, a large number of neighbourhood structures allows for more diversification. Also the search achieves diversification by changing the neighbourhood structure when the current candidate solution cannot be improved.

2.4.2. Population-Based Methods

Population based methods consider multiple points in the search space at the same time because they operate on multiple candidate solutions at the same time. As a result they have an inherent ability to quickly sample the search space and have strong diversification capabilities. In pursuit of the global optima the search may converge to a local optimum and return less desirable results. This issue is known as premature convergence and good performing population-based methods need to effectively and efficiently deal with this issue.

Genetic Algorithms (GAs) evolve the population from one generation to the next by using genetic operators. The traditional GA uses a crossover operator to exploit the search space by recombining the traits of parents to create offspring. However, over many generations the crossover operator causes the individuals to become more similar to each other. As diversity in the population decreases, the diversification capability of the GA also decreases. To solve this problem the GA makes use of the mutation operator to maintain diversity and increase diversification. The mutation operator modifies individuals in the population and in doing so moves the search to different areas in the search space. Conversely, too high diversity in the population can lead to deterioration in performance [58]. A user-defined probability is used to determine the rate of mutation and limit the diversity in the population. Other techniques have been used in GA to preserve diversity and to avoid premature convergence. Gupta and Ghafir [59] provide a review of methods used to maintain diversity. One such technique is Crowding introduced by de Jong [60]. Crowding takes place at the replacement stage in the GA after the new offspring have been created. For example, in the Deterministic Crowding approach, offspring are paired with the parents they are similar to. If the offspring is fitter than the parent it is paired with, then it takes the place of that parent in the next generation. Another key design issue in GA is the size of the initial population. Larger populations enable the search to be more exploratory as more points in the search space are sampled. However, as the population size is increased the overhead in terms of processing time and resources are increased. A population size that is too large causes very slow convergence which may be impractical in some applications. A small population size also affects performance as diversification is reduced and may result in premature convergence [61]. Some approaches have been proposed to improve processing times when larger populations are required. Parallel Genetic Algorithms (PGAs) may be used to reduce processing times when larger population sizes are required because it is implemented to run over multiple processors instead of a single serial machine [62] [63]. Pappu et al. [64] proposed a PGA for the Examination Timetabling Problem by implementing a client-server architecture. The initial population was generated on the server. Thereafter copies of the population was evolved on separate clients. At regular intervals the information from the clients was used to update the population on the server. The design of the system allowed for new clients to be added to increase exploration without the need to disrupt the entire process. The Island Model paradigm is another approach that takes advantage of the architecture of PGA when dealing with large population sizes [65]. In this case the initial population is divided into groups and each group is evolved independently on different hardware processes. Corcoran and Wainwright [66] made use of an Island Model in their PGA to solve the Job Scheduling Problem. Vasileios et al. [67] implemented a PGA on a Graphics Processor Unit (GPU) for the Examination Timetabling Problem. It was argued that a GPU makes use of expanded memory and threads to speed up processing times. The use of threads allowed for the crossover and mutation operations to be performed in parallel. The approach was more exploratory as it made use of larger population sizes. Umbarkar and Joshi [68] provide a review of this and other parallel computing paradigms that have been used to reduce GA runtimes.

Ant Colony Optimisation (ACO) is based on how biological ants follow trails produced by pheromone deposits to find the shortest path to their food. Premature convergence is also a concern with ACO. The method converges when more and more ants begin to follow paths that have common points. This leads to a loss in diversity. A high diversity means that most of the ants are not influenced by the current best path found. A low diversity means that most of the ants are following the best ant (current best path) and convergence has occurred. In the standard ACO the pheromone evaporation rate may be used to balance the level of diversification and intensification performed by the search and avoid premature convergence. A lower rate of evaporation increases diversification because it causes the ants to traverse suboptimal paths which improves the likelihood of finding new paths [69]. Conversely, a higher rate of evaporation has an intensifying effect. The ants are forced to converge on paths that are traversed the most in order to find shorter paths. Other approaches have been proposed to improve diversity. Mohsen [70] put forward a hybrid ACO method to solve the Travelling

Salesman Problem (TSP). ACO was used to create an initial population. The population diversity was then calculated using a method based on Euclidean distance. If the diversity was too low then a mutation operator was used to increase diversity As a result, diversification was increased because mutation moves the search to a different area in the search space. On the other hand, if the diversity was too high then a version of Simulated Annealing was used to increase intensification. The Max-Min Ant System (MMAS) introduced by Stutzle and Hoos [71] limited the pheromone trails values with an interval defined upfront to avoid search stagnation where the same solution is returned by multiple ants. In this study only the best performing ant was allowed to update pheromone trails and two alternatives were considered. The trails were either updated by the current best performing ant in each iteration or the global best performing ant from all previous iterations or evolutionary cycles. Chen and Liu [72] state that a shortcoming of MMAS is the slow convergence rate due to the limits imposed on the pheromone trail values. The authors proposed a multi-colony system for TSP on top of a cluster of processors. The master processor keeps track of the global best solution found so far by all colonies. Each colony runs on a slave processor. Once a colony converges to local optima it is reported to the master and a new colony is initialised to run in its place. The method reported convergence times faster than parallel independent runs. The authors showed that multi-colony systems have an increased capability to escape local optima compared to single colony systems.

Particle Swarm Optimisation (PSO) is based on the flocking behaviour of birds. Particles represent candidate solutions and they fly through a multi-dimensional search space in pursuit of a global optima. Each particle has a position and velocity and is influenced by the global information of the flock. Earlier versions made use of velocity limits to control the level of intensification and diversification in the search process. If a particle's velocity was higher than a defined limit then it was reset to the value of the limit. This velocity clamp mechanism improved convergence but was ineffective in enabling the search to escape a local optima [73]. Shi and Eberhart [74] introduced the inertia weight *w* to balance the amount of diversification and intensification and overcome premature convergence. *W* models the inertia of the particle and represents its resistance to steering. The inertia weight *w* controls the influence of previous velocities on the particle's current velocity. By gradually decreasing *w* in a linear fashion, diversification is reduced and intensification is increased. Cheng and Shi [73] state that the capability of the search is limited because of the difficulty in dynamically adjusting *w* to control the level of exploration and exploitation especially in complex or large-scale problems. The authors propose a novel equation to control population diversity by making use of the average velocity of the swarm and a user-defined parameter value. The equation is used to vary the current position of a particle in order to increase or decrease swarm diversity. There have been many hybrid implementations of PSO which incorporated local search to improve performance [75]. Abayomi et al. [76] combined PSO with local search to successfully develop an efficient automatic exam scheduler for a university in Nigeria. Tassopoulos [77] also combined local search with PSO in their hybrid method to solve high school timetabling problems. A move operator was used to change the particle's current position. It swapped courses between two timeslots in a random fashion. The performance of the hybrid was shown to be superior to the GA and SA for the same problem sets.

2.5. Summary

Most combinatorial optimisation problems cannot be solved optimally in polynomial time. In such cases it is more desirable to find near-optimal solutions in finite time. Metaheuristics have received much attention over the years because they are able to find good performing solutions in reasonable time to difficult problems. In general, a metaheuristic makes use of higher-level strategies to overcome the limitations of lower level search heuristics to find better solutions.

This means that they are able to strike a better balance in diversification and intensification. Good performing methods need to perform adequate diversification in order to find better performing search areas. At the same, the method needs to be effective in searching good areas.

There are two categories of metaheuristics, namely, trajectory-based methods and population-based methods. The former considers one solution at a time and follows a single trajectory in the search space. The latter considers multiple solutions at a time. A key problem is the issue of local optima traps in trajectory-based methods where the search is ended prematurely leading to poor performance. Such methods need to have a way to overcome this challenge. For example, TS makes use of a tabu-list. The search is not allowed to perform moves that feature in the tabu-list. This causes the search to eventually accept an uphill (worse) move and is then able to move out of the local optimum. The tenure of the list is also important. Tabu-lists that are too short may cause the search to revisit old moves more often. This limits exploration. Other trajectory-based methods covered in this chapter include SA, GRASP and VNS. SA makes use of a cooling schedule to control the amount of intensification and diversification performed by the search. If the temperature of the system is cooled too quickly then the level of exploration is reduced. GRASP makes use of a RCL to control the amount of intensification and diversification performed by the search.

Population-based methods were also discussed in this chapter. A key challenge with these methods is premature convergence. This happens when the search converges too quickly and further exploration is halted. GAs are able to overcome this issue to some extent in different ways. Making use of larger populations improves exploration and slows shown convergence. In some cases this may be undesirable particularly if processing time is an issue and a solution needs to be found in a reasonable time. The GA also makes use of mutation to increase diversity in the population. By increasing the population diversity the convergence is slowed. Again, there needs to be a fine balance as too much diversity can lead to poor performance. It is important for convergence to happen in an acceptable time. Other population-based methods discussed include Ant Colony Optimization and Particle Swarm Optimization. Both these methods are inspired by nature and are based on the foraging behaviour of ants and bird respectively. The diversification and intensification mechanisms of these methods are also elaborated on. In next chapter the Examination Timetabling Problem is introduced and defined.

CHAPTER 3: EXAMINATION TIMETABLING PROBLEM

The Examination Timetabling Problem (ETP) is defined in this chapter along with the best performing methods for this problem domain. The chapter is structured as follows; Section 3.1 defines the ETP. Section 3.2 presents the benchmark sets used to represent the capacitated version of the ETP and some of the best performing approaches for these benchmarks. Section 3.3 focuses on the benchmark set used to represent the incapacitated version of the ETP and the best performing approaches for that set. A critical analysis of the literature is given in Section 3.4 and section 3.5 concludes the chapter.

3.1. Examination Timetabling Problem Defined

The Timetabling Problem remains a well-studied one. The problem is a combinatorial one and in some cases optimisation by exact techniques is not possible in an acceptable time. According to Lawler [78], the mathematical analysis of an arrangement of discrete objects or the ordering thereof is a combinatorial one. However, in most cases it is not necessary to enumerate all the possible arrangements but rather to find the optimal arrangement which Lawler defines as combinatorial optimisation. Wren [79] further categorized the Timetabling Problem as the allocation of a set of resources to a set of objects in space time whilst satisfying a given set of problem constraints. Burke et al. [80] provide a more general definition for the Timetabling Problem. The problem is defined as a set of parameters T, R, M, C which are finite sets that represent timeslots, resources, meetings and constraints respectively. The aim is then to find the best allocation of resources and times for meetings. The allocation should be one that satisfies all the problem constraints as far as possible. Schaerf [81] further categorizes timetabling into three main focus areas, namely, school timetabling, university course timetabling and examination timetabling. School timetabling involves organising school resources in a way that satisfies a set of requirements set by the school. More specifically, it is the scheduling of one or more tuples to timetable slots. Each tuple is a set of resources such as teachers, subjects and classrooms. Course timetabling is similar to school timetabling. It involves scheduling a set of courses to a limited number of timetable slots. Each course is taught to a group of students by a specific lecturer in a specific lecture room. Lastly, examination timetabling is the scheduling of examinations to timetable slots. An exam is taken by a group of students in a specific venue.

All timetabling problems have a set of constraints that need to be satisfied. There are two types of constraints, namely, hard constraints and soft constraints [82]. In order for a timetable to be usable (feasible) it must satisfy all hard constraints. For example, a hard constraint for the school timetabling is that no teacher must be allocated to different classrooms in the same timetable slot. In the case of examination timetabling, a hard constraint is that exams that have common students must not be allocated to the same timetable slot. It is not necessary to satisfy all soft constraints in order for a timetable to be feasible. Also, it may not be possible to satisfy all soft constraints because some may be contradictory. Instead, the quality of a timetable is determined by the extent to which all soft constraints are satisfied. An example of a soft constraint for school timetabling is that teachers may have individual preferences when it comes to teaching specific subjects at specific times. An example of a soft constraint for examination timetabling is the requirement that the scheduling of exams with different durations to the same room in the same timetable slot should be avoided to minimise student movements during the exam.

There are two versions of the ETP, namely, the capacitated and uncapacitated version. In the capacitated version the sizes of the rooms used to write the exams need to be considered whereas, in the uncapacitated version there is no limit on the room sizes. Each version is represented by one or more benchmark sets in the literature.

3.2. The Uncapacitated Version of ETP

In the uncapacitated version of the ETP, the capacities of the exam venues or rooms are not considered. Room sizes are assumed to be unlimited. This version is represented by the Carter benchmark set which is introduced in section 3.2.1. This is followed by section 3.2.2 which presents the best performing approaches for this benchmark set.

3.2.1. The Carter Benchmark Set

The Carter benchmark set was introduced in 1996 [83]. It consists of thirteen instances taken from selected schools and universities in Canada, America, United Kingdom and Middle East. The benchmark set is shown in table 3.1. Each column gives the characteristics of each instance, namely, number of periods/timeslots, number of exams, number of students and the conflict density. The conflict density indicates the proportion of students who write the same exams. Hec-s-92 has the highest conflict density which indicates a high proportion of students sit for the same exams.

Table 3.1 The Carter Benchmark Set

Instance	Number of Periods	Number of Exams	Number of Students	Conflict Density
car-f-92	32	543	18419	0.14
car-s-91	35	682	16925	0.13
ear-f-83	24	190	1125	0.27
hec-s-92	18	81	2823	0.42

kfu-s-93	20	461	5349	0.06
lse-f-91	18	381	2726	0.06
rye-s-93	23	486	11483	0.08
sta-f-83	13	139	611	0.14
tre-s-92	23	261	4360	0.18
uta-s-92	35	622	21266	0.13
ute-s-92	10	184	2749	0.08
yor-f-83	21	181	941	0.29
pur-s-93	42	2419	30029	0.03

This benchmark set has only one hard constraint which is that no student should be scheduled to take two of more exams in the same period. The spacing of exams for students has a direct influence on the pass rate. Also students tend to prefer timetables that have an element of fairness where the timetable does not disadvantage certain students [84]. The soft constraint that measures the spacing of exams which have common students aims to address this issue by promoting timetables that are more evenly spaced. The exam spread is calculated as shown in equation 3.1 below and timetables that minimize this value are preferred.

$$Spread = \frac{\sum w(D_{ij}) com_{ij}}{S} \tag{3.1}$$

where:

1) $w(D_{ij})$ is distance in periods between $exam_i$ and $exam_j$.

2) com_{ij} is the number of students sitting for both exams

3) S is the total number of students.

4) $w(D_{ij}) = 2^{5-D_{ij}}$ where $D_{ij} \in \{1,2,3,4,5\}$, 0 otherwise.

3.2.2. Best Performing Methods for the Carter Benchmark Set

Burke and Bykov [85] built on the success of the Great Deluge in solving timetabling problems by proposing a variant known as the Flex-Deluge. The Great Deluge accepts both uphill (worse) and downhill (better) moves. However, it was observed that by modifying the acceptance criterion the performance may be improved. The Flex-Deluge introduces an additional parameter to slow the rate of acceptance of both uphill and downhill moves. This allows the Flex-Deluge to spend more time than the Great Deluge exploring and finding better solutions.

Leite et al. [86] also made use of the Great Deluge in an evolutionary approach by proposing a memetic algorithm. A memetic algorithm is a GA that incorporates local search to improve the fitness of the offspring after the recombination process. The initial population was organised into groups (complexes) and each complex was evolved independently. The Great Deluge (see section 2.3.1.1) was used to improve solutions after the recombination process. After each evolutionary step, the complexes were shuffled by reorganising the individuals. The shuffling of complexes and use of recombination between complexes enabled the search to explore new areas.

Later on in 2018, Leite et al. [87] proposed a cellular memetic algorithm combined with the Threshold Algorithm (TA). TA is a variation to Simulated Annealing (SA) (see section 2.3.1.1). In SA the selection criterion is probabilistic whereas in TA it is deterministic and a solution is only accepted if it is below the annealing threshold. Cellular memetic algorithms organise the population in a connected graph where each vertex represents an individual that is linked to its neighbours. During the recombination process an individual is recombined with its nearest neighbour in the graph allowing for greater diversity in the chosen parents.

Bellio et al. [88] also made use of SA in an approach that has two stages. SA was implemented for both stages. The aim of the first stage was to render a randomly generated timetable feasible. This was achieved by performing moves that reduced examination conflicts until there were no hard constraint violations. The aim of the second phase was to improve the quality of the timetable from the first phase by making use of move operators that reduced the soft constraint cost.

Rather than working on improving complete solutions Mandal and Kahar [89] considered improving partial solutions. After each iteration a user-defined number of exams were assigned to the partial solution and then the Great Deluge (see section 2.3.1.1) was used to improve the solution quality. The process was repeated until all exams were scheduled. The candidate solutions were constructed using different construction heuristics. The saturation degree heuristic performed the best from the four heuristics considered. The saturation degree construction heuristic gives priority to the exam with the least number of feasible period options in the current version of the timetable.

Caramia et al. [90] instead scheduled exams in a greedy fashion. Exams were scheduled to slots that incurred the least cost. Backtracking was used to resolve exam conflicts. Each scheduled exam was assigned a penalty score which was updated after a new exam was scheduled. A move operator was used to reduce exam penalties. In some cases it was observed that after all exams were scheduled some periods remained unused. In this case the cost reduction in moving all the exams to an unused period was computed. Only exams with the highest cost reduction were then moved to the new period.

3.3. The Capacitated Version of ETP

In the capacitated version of the ETP, the capacities of the exam venues or rooms must be considered. In this study, the capacitated version of the ETP is represented by the examination benchmark set released for the International Timetabling competition in 2007 (ITC2007) and the Yeditepe benchmark set. The ITC2007 benchmark set is discussed in section 3.3.1 along with best performing approaches found in literature for this benchmark set in section 3.3.2. The Yeditepe benchmark set is discussed in section 3.3.3 along with best performing approaches found in literature for this benchmark set in section 3.3.4.

3.3.1. The ITC2007 Benchmark Set

The ICT2007 benchmark set was meant to present a realistic view of examination timetabling in practice [91]. It is more constrained than the Carter benchmark set in that it has a richer set of hard and soft constraints. The twelve instances for this benchmark set is shown table 3.2. The columns from left to right in the table show the number of students in a particular problem instance, the number of exams in that instance, the number of rooms, the conflict density and lastly the number of periods.

Table 3.2 The ITC2007 Benchmark Set

Instance	Number of students	Number of exams	Number of rooms	Conflict Density	Number of Periods
1	7891	607	7	0.05	54
2	12743	870	49	0.01	40
3	16439	934	48	0.03	36

4	5045	273	1	0.15	21
5	9253	1018	3	0.009	42
6	7909	242	8	0.06	16
7	14676	1096	15	0.02	80
8	7718	598	8	0.05	80
9	655	169	3	0.08	25
10	1577	214	48	0.05	32
11	16439	934	40	0.03	26
12	1653	78	50	0.18	12

The hard constraints for the ITC2007 are summarized below;

- There must be no student clashes.
- The capacity of each room is fixed and must not be exceeded.
- The exam duration must not exceed the period duration.
- Some exams must be written simultaneously.
- Some exams must be written before others.
- Some exams require exclusive use of the room.

The soft constraints for the ITC2007 are summarized below.

- The number of times a student has consecutive exams in the same day.
- The number of times a student writes more than one exam in the same day.
- The number of times the spacing between exams for a student is less than a specific value is also counted.
- Frontload penalty which penalises large exams that are scheduled later on.

- Exams with different durations being written in the same room are also penalized.
- The scheduling of exams in specific periods incurs certain penalties.
- The scheduling of exams in specific rooms incurs certain penalties.

Some of the soft constraints for this benchmark set are contradictory such as the period spread and front loading constraints. The period spread constraint promotes timetables that have more free periods between exams for students. The front loading constraint penalizes timetables that have larger exams in terms of student numbers scheduled in the last periods. Therefore period spread is restricted by the front loading constraint.

McCollum et al. [91] gives a more detailed description of the constraints and description of the objective function.

3.3.2. Best Performing Methods for the ITC2007 Benchmark Set

The multi-phased approach by Muller [92] performed the best in the Second Timetabling Competition in 2007. An initial solution was constructed in the first phase. The second phase employed a problem specific hill climbing routine to find the local optimum. The solution was further improved upon by a bounded Great Deluge (see section 2.3.1.1) in the improvement phase. The bound value was increased whenever the search reached the lower limit of the bound. The number of times the bound was increased with no improvement in the overall best solution was recorded. If this number exceeded a predefined limit then the upper bound was increased to allow search to move out of the local optimum. Simulated Annealing (SA) (see section 2.3.1.1) was used in the next phase for further improvement of the solution. The method also proved to be versatile as it was applied to the other two tracks in the competition with minor modifications.

The FASTSA method proposed by Leite et al. [14] adopted the same approach as the previous method to resolve conflicts in the initial solution construction process. The method used a variant of SA termed FASTSA. The FASTSA was able to achieve faster processing times over the standard SA by reducing the number of evaluations performed on every iteration. The temperature schedule was segmented into intervals referred to as bins. If an attempt to move an exam in the previous iteration was not successful then it was recorded in an appropriate bin and the search ignored any moves relating to that exam in future iterations. One of the drawbacks of FASTSA was the loss generality that is present in the standard SA.

The multi-staged approach proposed by Gogos et al. [93] was placed second behind Muller in the same competition. GRASP was used to generate the initial solutions. A limited form of backtracking was used to resolve any conflicts in the initial solution construction phase. A tabu list was also used to record which exams were reassigned in the backtracking process. SA with Kempe chains was used in the second phase to refine initial solutions. In the third and last phase each period was inspected in turn using a novel Integer Programming approach to further improve the solutions. In 2010, Gogos et al. [54] proposed several improvements to the approach to achieve better performance. One of the improvements was the addition of more search heuristics.

The cellular memetic approach by Leite et al. [87] also performed well for this benchmark set. It was already discussed in section 3.2.2.

Many of the best performing methods discussed so far incorporate SA or a variation thereof. Bykov and Petrovic [94] state that whilst SA has proven to be effective in solving the examination problem there is a need to consider approaches that are similar to but easier to implement than SA. The Late Acceptance Hill-Climbing Algorithm (LAHC) by Burke et al.

[82] is one such example of a system that is similar to SA without the need for a cooling schedule. Following the success of LAHC in solving the ETP a new hill-climbing method called Step Counting Hill Climbing Algorithm (SCHC) was proposed by Bykov and Petrovic [94]. A cost bound is used as the acceptance criterion for new solutions. It is updated using the current best after a user defined number of steps (iterations). Different variations to SCHC were considered and the results show that the variant where only the accepted moves were counted performed the best.

3.3.3. The Yeditepe Benchmark Set

The Yeditepe benchmark set is made up of eight instances taken from the Yeditepe University from eight semesters over three years [95]. It differs from the other two benchmark sets in that the timetables are a fixed size for all problem instances with each day having only three periods. It differs from the Carter benchmark because the capacity of exam venues must not be exceeded. It is also more realistic than the Carter benchmark set because it provides additional information on the time and day of each period which is absent in the Carter benchmark. The soft constraint is more restrictive than the Carter benchmark because it requires that students must not be scheduled to sit for exams on consecutive periods on the same day. The Yeditepe benchmark set is similar in nature and format to the ITC2007 benchmark set but has fewer constraints. For example, there are no ordering rules when scheduling exams as is the case with ITC2007. As a result the Yeditepe benchmark set provides the researcher with an opportunity to study real-world problems without too many real-world constraints. The three different benchmark marks set provides three levels of difficulty in terms of how constrained the problem is, with the Carter benchmark set being the easiest to solve followed by the Yeditepe benchmark set and then lastly the ITC2007 benchmark set being the hardest.

Parkes et al. [95] provides more details on this set and the definition of its objective function. The Yeditepe benchmark set is listed in table 3.3 followed by the hard and soft constraints of the problem. The columns in the table from left to right show the number of students in that problem instance, the number of exams, the number of enrolments and lastly the conflict density.

Table 3.3: The Yeditepe Benchmark Set

Instance	Number of students	Number of exams	Number of enrolments	Conflict Density
20011	559	126	3486	0.18
20012	591	141	3708	0.18
20013	234	26	447	0.25
20021	826	162	5755	0.18
20022	869	182	5687	0.17
20023	420	38	790	0.2
20031	1125	174	6714	0.15
20032	1185	210	6833	0.14

The hard constraints this benchmark set are:

- No student clashes are allowed.
- The number of students taking an exam cannot exceed the room capacity.

The soft constraints for this benchmark set are:

- The number of times a student is expected to take exams in consecutive periods must be minimized otherwise a penalty is incurred.

3.3.4. Best Performing Methods for the Yeditepe Benchmark Set

Very few approaches have been applied to the Yeditepe benchmark set. The multi-phased method Muller [92] used to win in the Second International Timetabling competition was successfully applied to this set and good results were obtained. This method was discussed in section 3.3.2.

Muklason et al. [84] used a different approach. A survey of student preferences with respect to exam timetables was carried out. The study revealed that students thought of a timetable as being fair if it did not have any exam arrangement that favoured certain students. Fairness was defined as new objective and included in the problem description. To solve the new multi-objective problem a two-phased approach was implemented. Ordering heuristics with Squeaky Wheel optimization [96] were employed in the first phase to construct solutions. To improve the solution in the second phase a hyper-heuristic was implemented to select the heuristic to apply to the solution next and the Great Deluge was used to handle the move acceptance part. The method obtained better results than the approach by Muller for seven problem instances.

3.4. Critical Analysis

A survey of the literature revealed that solving the Examination Timetabling Problem (ETP) is not a trivial task. Trajectory-based methods, population-based methods and hybrid methods were proposed to solve ETP with varying degrees of success. Some of the more popular population-based methods include Evolutionary Algorithms and Ant Systems [75] [97]. Trajectory-based methods like Simulated Annealing and Tabu Search also proved to be popular [98] [6]. One of the challenges encountered in the ETP is that timetables with different structures map to the same objective value. This issue is discussed in more detail below in subsection 3.5.1 and a justification for a structure-based search to address this problem is provided. An analysis of the literature also provided insight on some of the promising directions taken by researchers to solve the ETP. One interesting approach is searching the partial solution space of timetables [3]. Subsection 3.5.2 provides information on this approach and a justification for this approach is also provided.

3.4.1. Justification for using a structure-based search

An analysis of the timetable space reveals that timetables that have different structure may have the same behaviour in terms of the objective value. The challenge is then to decide which of these timetables is suitable. A survey of the best performing approaches for the ETP shows that little or no consideration is given to structure by current solvers. Most of the methods have an objective value bias which means the search is mostly guided by the objective value. Both Tabu Search and Simulated Annealing only accept permitted neighbouring solutions that have a better objective value than that of the current solution. Relying only on the objective value makes the search more prone to local optimum traps and approaches need to have mechanisms in place to deal with this issue. For example, some SA approaches allow for the temperature T to be increased in order to accept worse moves.

Blum et al. [51] state in order to achieve good performance, the search should not be guided solely by the objective value. Existing approaches make use of only the objective value to decide where to move the search and take structure into consideration when applying operators to move the search. In this work a search that combines objective value with structure is proposed to move the search. It is anticipated that the use of structure will reduce the objective value bias which may lead to an improved performance by the search. Regions of the solution space may be identified by considering the structure of the timetables that occupy the same regions. By taking this approach it is hypothesized that although regions may exhibit the same behaviour in terms of the objective value, the same regions will be distinguishable in terms of structure. New regions would be those that are structurally different from ones previously considered.

The SBPSS uses structure explicitly and combines it with the objective value to move the search. The other approaches make use of only the objective value to move the search. Also the other approaches.

3.4.2. Justification for the searching in partial solution spaces

The success of the Developmental Approach (DA) in solving ETP highlighted how searching the space of partial timetables whilst incrementally building them leads to good quality timetables [2]. The DA is a trajectory-based method that terminates once construction of the timetables is completed. Each iteration consists of an exam being scheduled to the partial solution and thereafter a set of hill-climbers are used to improve the partial solution. This approach is different from other popular trajectory-based methods like Tabu Search where solution refinement is only performed on completed solutions [6]. A single trajectory is followed in the search space [75]. Burke et al. [29] state that if a completed solution is too far from the global optimum it may be difficult to reach the optimum especially if its neighbourhood is disconnected. The use of search during the construction of timetables allows the DA to constantly adjust the trajectory of the search in order to improve performance. Mandal and Kahar [89] also adopted this strategy to search the space of partial examination timetables. The method differed from the DA in two ways. Firstly, search was performed after a group of exams were scheduled whereas the DA performed search at every iteration after an exam had been scheduled. Secondly, the DA employed simple hill-climbers for the local search whilst Mandal and Kahar [89] used a more advanced search method in the form of the Great Deluge.

The good performance of methods that search partial solution spaces suggests that this strategy is promising and worth pursuing. Furthermore, the search may be performed at regular or variable intervals during construction. The search itself may be in the form of simple hill-climbers or using other approaches like the Great Deluge.

3.5. Summary

Timetabling problems are concerned with the allocation of meetings to timeslots whilst making the best use of resources within a set of constraints. Examination Timetabling falls in this category and is the scheduling of examinations to timeslots. The problem has a set of hard and soft constraints. A hard constraint may be one where the capacity of a venue must not be exceeded. A soft constraint may be one where exams need to be evenly spread out. In order for a timetable to be feasible it must meet all the hard constraints. It is not always possible for all the soft constraints to be met as some may be contradictory. The extent to which the soft constraints are satisfied represents the quality of the timetable. In this study three benchmark problem sets are used to provide an empirical evaluation of the proposed approach. The first is the Carter benchmark set which represents the uncapacitated version of the Examination Timetabling Problem (ETP). The capacity of examination venues are not considered for this set of problems. The Carter benchmark consists of thirteen problem instances taken from various universities and schools across Canada. The only hard constraint is that no student must be scheduled to write two or more exams in the same period. The soft constraint cost measures the spread of exams for students. The next benchmark set used is the Yeditepe Benchmark set. This set is of importance because it consists of real-world problems from the University of Yeditepe.

It has eight problem instances and represents the capacitated variant of the Examination Timetabling Problem. The capacity of examination venues must be not be exceeded. The soft constraint is that students must not be scheduled to sit for exams on consecutive periods on the same day. The timetables are a fixed size with each day having only three periods. The ITC2007 benchmark set is the most constraint of the three benchmarks and resembles real-world problems more closely. It is made of twelve problem instances and represents the capacitated version of ETP. This benchmark set has hard constraints for both the venues and the timeslots. For example, some examinations require exclusive use of venues and some examinations need to be written in a particular order or at the same time. For each benchmark set the best performing approaches found in literature are discussed in this chapter.

A critical analysis of the literature was also carried out in this chapter. A case is made for the use of a search that combines both the objective value and structure. It was argued that making use of objective value alone to guide the search is not enough to improve performance. The use of structure with the objective value to guide the search has been proposed. One of the reasons is that regions that have similar objective values may have different structure. By using structure the search may then be able to find new regions that are structurally different. A case was also made for searching partial solution spaces whilst incrementally constructing solutions. The Developmental Approach (DA) was cited as an example of an approach that proved good performance can be obtained for this problem domain. It searches the partial timetable space whilst constructing the timetables. In the next chapter the methodology adopted in this study is presented.

CHAPTER 4: RESEARCH METHODOLOGY

In this chapter the methodology adopted in this research is described. In the first section a summary of the popular research methodologies in the field of Computer Science is given. In section 4.2 reasons are given for the choice of methodology used in this work. Section 4.3 outlines the methodology steps. Section 4.4 lists the benchmark sets that are used to test the proposed method. Section 4.5 provides details of the technical specifications. Section 4.6 describes the parameter tuning process used in this study. The chapter is summarized in section 4.7.

4.1. Research Methodologies

According to Bakar [99] the main objective of research is add something new to the body of knowledge. It was also noted that although Computer Science itself is a highly complex and technical field, the applicability and usability by everyone should be considered. Deyemer [100] argues that research in Computer Science has many branches or areas of interest. It has roots in both mathematics and engineering. This has resulted in some methodologies being based on mathematics such as axioms and proofs and, on the other hand, some methodologies being based on approaches in engineering.

Ramesh et al. [101] proposes a way to characterize research in Computer Science by considering certain key areas such as how the research was done, the level of analysis undertaken in the study and so forth. The same work made use of four classifications, namely, mathematical studies, simulation, concept implementation and laboratory experiment. Work that made use of mathematical techniques was classified under mathematical studies. Simulation referred to work that made use of simulations as their primary methods. Concept implementation referred to those studies where a prototype was developed to demonstrate a proof. Lastly, laboratory experiment referred to work where a newly proposed method (system) was compared to existing methods (system). However, it was noted not all the classifications are related to research in Computer Science. Another classification was proposed by Johnson [102]. It consisted of four research methodologies commonly adopted by researchers in Computer Science, namely, proof by demonstration, the empirical method, proof using mathematical means and hermeneutics.

The proof by demonstration methodology involves proposing an initial solution and then recording the results of the implementation. The recorded results are then used to refine the solution. This process is repeated until the objectives have been achieved or until no further refinement is possible. A drawback of this methodology is that no hypothesis is proposed prior to the solution being built. Another challenge is that in some cases the initial solution may fail completely and not yield any conclusions. The empirical method moves away from developing artefacts. Instead a hypothesis is formulated at the beginning of the process. The method used to test the hypothesis is then outlined because it is important that the test is repeatable to enable other researchers to assess the method. The results are recorded and used to make an informed decision on whether to accept or reject the initial hypothesis. This method also has some challenges. It is difficult to keep the control variables such as the environment and computing platforms the same to ensure the tests are repeatable to draw the same conclusions. To overcome the issue of maintaining an unbiased test, the proof by mathematical means methodology was proposed. Mathematical derivations are used to accept or refute any inferences at the beginning of the process. Hermeneutics is similar to the proof by demonstration methodology but instead of being objective in the testing process the system is implemented in its intended environment. By using subjective testing a more realistic picture of the success of the system can be realised.

4.2. Justification for chosen research methodology

As stated in Chapter 1 the objective of this work is to develop a novel approach to solve the Examination Timetabling Problem (ETP). Furthermore, the proposed approach should perform well for all versions of ETP using the same parameter tuning process and values. This would ensure that the approach is a general one which works well for the whole problem domain.

According to Baker [99] one of the key research activities is the choice of research methodology used. It is also noted that the methodology chosen must be suitable to ensure that the conclusions drawn are accepted. As stated in the previous section there are four widely used methodologies in computer science, namely, namely, proof by demonstration, the empirical method, proof using mathematical means and hermeneutics. The appropriateness of each of these methodologies will be considered in turn in section 4.2.1 to section 4.2.4.

4.2.1. Proof by demonstration

The proof by demonstration method allows for an initial prototype to be developed and then refined many times until the objectives have been met. In the case of ETP it may not be possible to develop a novel approach that works well initially especially if the problem domain has different versions. Also, problem instances in the same problem set may have different characteristics and complexities. To achieve the objectives it would be necessary to first develop a prototype and repeatedly test and refine the approach until a suitable level of performance is achieved. The proof by demonstration method is considered a suitable method because it allows the refinement of an initial prototype until the objectives has been met.

4.2.2. Empirical method

A hypothesis is formulated at the beginning of this method and then it is either accepted or rejected at the end. This method does not allow for use of any artefacts or prototypes. There is no refinement of any initial prototype in this methodology. Therefore, it is not suitable because the study requires that a prototype be developed and iteratively refined. Also, no initial hypothesis is required in this study.

4.2.3. Proof using mathematical means

This method involves the use of proofs and derivations which is not appropriate for this research.

4.2.4. Hermeneutics

Hermeneutics has its roots in social science. The system is deployed in its intended environment and observed. This allows for a more subjective assessment of the proposed system. Hermeneutics requires that humans using the system be observed which is not relevant for this study.

In conclusion, it has been shown that the most suitable approach from all the methodologies described in section 4.1 is the proof by demonstration methodology for this study. This methodology allows for an initial prototype to be built and then refined many times until the objectives have been met. The next section provides the methodology steps to be followed in this research when developing the new approach to solve the ETP.

4.3. Methodology steps

The methodology steps in the study are as follows:

- An initial approach will be developed based on the conclusions drawn from the analysis of the literature. The approach will make use of structure and objective value to guide the search process. The search process will search the solution spaces of partial timetables.

- The approach will be evaluated using the benchmark sets that represent the different versions of the ETP.
- The approach is stochastic in nature and multiple runs will be done to assess the performance and test the statistical significance of the results. For each run a different seed will be used for the random number generator. The performance of the approach will be compared to other leading methods for the same benchmark sets. In the case where the approach has failed to find a feasible solution or produce good results then the reasons for failure or poor performance will be recorded.
- The approach will be revised based on the recorded reasons for poor performance.
- The revised approach will be re-evaluated using the same benchmark sets and any reasons for poor performance or failure will be recorded again. Statistical tests will be done to ascertain the significance of the results. This iterative refinement process will continue until no significant improvement in the approach can be achieved or until the objectives have been met.
- In the case where the approach has failed to find suitable solutions to any problem instances then the reasons for the failure will be reported. Reasons for any poor performance will be also reported. This report may be used in future work to improve the performance of the approach.
- The results obtained by the final revised approach will be used to draw conclusions on the performance of the approach in solving the ETP.

4.4. Benchmark Sets

This approach will be tested on three popular benchmarks used to compare and contrast methods. The benchmark sets are the Carter benchmark set, the ITC2007 benchmark set and the Yeditepe benchmark set. All these benchmark sets were described in detail in Chapter 3.

4.5. Technical Specifications

The approach will be developed on a computer desktop running Windows 7 professional (64-bit Operating System). The approach will be developed using Java version 1.6 and the open source Eclipse IDE from IBM. The technical specifications of the desktop are Intel® Core™ i7-6700 CPU @ 3.40 Ghz with 7.88 GB usable RAM. The server cluster at the CHPC (Centre for High Performance Computing) in Cape Town, South Africa will be used to run simulations [103]. The wall time will be restricted to 48 Hours in keeping with CHPC policy.

4.6. Parameter Tuning

All metaheuristics have a set of user-defined parameters that need to be calibrated before such methods can be used to solve problems [74]. Eben and Smit [104] state that there are two distinct classes of parameters, namely, parameter control and parameter tuning. In parameter control the parameter values which are set at the beginning are changed based on a chosen control strategy as the algorithm progresses. Parameter tuning is different in that the parameter values are set at the beginning and remain the same until the algorithm terminates. In this study only parameter tuning is relevant as the parameter values set at the beginning of the proposed approach to solve the ETP will not be changed as the algorithm progresses.

Parameter tuning is not a trivial process [105]. For example, parameters with numeric parameter values may have large parameter spaces that need to be considered to find optimal parameter values. As a result this exercise is also an optimisation problem. Dobslaw [106] states that there is no generic approach that may be used to set optimal parameter values in metaheuristics for a given problem domain. It is also the case that there are no optimal parameter values for problem instances within the same problem domain due to the difference in features and the complexity between the instances. However, taking the approach to perform parameter tuning for every problem instance is both time consuming and computationally expensive [105].

The aim of this work is to find an approach that works well for both the capacitated and uncapacitated versions of the ETP. For this reason parameter tuning will not be done for each problem instance in each benchmark set representing each version of the problem. Instead, an initial set of parameter values will be set up front and kept the same for each problem instance. In this way, the aim of finding an approach that generalises well across all problem sets may be achieved. To find the initial parameter values, the following approach will be used;

- First, all parameters will be set to those typically found in literature.
- Next, trial runs will be performed to assess the impact on performance caused by varying the value of a single parameter whilst keeping the other parameter values the same. Once a suitable parameter value has been found for that parameter which performs satisfactorily, the other parameters will be investigated in turn.
- In the end a suitable set of initial parameter values for the proposed approach will be found.

4.7. Summary

The proof by demonstration methodology will be used to meet the objectives of this research. This methodology allows for an initial approach to be iteratively refined until the objectives have been met or until no further improvement is possible. The output or results from each refinement will be used to inform the refinements made to the approach in the next testing cycle.

Three benchmark sets will be used for the evaluation. The chosen benchmark sets represent both the capacitated and uncapacitated version of the ETP. Technical specifications are given for the computer that will be used to build the approach and revisions. Lastly, the method used for parameter tuning in the proposed approach is outlined. In the next chapter, the proposed Structure Based Partial Solution algorithm is introduced and the approach is presented in detail.

CHAPTER 5: STRUCTURE BASED PARTIAL SOLUTION SEARCH

In this chapter the Structure Based Partial Solution Search (SBPSS) method is introduced and discussed in detail. Section 5.1 provides an overview of the SBPSS. Section 5.2 describes the SBPSS algorithm in detail. Section 5.3 outlines the SBPSS parameters values used. Section 5.4 discusses the application details for the three benchmark sets. Section 5.5 concludes the chapter.

5.1. Overview of the SBPSS

In this subsection the phases of the SBPSS are discussed in section 5.1.1. This is followed by a description of the search process employed by the SBPSS in section 5.1.2.

5.1.1. SBPSS Phases

The SBPSS is made up of two phases, namely, a construction phase and a deconstruction phase as shown in figure 5.1.

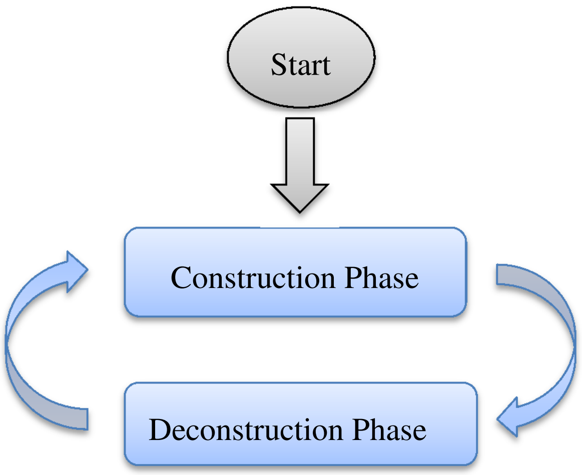

Figure 5.1 SBPSS Phases

The algorithm begins with the construction phase. In this phase multiple timetables are constructed simultaneously by scheduling exams incrementally. A construction heuristic is used to determine the order in which the exams are scheduled. This phase terminates when all exams have been scheduled and completed timetables have been produced. The algorithm then enters a deconstruction phase. In this phase randomly chosen examinations are removed from the completed timetables to produce partially completed timetables. The algorithm then re-enters the construction phase where those examinations that have been removed are rescheduled to produce new completed timetables. The algorithm cycles between the construction and deconstruction phase until the stopping condition has been met. This recycling process enables the search to explore more of the solution space in an attempt to find better performing regions.

5.1.2. SBPSS Search Process

In section 2.4.2 in chapter 2, it was stated that multi-point search methods have good diversification capabilities because these methods sample more of the search space at the same time. The SBPSS inherits this trait as it is a multi-point search method which constructs multiple timetables at the same time. The critical analysis in chapter 3 (section 3.4) argued that a search that is structure-based and one that works in the partial solution space should be considered. The SBPSS makes use of a structure-based search and searches partial solution spaces whilst constructing timetables. The behaviour of the search process in the SBPSS is determined by both the construction phase and deconstruction phase which is discussed in sections 5.1.2.1 and section 5.1.2.2 respectively.

5.1.2.1. <u>Search Performed in the Construction Phase</u>

The aim of the construction phase is to construct complete timetables. Therefore, multiple iterations of the search process shown in figure 5.2 take place in this phase until construction of all timetables have been completed. Each step in the search process is discussed in turn below.

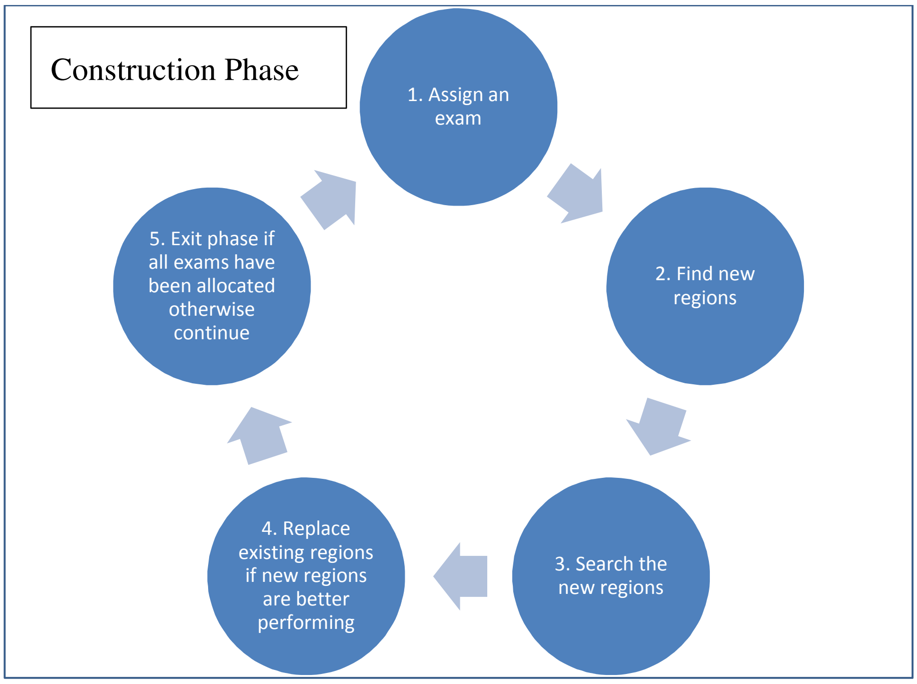
Construction Phase
1. Assign an exam
2. Find new regions
3. Search the new regions
4. Replace existing regions if new regions are better performing
5. Exit phase if all exams have been allocated otherwise continue

Figure 5.2: Search Process in Construction Phase

- Step 1 – Assign an exam

Every iteration the process is started by assigning an exam to the partial timetable under construction. A construction heuristic is employed to determine the order in which exams are scheduled.

- Step 2 – Find new regions

After the assigning of exams to the partial timetables, the search explores the partial solution space to find new regions. It makes use of the structure of the timetables to achieve this objective by delineating points in the partial solution space into regions based on structure. Points that are similar in structure occupy the same region in the solution space. The region is then defined by the common structural components among the points in that region. Similarly, in examination timetabling, the partial timetables which represent points in the solution space are organised into regions with the ones that are similar belonging to the same region. Two or more timetables may be considered similar if the number of common structural components among them is greater than a similarity threshold value which is a parameter value for the algorithm. This is computed by making use of equation 5.1 below where p_i and p_j are partial timetable solutions under inspection. If the computed value is above the threshold value then both solutions are considered to be similar.

$$Simlarity\ Threshold = \frac{\sum_{i=1}^{m} c_{ij}}{m} * 100 > simThresh \qquad (5.1)$$

Where: m is the number of components in p_i and $c_{ij} = 1$ if c_{ij} is a solution component in both p_i and p_j. Example 5.1 provides an illustration of this process. Table 5.1 shows two timetables with five periods each and ten exams. The similarity threshold in this example is 5%.

Table 5.1 Similarity Index Example

	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
Timetable$_1$	0	1	1	2	0	2	4	2	3	4
Timetable$_2$	0	1	0	2	1	4	4	2	3	4

The table shows that in timetable$_1$; exam e_0 is assigned to period 0 and exam e_1 assigned to period 1 and so forth. Using equation 5.1 the value of m (number of assignments in both timetables) is ten because ten exams have been scheduled in both timetables. Also in seven of the ten assignments, the same exams have been scheduled to be taken in the same periods. Therefore, the number of common assignments is seven and the computed similarity index is 70%. The two timetables are then considered to be similar in structure because their similarity index is above the 5% threshold. Finally, the region occupied by both timetables is defined by the exams that have the same scheduling. The size of the timetable does not have an impact on the similarity index or influence its value because it is expressed as a ratio of the number of common components for the timetables being compared which is *m* in equation 5.1 shown above

The SBPSS continues in this fashion to organise all partial timetables under construction into regions. It may be the case that the similarity index for some timetables may not be above the required threshold. This may result in regions that have only one timetable. The end result is a set of one or more new regions where each region consists of one or more partial timetables. Each region may be defined by the common exam assignments among the timetables in that region. For example, figure 5.3 shows ten timetables represented by points in the solution space S. In this example three regions were found. *Region$_1$* has only one point because that timetable does not share any common structural components with any of the other timetables. *Region$_2$* has five points because the similarity index for all five timetables in this region is above the required threshold. The same argument is made for *region$_3$* which has four points.

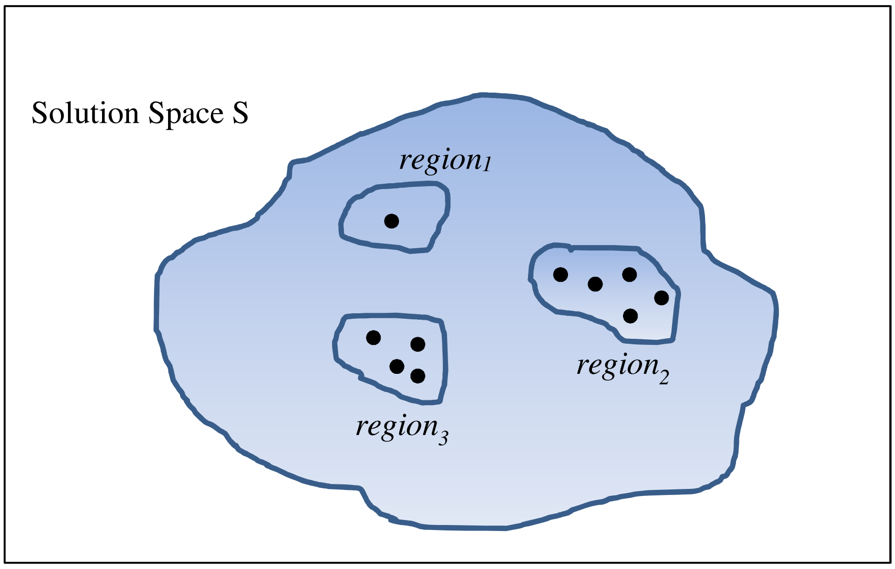

Figure 5.3: Solution Space S delineated into three regions

- Step 3 – Search the new regions.

After a set of new regions has been found the SBPSS proceeds to search within each of these newly found regions. The SBPSS makes use of a number of hill-climbers. The aim of these methods is to investigate and exploit the new regions to find better quality partial timetables. The SBPSS makes use of two distinct types of search methods. The first type searches the same partial timetable in an attempt to improve the quality. The second type exchanges information between partial timetables in the same region in order to improve quality of one or more timetables. These search methods are problem dependant and are discussed later as part of the implementation details in section 5.4.1.3.

- Step 4 – Replace existing regions if new regions are better performing.

After the new regions have been searched, the SBPSS evaluates them to determine if the same regions should be retained. If the search was successful in improving the quality of some of the timetables in some of the regions then the number of common solution components among the timetables in those changed regions would increase. Also, these regions will be considered to be better performing because they have better quality timetables. If this is the case then the new regions replace the existing set of regions before entering a new iteration in the construction phase. The evaluation is done by counting the number of common exam assignments (structural solution components) among the partial timetables in each region (similarity index of the region). Thereafter, the average similarity index value for the whole set of regions is calculated. If this average commonality value is greater than that of the current set of regions then the new set of regions replaces the current set. This means that the new set of regions are better performing because there is a greater number of common exam assignments in each region because of the improved quality of partial timetables in them.

Example 5.2 provides an illustration of this process. Figure 5.4 shows the current set of regions and the new regions found by the search in solution space S. The current set has three regions with two timetables found in *region$_1$*, four timetables found in *region$_2$* and so forth. Similarly the new set has two regions with six timetables found in *region$_1$* and four timetables found in *region$_2$*. Table 5.2 shows the same two sets of regions. The similarity index for each region is given and is computed using equation 5.1. This value represents the number of common exam assignments among the timetables in that region as described earlier in example 5.1 using table 5.1. The average similarity index is also given. This value is calculated by summing the similarity index for each region and then dividing it by the number of regions being counted. The results in table 5.2 show that the average similarity index for the current set and the new set is 7% and 5.5% respectively. Therefore, the SBPSS retains the current set of regions and discards the new set of regions.

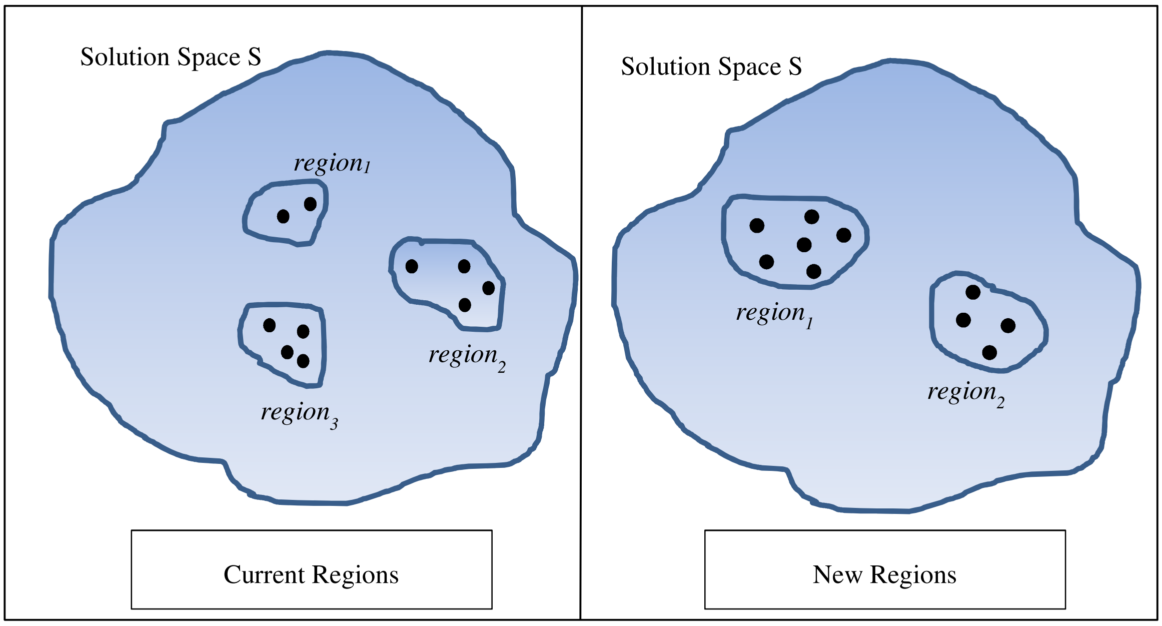

Figure 5.4 Solution Space S with New Regions and Old Regions

Table 5.2 Average Similarity Index for Regions

Current Set		**New Set**	
Region	**Similarity Index**	**Region**	**Similarity Index**
Region1	7%	Region1	6%
Region2	8%	Region2	5%
Region3	6%		
Average Similarity Index	7%	Average Similarity Index	5.5%

- Step 5 – Exit the phase if all the exams have been allocated, otherwise continue.

After a decision has been made on whether to keep the current set of regions or replace the current set of regions with the new set of regions, the SBPSS exits the construction phase if all the exams have been assigned and timetable construction has been completed. Otherwise, a new search cycle is performed in this phase.

5.1.2.2. <u>Search Performed in the Deconstruction Phase</u>

No search takes place in the deconstruction phase. However, this phase enables the search to be restarted at a different location in the partial solution space. By removing random examinations from a completed timetable, a new partial timetable is produced. In the next construction cycle the search uses this new partial timetable as a new starting point in the partial solution space. In chapter 2 it was stated that performance may be improved in two ways;

- Allow the search to return to areas already found to be promising for further investigation because these areas are likely to have better solutions and
- Allow the search to find new areas to search.

To ensure that the SBPSS is able to achieve both these objectives two input parameter values are used to determine the number of examinations to remove from the completed timetables in this phase. The two parameter values used are *unschePsmall* and *unschePlarge*. These parameter values are defined later in section 5.3 and are problem dependant. The smaller of the two *unschePsmall* allows for a smaller number of examinations to be removed. As a result, the search is restarted close to good solutions already found. However, it may be the case that after a limited number of attempts the search may not be able to find better solutions in the same area. In this case the search is moved to a further location by removing a larger number of examinations from the completed solutions using the larger *unschePlarge* parameter value. In this way, the search is more exploratory because it is able to return to good areas to find better solutions or move to new unvisited areas to search.

5.2. The SBPSS algorithm

In this section the SBPSS algorithm is presented and discussed in detail. The SBPSS algorithm is shown in figure 5.5.

The parameter values required by the SBPSS algorithm shown in figure 5.5 are simThresh, unschePsmall and *unschePlarge*. These parameter values are defined in section 5.3. Section 5.1.2.1 also explains the use of *simThresh* by way of an example. Section 5.1.2.2 explains the need for *unschePsmall* and *unschePlarge*. The SBPSS algorithm begins by initializing all the partial solutions (line 2). Each solution is initialized by selecting a solution component at random from the list of unscheduled solution components and assigning it to the empty solution. Thereafter, the partial solution space is delineated into an initial set of regions *curRegs* using the *find_regions* operator (line 3). The delineation process is discussed in detail in section 5.1.2.1. The *find_regions* operator is shown in figure 5.6 and requires the set of partial solutions *nParSolns* to delineate and a similarity index threshold *simThresh* to determine structural similarity. The operator iterates through each partial solution and organises them into regions based on similarity in structure using equation 5.1 (refer to example 5.1 in section 5.1.2.1 for an illustration of this process).

After the initial set of regions has been identified, the SBPSS then proceeds to construct *nParSolns* solutions by repeating the steps from line 5 to line 12 until construction is completed. First a solution component is added to each of the partial solutions in *parSolns* to produce a set of new partial solutions *newParSolns* (line 6). The set of *newParSolns* are then delineated into a set of new regions *newRegs* using the *find_regions* operator again (line 8). The SBPSS then uses a set of search operators (discussed in section 5.4.1.4) to search each region in turn (line 9).

Thereafter, the new regions *newRegs* are evaluated (line 10). The evaluation involves counting the number of common solution components among each partial solution in a region. Thereafter, the average for the whole set of regions is calculated. If the average commonality value is greater than that of the current set of regions then the new set of regions replaces the current set (refer to example 5.2 in section 5.1.2.1 for an illustration of this process). It may be the case that the new set of regions does not perform as well at the current set. To ensure that sufficient exploration of the search space takes place it is necessary for the SBPSS to make numerous attempts to find better performing regions (lines 7-11). The amount of exploration is controlled by a parameter value input to the algorithm, namely, *exploitIts*. This parameter is set up front and determines the amount of exploration that takes place.

```
1: Procedure SBPSS (simThresh, unschePsmall, unschePlarge)
2:   Initialize parSolns partial solution points
3:   Delineate parSolns into regions curRegs = findRegions (simThresh; parSolns)
4:   While maxExplorationHours has not been reached Do
5:     While parSolns partial solution points are incomplete Do
6:         Add a new solution component to each partial solution in parSolns to produce newParSolns
7:       For Its ← 1, exloitIts Do
8:          Delineate newParSolns into regions newRegs = findRegions (simThresh; newParSolns)
9:          Perform search in each region in newRegs
10:         Evaluate newRegs and replace curRegs with newRegs if better
11:      End For
12:    End While
13:    The current_best performing solution becomes the best_so_far solution if it is better
14:    If the best_so_far is unchanged after nAttempts attempts Then
15:       remove unschePlarge solution components in each completed candidate solution
16:     Else
17:       remove unschePsmall solution components in each completed candidate solution.
18:     End If
19:  End While
20: End Procedure
```

Figure 5.5 Structure Based Solution Search Algorithm

```
1: Procedure FINDREGIONS( simThresh, nParSolns)
2:  Add the first partial solution to $reg_0$
3:  Add $reg_0$ to list of regions
4:  While nParSolns is not empty Do
5:    Remove parSoln from nParSolns
6:      For $i \leftarrow 1, n$ Do
7:          If parSoln is similar to solutions in $reg_i$ Then
8:                  Add parSoln to region $reg_i$
9:          End If
10:     End For
11:     If parSoln is not added to a region Then
12:       Create a new region $reg_j$
13:       Add parSoln to $reg_j$
14:     End If
15: End While
16: End Procedure
```

Figure 5.6 FindRegions Algorithm

After the construction phase has been completed the SBPSS finds the current best performing solution in terms of the objective value in the current cycle and replaces the best performing solution found so far with it if it is better (line 13). The deconstruction phase follows immediately after the construction phase (lines 14-18). In this phase solution components are removed from each completed solution. The solution components that are removed from one solution differ to those of another solution due to the random nature of the process. Two different parameter values which are supplied upfront as input values to the algorithm are used to determine the number of solution components to remove from the solutions, namely, $unschePsmall$ and $unschePlarge$. The reasons for using these two parameter values are given in section 5.1.2.2. The smaller of the two $unschePsmall$ is used to determine the number of solution components to remove during each deconstruction cycle. However, if the best solution found so far remains unchanged after a number of attempts then a larger number of solutions components are removed using the $unschePlarge$ value. The number of attempts is a parameter value to the algorithm.

After the deconstruction stage the SBPSS re-enters the construction phase to reconstruct the deconstructed partial solutions (refer to section 5.1.1). The SBPSS continues in this fashion interchanging between the construction and deconstruction phases until the maximum run time allowed as determined by the *maxExplorationHours* parameter value has been reached. This terminates the algorithm. As explained in section 5.1.1 the interchanging between these two phases enables the search to return to areas previously found to be promising or to find new regions for investigation.

5.3. SBPSS parameters

The user defined parameters for the SBPSS are defined in table 5.2:

Table 5.3 SPBSS Parameters

SBPSS parameter	Description
simThresh	Integer percentage value that determines whether partial solutions are similar.
unschedPsmall	Small integer percentage value that determines the number of solution components to be removed during the deconstruction process.
unschedPlarge	Large integer percentage value that determines the number of solution components to be removed during the deconstruction process.
nAttempts	Integer value that specifies the number of operator iterations.
exploretIts	Integer value that specifies the number of exploration iterations.
maxExplorationHours	Integer value that specifies the number of hours the SBPSS should run for before terminating.

5.4. Applying SBPSS to the Examination Timetabling Problem

The SBPSS is applied to the capacitated and uncapacitated versions of the ETP. However, there are details which are common for both versions and these are provided in section 5.4.1. Section 5.4.2 provides the specific application details for the capacitated version of ETP.

Section 5.4.3 provides the specific application details for the uncapacitated version of ETP.

5.4.1. Common application details

The common application details consist of the parameter tuning process, solution initialization process, solution construction heuristic and move operators which are described in section 5.4.1.1, section 5.4.1.2, section 5.4.1.3 and section 5.4.1.4 respectively.

5.4.1.1. Parameter tuning

It has already been stated in section 4.6 in chapter 4 that the approach should generalise well across all problem instances in both versions of the ETP, namely, the capacitated and the uncapacitated version. Section 4.6 also outlines the process followed for parameter tuning in this work. Basically it involves tuning each parameter value whilst keeping the rest of the parameter values the same until all parameter values have been tuned.

Table 5.4 shows the parameter values used in this study for all three benchmark sets tested, namely, ITC2007 benchmark set, Carter benchmark set and Yeditepe benchmark set. For the problem at hand 100 timetables were found to be a suitable representation of the search space. Using a larger number of timetables resulted in minimal improvement but increased runtimes considerably. This was most noticeable for the problem instances that have larger number of students and exams. The value for *simThresh* was set at 5%. Using smaller values for this parameter resulted in a fewer regions being found by the *FindRegions* operator with each region consisting of a larger number of timetables. For example, the *set12* problem instance in the ITC2007 benchmark set has only 78 exams and using a *simThresh* of 1% resulted in 2 or less regions being found as the algorithm progressed. This is not ideal as more regions need to be found to improve the diversification capabilities of the search. Regions that are too large restrict the intensification capabilities of the search because more points in the region under investigation needs to be considered. The value for *unschePsmall* was set at 5%. Using a smaller value than this was not practical for the smaller problem instances. For example, in the case of the *set12* problem instance, making use of a value of 1% results in no exams being removed from the completed timetables in the deconstruction phase. As a result, the algorithm is halted prematurely. Using large values resulted in large jumps being made in the solution space for the larger problem instances. Making large jumps on every iteration reduced the effectiveness of the search. The value of *unschePlarge* was set at 20%. Using smaller values did not allow the search to effectively move beyond a local optimum for the smaller problem instances. Using a larger value caused the performance of the search to deteriorate in most cases as the search was moved away from promising regions in the search space. The values for *exploitIts* and *nAttempts* were both set at 50. These values were informed by the literature; in particular the study by Rajah and Pillay [3] which showed that satisfactory results may be obtained with these parameter values when the Developmental Approach was tested on the Carter benchmark set. Lastly, the number of runs used was thirty to allow statistical testing.

Table 5.4 SBPSS Parameter Values

Parameter	**Value**
$exploitIts$	50
$unschePsmall$	5%
$simThresh$	5%
$unschePlarge$	20%
$nAttempts$	50
$runs$	30

5.4.1.2. <u>Solution Initialization</u>

For all three benchmark sets, the solution timetables are initialized in the same way. An exam is randomly selected from the list of unscheduled exams. The exam is removed from the list and is assigned to a feasible timeslot. A timeslot is considered feasible if it does not violate any hard constraints to the problem at hand. A timeslot consists of just a period for the uncapacitated problem set whilst for the capacitated version it consists of a period and a room. Therefore in the uncapacitated version only a feasible period is required. However, for the capacitated version a feasible timeslot is one where a feasible period is found and that period has a feasible room as well.

5.4.1.3. Solution construction heuristic

The timetables are constructed using a construction heuristic, namely, the saturation degree, to determine the order in which the exams are scheduled. This heuristic works as follows; each unscheduled exam is assigned a saturation degree score which represents the number of feasible periods that it may be assigned to in the current timetable. The exam with the least saturation degree is then scheduled first because it is the most constrained from the other unscheduled exams. If there is more than one feasible period then the exam is scheduled in the period with the least cost. After an exam has been scheduled the saturation degrees of the remaining unscheduled exams are updated. It may be the case that there is more than one exam with the same saturation degree. In this study, the exam with the most students is used to break the tie and is scheduled next. If there are remaining exams to be scheduled and there are no feasible periods available then the exams are scheduled in periods chosen at random.

5.4.1.4. Move operators

All move operators employed by the SBPSS are hill-climbers because only moves that improve the solution quality are accepted. The aim of the operators is to exploit the region of interest in order to find better quality solutions. All move operators make use of the Kempe chain heuristic to resolve exam conflicts and maintain feasibility [14]. The Kempe chain heuristic moves exams between timeslots that are in conflict.

Figure 5.7 shows two periods. Period A has four exams, namely, e_1, e_2, e_3 and e_4. Period B has two exams, namely, e_5 and e_6. The exam conflict matrix is shown in Table 5.5. The conflict matrix shows that exams e_1 and e_5 have common students and should not be scheduled together.

Also exams e_2 and e_5 have common students. The hard constraint for this illustration is that exams that share students should not be scheduled in the same period.

Table 5.5 Example Conflict Matrix

	e_1	e_2	e_3	e_4	e_5	e_6
e_1	0	0	0	0	1	0
e_2	0	0	0	0	1	0
e_3	0	0	0	0	0	0
e_4	0	0	0	0	0	0
e_5	1	1	0	0	0	0
e_6	0	0	0	0	0	0

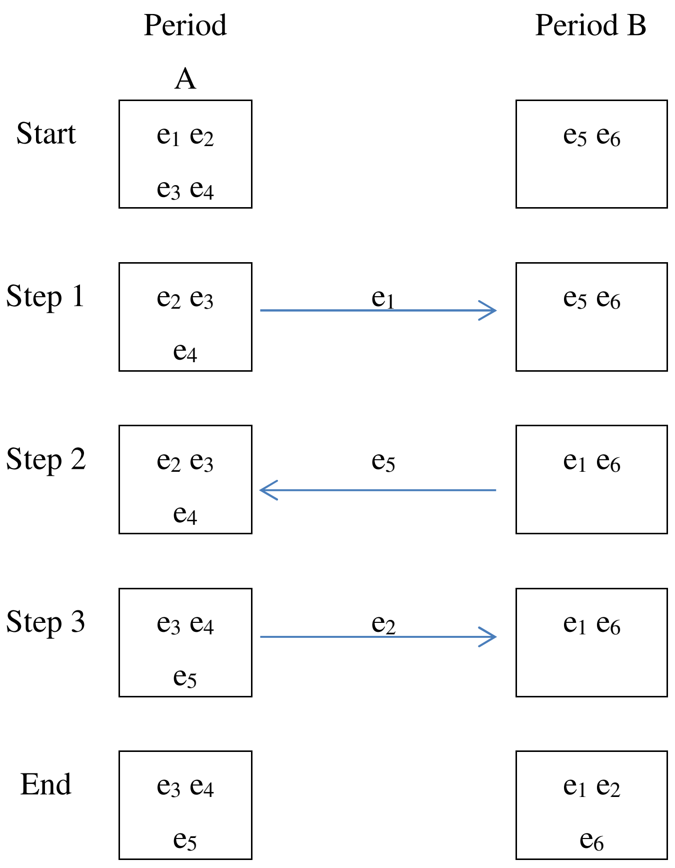

Figure 5.7 Kempe Chain Heuristic Example

The aim in this example is to move e_1 from period A to period B. Step 1 shows e_1 being moved to period B. This move causes e_5 to be moved out of period B to period A as shown in step 2 because e_5 and e_1 are in conflict with each other and having them both in the same period would violate the hard constraint. In step 3 e_2 is moved out of period A to period B to maintain feasibility because it is in conflict with e_5. After the process has stopped exams e_3, e_4 and e_5 are left in period A. Exams e_1, e_2 and e_6 are left in period B.

The move operators are MovePeriodSame, MovePeriodRandom, SwapPeriodRandom, 2WaySwapPeriodRandom, PeriodChange, MoveRoomSame and MoveRoomRandom. These operators are presented in the section 5.4.1.4.1 to section 5.4.1.4.7 respectively.

5.4.1.4.1. <u>MovePeriodSame</u>

This operator uses information from one timetable to make informed decisions on move operations on another timetable. In this way solutions share knowledge about the solution space. Figure 5.8 shows two timetables *timetable$_i$* and *timetable$_j$* before and after the move is performed. Both timetables are randomly selected in a region. Both timetables have ten exams and five periods. In *timetable$_i$* e_0 is assigned to period 0, e_1 is assigned to period 1 and so forth. Before the move e_3 in *timetable$_i$* is assigned to period 2. However, the same exam is assigned to period 3 in *timetable$_j$*. This information is used by the operator to move e_3 in *timetable$_i$* to period 3 to match the assignment in *timetable$_j$*. The move is only performed if it results in a reduction in the hard or soft constraint costs.

		e_0	**e_1**	**e_2**	**e_3**	**e_4**	**e_5**	**e_6**	**e_7**	**e_8**	**e_9**
Before	***timetable$_i$***	0	1	1	2	0	2	4	2	3	4
	timetable$_j$	0	3	0	3	1	4	4	2	3	4

After

timetable$_i$	0	1	1	3	0	2	4	2	3	4
timetable$_j$	0	3	0	3	1	4	4	2	3	4

Figure 5.8 MovePeriodSame Example

5.4.1.4.2. <u>MovePeriodRandom</u>

This operator is different from the *MovePeriodSame* operator in that it involves only one timetable. The aim of this operator is to perform local search on the same timetable.

Figure 5.9 shows two randomly selected periods in a randomly selected timetable before and after the move is performed. An exam is randomly selected in period A and it is moved to period B. In this example, exam e_1 which is assigned to period A is chosen and moved to period B. The move is only performed if it results in a reduction in the hard or soft constraint costs.

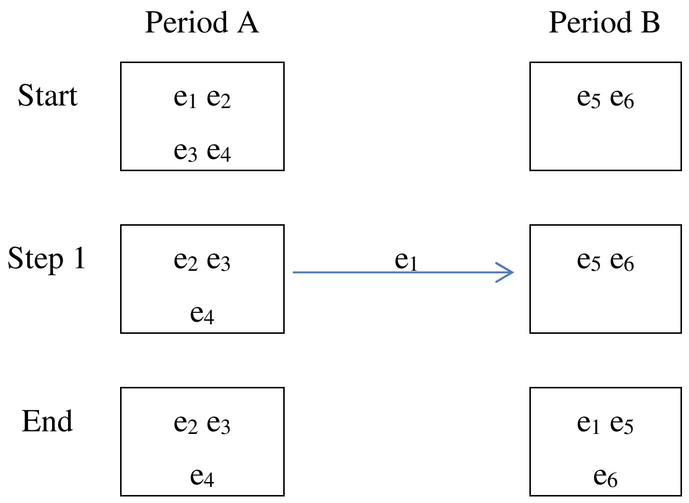

Figure 5.9 MovePeriodRandom Example

5.4.1.4.3. *SwapPeriodRandom*

This move operator is different from the *MovePeriodRandom* operator in that it involves two periods in the same timetable. The aim of this operator is to perform local search on the same timetable by swapping exams between periods.

Figure 5.10 shows two randomly selected periods in a randomly selected timetable before and after the move is performed. An exam is randomly selected in period A and another exam in randomly selected in period B. The selected exams are then swapped between the periods. In this example, exam e_1 and exam e_5 are swapped between the periods. After the move is completed exams e_2, e_3, e_3 and e_5 are left in period A and exams e_1 and e_6 are left in period B. The move is only performed if it results in a reduction in the hard or soft constraint costs.

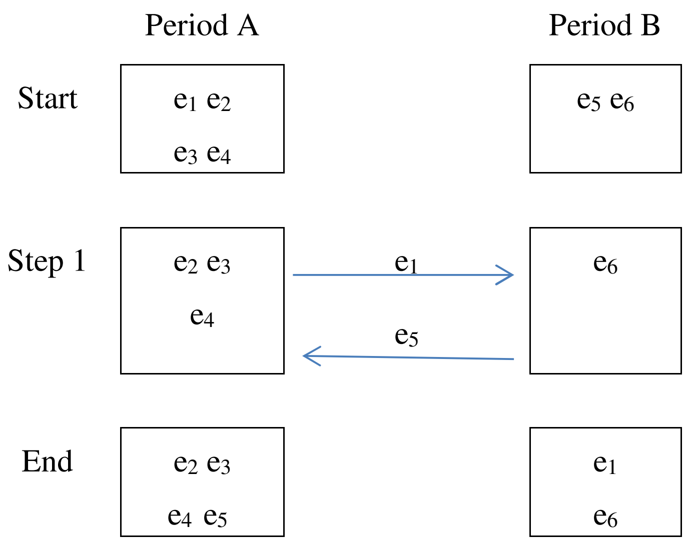

Figure 5.10 SwapPeriodRandom Example

5.4.1.4.4. *2WaySwapPeriodRandom*

This move operator is similar to the previous operator but instead of two periods being compared there are three periods in the comparison. The aim of this operator is then to search a wider area in the same timetable. The cost of swapping an exam between period 1 and period 2 is compared with swapping the exam with period 3. The swap that has the most savings in terms of the soft constraint cost and the hard constraint costs is then performed. The Kempe chain heuristic is used to resolve any potential exams conflicts that may arise as a result of the move. The move is only performed if it results in a reduction in the hard or soft constraint costs.

5.4.1.4.5. PeriodChange

Rather than swapping just a pair of exams between periods, this operator swaps all the exams between periods. In this way it is more exploratory then the other two swap operators, namely, *SwapPeriodRandom* and *2WaySwapPeriodRandom.* A timetable is randomly selected in a region and then two periods are randomly chosen in the same timetable. All the exams in one period are swapped with those in the other period. The move is only carried out if there is a decrease in the soft constraint cost of the timetable and the move does not violate any hard constraints. In this example, in figure 5.11 period A starts off with exams e_1, e_2, e_3 and e_4. Period B has exams e_5 and e_6. After the move is completed exams e_5 and e_6 are left in period A. Exams e_1, e_2, e_3 and e_4 are left in Period B.

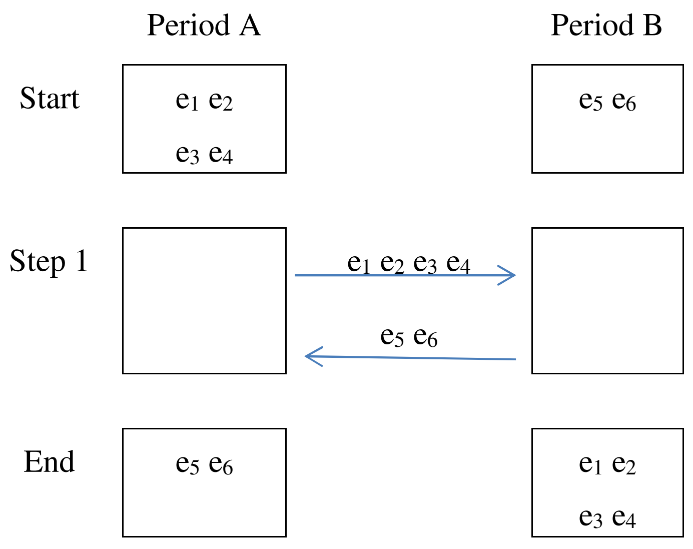

Figure 5.11 PeriodChange Example

5.4.1.4.6. MoveRoomSame

The previous move operators involved period assignments whereas this operator and the next involve room assignments. The aim of this operator is to search the room spaces in periods. Similar to the *MovePeriodSame* operator, this operator uses the information from one timetable to make an informed move in another timetable to further exploit the search space. An exam from *timetable*$_i$ is randomly selected and is moved to a new room. The destination room is the same one that the same exam is assigned to in *timetable*$_j$. Figure 5.12 shows timetables *timetable*$_i$ and *timetable*$_j$ that have been randomly selected in a region. Exam e_0 is assigned to period 0 and room 0 in *timetable*$_i$. It is assigned to period 1 and room 3 in *timetable*$_j$. Exam e_0 is then moved to room 3 in period 0 in *timetable*$_i$ to match the room it is assigned to in *timetable*$_j$. The move is only performed if there is a reduction in hard and soft constraint costs.

		e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
Before	***timetable***$_i$	[0,0]	[0,1]	[0,1]	[0,2]	[0,0]	[0,2]	[0,2]	[0,2]	[0,3]	[0,0]
	timetable$_j$	[1,3]	[1,0]	[1,0]	[1,3]	[1,1]	[1,2]	[1,1]	[1,2]	[1,3]	[1,3]
After	***timetable***$_i$	[0,3]	[0,1]	[0,1]	[0,2]	[0,0]	[0,2]	[0,2]	[0,2]	[0,3]	[0,0]
	timetable$_j$	[1,3]	[1,0]	[1,0]	[1,3]	[1,1]	[1,2]	[1,1]	[1,2]	[1,3]	[1,3]

Figure 5.12 MoveRoomSame Example

5.4.1.4.7. MoveRoomRandom

This operator searches the room spaces in the same period in the same timetable and is different from the *MoveRoomSame* operator in that it involves only one timetable.

Figure 5.13 shows two randomly selected rooms in a randomly selected timetable before and after the move is performed. Next an exam is randomly selected in room A and it is moved to room B. In this example exam e_1 which is assigned to room A is chosen and moved to room B. The move is only performed if it results in a reduction in the hard or soft constraint costs. After the move is completed exams e_2, e_3 and e_4 are left in room A. Exams e_1, e_5 and e_6 are left in room B.

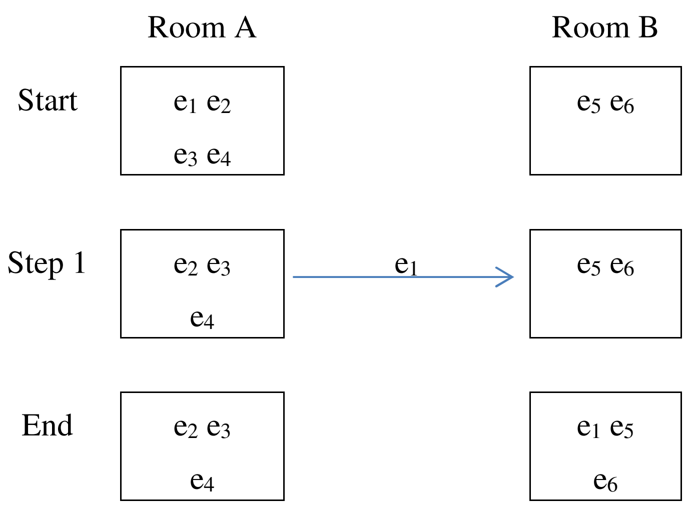

Figure 5.13 MoveRoomRandom Example

5.4.2. Applying the SBPSS to the Capacitated Version of the ETP

The SBPSS was applied to both the ITC2007 benchmark set and Yeditepe benchmark set which represent this version of the ETP. The hard and soft constraints for both these benchmark sets are provided in Chapter3. As stated in section 3.3, in this version of the ETP the capacity of the rooms must be considered. Exams can only be written in rooms where the capacity of the room equals or exceeds the total number students for those exams. Some exams require exclusive use of a room as a hard constraint. At the start of the algorithm the timetables are initialized as described in section 5.4.1.2 using the parameter values listed in section 5.4.1.1. The saturation degree is used to construct the solutions as described in section 5.4.1.3. All move operators listed in section 5.4.1.4 were implemented for both these benchmark sets.

5.4.3. Applying the SBPSS to the Uncapacitated Version of the ETP

The SBPSS was applied to the Carter benchmark set which represents the uncapacitated version of the ETP. The problem constraints are given in section 3.2.1 in chapter 3. Timetables were initialized as describe in section 5.4.1.2 and constructed as described in section 5.4.1.3 using the parameter values listed in section 5.4.1.1. The move operators involving rooms were not implemented because there are no room constraints for this version. Therefore, only the move operators dealing with periods, namely, *MovePeriodSame*, *MovePeriodRandom*, *PeriodChange*, *SwapPeriodRandom* and *2WaySwapPeriodRandom* were implemented.

5.5. Summary

This chapter first provided an overview of the SBPSS by introducing the construction and deconstruction phases. The search process for each of the phases was given in detail. Search takes place during construction using a search that combines structure with the objective value. Search is done on every iteration. The SBPSS algorithm was presented and discussed in detail. The saturation degree was used as the solution construction heuristic. The parameter tuning process and parameter values used in this study is also given. The SBPSS is applied to both the capacitated and uncapacitated versions of the ETP. Hill-climbers are applied for each version to improve the quality of the timetables under construction. The room operators were not required for the uncapacitated version of the ETP. The results of the implementation are given in the next chapter.

CHAPTER 6: RESULTS AND DISCUSSION

In this chapter the results of the SBPSS is presented and discussed. The key design properties are assessed and the results are presented in section 6.1. The results obtained by the SBPSS when applied to the capacitated version of the Examination Timetabling Problem (ETP) are given in section 6.2. This is followed by the results for the uncapacitated version in section 6.3. Section 6.4 concludes the chapter.

6.1. Assessment of SBPSS Design Properties

The SBPSS has two key design features, namely, it searches the space of partial timetables and has a search that is guided by structure. Section 6.1.1 assesses the effectiveness of working in the partial solution space. Section 6.1.2 assesses the effectiveness of employing a search that is guided by structure.

6.1.1. Effectiveness of search in partial solution space

An investigation was undertaken to test the impact on performance when working in the partial solution space for the ETP. The investigation involved comparing the difference in performance between performing search or no search during construction. Four problem instances from the Carter benchmark set, namely, hec-s-92, ute-s-92, yor-f-83 and tre-s-92 were used for this investigation. Thirty runs were done to allow for statistically testing. Each run used a different seed for the random number generator. A hundred timetables were constructed in each run as this was considered to suitably represent the solution space (refer to section 5.4.1.1). All runs produced feasible timetables and the results of soft constraint costs are given in Table 6.1. The best, average and variance of the soft constraint cost from all runs performed is shown. The results show clearly that improved results are produced for all problem instances when search is performed during solution construction as opposed to no search during construction.

Table 6.1 Results of No Search and Search during Construction

Instance	No Search			Search during Construction		
	Best	Average	Variance	Best	Average	Variance
hec-s-92	13.027	13.734	0.115	**10.601**	**10.756**	0.007
yor-f-83	42.260	44.515	0.376	**36.227**	**37.053**	0.143
ute-s-92	30.012	30.900	0.232	**25.024**	**25.189**	0.007
tre-s-92	9.761	10.230	0.029	**8.036**	**8.156**	0.004

Statistical tests were done to ascertain the significance of the result that using search during construction improves performance. The null hypothesis (H_0) is that search makes no difference to performance during construction. The alternate hypothesis is that search improves performance. Table 6.2 shows the critical values for each level of significance and the decision rule on whether to accept or reject the null hypothesis. Table 6.3 shows the Z-values for each of the problem instances.

Table 6.2 Levels of Significance and Decision Rules

Levels of significance	Accept/Reject Rules
0.01	Reject H_0 if $Z > 2.33$
0.05	Reject H_0 if $Z > 1.64$
0.10	Reject H_0 if $Z > 1.28$

Table 6.3 Z-Values for each Problem Instance

Instance	Z value
hec-s-92	46.82
yor-f-83	56.74
tre-s-92	62.44
ute-s-92	63.70

The result that using search during solution construction is better than no search is significant at all levels of significance for the four problem instances tested. Therefore, the use of search construction is justified.

The next investigation assessed the impact on performance when search is performed at different intervals during solution construction. The same problem instances from the previous investigation were used. The same number of runs and timetables in each run as in the previous investigation was used. Four scenarios were considered in the investigation:

- Search is performed on every iteration of algorithm.
- Search is performed on every 5th iteration of the algorithm.
- Search is performed on every 10th iteration of the algorithm.
- Search is performed on every 20th iteration of the algorithm.

All the runs for all the problem instances produced feasible timetables. Therefore, only the soft constraint cost was used to compare the results of each scenario. Table 6.4 shows the iteration search results from the study. For each of the scenarios, the best, average and variance of the soft constraint cost from all runs performed is shown. However, using the best produced result to determine the interval at which search should be performed does not allow for conclusive findings. For example, in the case of tres-s-92, in the same table, a better result is obtained when search is performed at every 10th iteration compared to every 5th iteration. This inconsistency may be due to the stochastic nature of the method. Therefore, it is more useful to make use the average results across all runs in order to draw meaningful conclusions. The best average values are in bold. By considering the average results it is clear that the best results are obtained when search is performed on every iteration in the algorithm.

Table 6.4 Iteration Search Results

Instance	Every Iteration			Every 5th Iteration			Every 10th iteration			Every 20th iteration		
	Best	Avg	Var	Best	Avg	Var	Best	Avg	Var	Best	Avg	Var
hec-s-92	**10.601**	10.756	0.007	10.734	10.942	0.015	10.819	11.063	0.014	10.951	11.394	0.092
yor-f-83	**36.227**	37.053	0.143	36.998	37.453	0.081	36.745	37.514	0.121	36.863	38.108	0.325
ute-s-92	**25.024**	25.189	0.007	25.009	25.214	0.009	25.003	25.217	0.015	25.081	25.282	0.008
tre-s-92	**8.036**	8.156	0.004	8.112	8.196	0.002	8.085	8.210	0.003	8.119	8.254	0.003

Figure 6.1 shown below provides a graphical comparison for each of the tested scenarios. The scenario where search is used at every iteration is used as the baseline to compare the average soft constraint cost of each scenario to complete the assessment. This approach allows for a pairwise comparison between competing scenarios. Figure 6.1 illustrates that the frequency at which search is performed affects performance. At the 5th, 10th and 20th iteration there is a positive increase in all the soft constraint costs indicating that the best performance is achieved when search is done at every iteration. Furthermore, in general performance decreases as the interval at which search is performed is decreased for all problem instances.

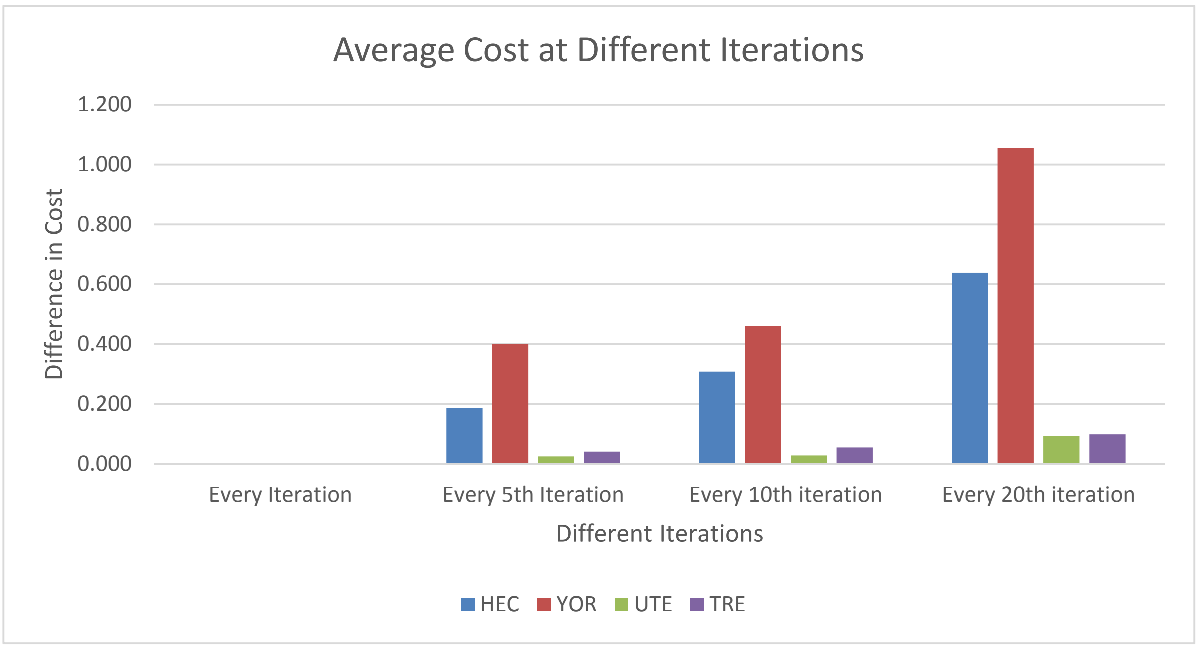

Figure 6.1 Soft Constraint Cost at Different Iterations

6.1.2. Effectiveness of a structure-based search

A separate study was undertaken to assess the effectiveness of using a search combined with structure and objective value instead of a search solely guided by the objective value. The same problem instances and number of runs from the previous study were used. All runs for all the problem instances tested produced feasible timetable solutions. Therefore, only the soft constraint cost was used to compare both approaches to search. The results are shown in Table 6.5. For each case, the best soft constraint cost obtained is shown in the table. This is followed by the average and variance soft constraint costs from all runs. The best results are in bold. The results in the table suggest that using a search combined with structure leads to superior performance. The average scores obtained for all problem instances are better for the combined search. Also, the lower variance scores by the combined search indicate that this approach performs consistently better across all runs.

Table 6.5 Objective Value Based Search versus Combined Search

Instance	Objective Value-Based Search			Combined Search		
	Best	Average	Variance	Best	Average	Variance
hec-s-92	10.7326	10.9952	0.0188	**10.6008**	**10.7561**	0.0068
yor-f-83	36.6142	38.1949	0.2824	**36.2274**	**37.0528**	0.1427
tre-s-92	8.1374	8.2326	0.0024	**8.0365**	**8.1561**	0.0038
ute-s-92	25.0447	25.2609	0.0135	**25.0236**	**25.2138**	0.0075

Statistical tests were done to ascertain the significance of the result that using a combined search is better than using a search guided solely by the objective value. The null hypothesis (H_0) is that both approaches have the same performance. The alternate hypothesis is that a combined search has a better performance. Table 6.2 shows the critical values for each level of significance and the decision rule on whether to accept or reject the null hypothesis. Table 6.6 shows the Z-values for each of the problem instances.

Table 6.6 Z-Values for each Problem Instance

Instance	Z value
hec-s-92	8.19
yor-f-83	9.59
tre-s-92	5.33
ute-s-92	1.78

The result that using search combined with structure is better than a search guided by the objective value alone is significant at all levels of significance for the four problem instances tested. Therefore, the use of search that is combined with structure is justified.

6.2. Results for the Capacacitated Version of the ETP

In this section the results for the benchmark sets representing the capacitated version of the ETP are presented and discussed. Section 6.2.1 presents results for the ITC2007 benchmark set and Section 6.2.2 presents results for the Yeditepe benchmark set.

6.2.1. The ITC2007 Benchmark Set Results

Table 6.7 summarizes the results for the ITC2007 benchmark set using the parameter values stated in chapter 5. The table shows the best soft constraint cost obtained, followed by the average soft constraint cost and the variance for all the runs.

Table 6.7 SBPSS Results for ITC2007 Benchmark Set

Instance	Best	Average	Variance	Average Runtime (seconds)
1	3925	4208.80	19501.06	43200
2	376	478.43	2115.56	43200
3	8245	8924.40	133525.77	43200
4	12493	13050.67	100582.92	43200
5	2659	2761.80	2908.51	86400
6	25205	25903.50	115897.91	43200
7	3901	4429.23	91177.70	86400
8	6756	8146.10	556412.16	86400
9	910	1008.83	3568.01	43200
10	12939	13091.43	5136.19	86400
11	24809	25684.50	292857.84	86400
12	5095	5192.83	2248.14	43200

The runtimes were varied for each problem instance based on the instance size as determined by the number of exams and students for that instance. Each of the runs for each of the problem instances produced feasible timetables. Table 6.8 compares the results of SBPSS to the current state of the art methods found in the literature that have achieved the best performance. The best results are in bold. These methods were discussed in section 3.3.2 in chapter 3.

Table 6.8 Comparison of SBPSS to State of the Art for the ITC2007 Benchmark Set

Instance	**SBPSS**	**Muller**[92]	**Gogos et al.**[107]	**Bykov and Petrovic**[94]	**Leite et al.**[14]	**Leite et al.**[87]
Set1	3925	4370	4128	**3647**	5050	6207
Set2	**376**	400	380	385	395	535
Set3	8245	10049	7769	**7487**	9574	13022
Set4	12493	18141	13103	**11779**	12299	14302
Set5	2659	2988	2513	**2447**	3115	3829
Set6	**25205**	26585	25330	25210	25750	26710
Set7	3901	4213	**3537**	3563	4308	5508
Set8	6756	7742	7087	**6614**	7506	8716
set9	**910**	1030	913	924	977	1030
set10	12939	16682	13053	**12931**	13449	13894
set11	24809	34129	24369	**23784**	30112	39783
set12	**5095**	5535	5095	5097	5148	5142

All the other approaches in the comparison used to improve initial completed solutions. The SBPSS is the only method in this comparison that works in the partial solution space. All of the other methods use a search that is guided solely by the objective function. The SBPSS is different in that it uses a search that combines structure as well. The SBPSS is able to produce the best results for four problem instances, namely, set2, set6, set9 and set12. For the rest of the instances, it produces results comparable to the other leading methods. It performs well for the larger problem instances like set2 and set3 with 870 and 934 exams respectively.

6.2.2. The Yeditepe Benchmark Set Results

Table 6.9 summarizes the results obtained when the SBPSS was applied to the Yeditepe benchmark set. All the runs produced feasible timetables resulting in zero hard constraint costs. The table shows the best soft constraint cost obtained, followed by the average soft constraint cost and the variance from the all the runs. The last column shows the average runtime in seconds.

Table 6.9 SBPSS Results for the Yeditepe Benchmark Set

Instance	Best	Average	Variance	Average Runtime (seconds)
20011	47	49.8	1.2	21600
20012	102	108.68	14.14	21600
20013	29	29	0	3600
20021	47	55.16	15.01	21600
20022	129	151.04	146.52	21600
20023	56	56	0	3600
20031	99	129.76	147.94	21600
20032	359	385.28	162.92	21600

The SBPSS has the same performance for all runs for instances 20013 and 20023 as indicated by the zero variance. It may be the case that these instances cannot be improved upon or the

SBPSS is less effective in solving them. Table 6.10 compares the results of SBPSS to the current state of the art methods found in the literature that have achieved the best performance for this benchmark set. The best results are in bold. These methods were discussed in section 3.3.4 in chapter 3. The SBPSS improves on the best results for all problem instances in this benchmark set. There are not many methods in the literature to compare the results with.

Table 6.10 Comparison of SBPSS to State of the Art for the Yeditepe Benchmark Set

Instance	**SBPSS**	**Muller**[92]	**Muklason et al.**[84]
20011	**47**	62	56
20012	**105**	125	122
20013	**29**	29	29
20021	**55**	70	76
20022	**129**	170	162
20023	**56**	70	56
20031	**169**	223	143
20032	**377**	440	434

The results show that the SBPSS outperforms both best methods it is compared to. All three approaches perform the same for the instance 20013. SBPSS is the only method in the comparison that searches partial solution spaces and makes use of a search that combines structure with the objective value.

6.3. Results for the Uncapacacitated Version of the ETP

In this section the results for the benchmark set representing the uncapacitated version of the ETP are presented and discussed. Section 6.3.1 presents results for the Carter benchmark set.

6.3.1. The Carter Benchmark Set Results

Table 6.11 summarizes the results obtained when the SBPSS was applied to the Carter benchmark set. The table shows the best soft constraint cost obtained, followed by the average soft constraint cost and the variance from the all the runs. The last column gives the runtime in seconds. All the runs produced feasible timetables.

Table 6.11 SBPSS Results for the Carter Benchmark Set

Instance	Best	Average	Variance	Average Runtime (seconds)
hec-s-92	10.032	10.165	0.0048	86400
car-s-91	4.390	4.632	0.0177	172800
car-f-92	3.701	3.880	0.0116	172800
ute-s-92	24.759	24.871	0.0041	86400
tre-s-92	7.619	7.891	0.0125	86400
lse-f-91	9.804	9.993	0.0079	86400
kfu-s-93	12.810	13.003	0.0087	172800
yor-f-83	34.413	35.240	0.0958	86400
uta-s-92	3.035	3.182	0.0061	172800
ear-f-83	32.588	33.120	0.0584	86400
sta-f-83	157.032	157.048	0.0001	86400
rye-s-93	7.849	8.079	0.0081	172800

Table 6.12 compares the results of SBPSS to the current state of the art methods found in the literature that have achieved the best performance for this benchmark set. The best results are in bold. These methods are discussed in section 3.2.2 in chapter 3. The SBPSS has results

comparable to the other leading methods. The SBPSS obtains the best result for two problem instances, namely, tre-s-92 and yor-f-83.

Table 6.12 Comparison of SBPSS to State of the Art for the Carter Benchmark Set

Instance	**SBPSS**	**Caramia et al.**[90]	**Burke et al.**[29]	**Burke and Bykov**[85]	**Mandal and Kahar**[89]	**Leite et al.**[87]	**Bellio et al.**[88]
car-s-91	4.39	6.6	4.6	4.32	4.58	4.31	**4.25**
car-f-92	3.70	6.0	3.9	3.67	3.82	3.68	**3.66**
ear-f-83	32.59	**29.3**	32.8	32.62	33.23	32.48	32.42
hec-s-92	10.03	**9.2**	10.0	10.03	10.32	10.03	10.03
kfu-s-93	12.81	13.8	13.0	**12.80**	13.34	12.81	**12.80**
lse-f-91	9.80	**9.6**	10.0	9.78	10.24	9.78	9.77
rye-s-93	7.85	**6.8**	-	7.91	9.79	7.89	7.9
sta-f-83	157.03	158.2	**156.9**	157.03	157.12	157.03	157.03
tre-s-92	**7.62**	9.4	7.9	7.64	7.84	7.66	7.68
uta-s-92	3.03	3.5	3.2	2.98	3.13	3.01	**2.97**
ute-s-92	24.76	**24.4**	24.8	24.78	25.28	24.80	24.79
yor-f-83	**34.41**	36.2	34.9	34.71	35.46	34.45	34.48

The method put forward by Mandal and Kahar [89] is similar to the SBPSS in that it also employs search during construction. It makes use of the Great Deluge to improve the quality of the partial timetables. The SBPSS makes use of simple hill-climbers to improve solution quality. Despite using the Great Deluge which is more advanced than local search it is unable to outperform the SBPSS. This seems to indicate that using a structure-based search is effective in moving the search to better performing areas. Leite et al. [87] used the Threshold algorithm with a memetic algorithm. The Threshold algorithm only accepts moves that are lower than a threshold. The SBPSS makes use of greedy hill-climbers because it only accepts moves that improve the current solution. Although both methods are different in this respect they are similar in performance especially for the problem instances with high conflict densities like hec-s-92 and ute-s-92.

6.4. Summary

This chapter presented the results of two separate investigations on the design properties of the SBPSS. In the first investigation the impact of using search during solution construction was studied. It was found that performance was improved when search was performed in the partial solution space during solution construction. Furthermore, it was also shown that performing search on every iteration in the algorithm after assigning a new exam maximized performance. The next investigation studied the impact on performance when the search was combined with structure. The results from the investigation showed that using a search that combined structure with the objective value improved performance. Statistical tests were done to ascertain the significance of the results. The results were found to be significant for all levels of significance.

The rest of the chapter presented the results when the SBPSS was used to solve the problems from the benchmark sets representing both the capacitated and uncapacitated versions of the ETP. The SBPSS was able to solve all problem instances for both the ITC2007 benchmark set and the Yeditepe benchmark set. For the ITC2007 benchmark set it obtained the best result for four of the problem instances. For the Yeditepe benchmark set it obtained the best result for all problem instances. The SBPSS also performed well for the Carter benchmark set which represented the capacitated version of the ETP. It found the best result for three of the problem instances. For the other problem instances the SBPSS obtained results comparable to the state of the art methods for all benchmark sets. The SBPSS is different from all the other leading methods in that it is the only approach to combine structure with the objective value to guide the search in the solution space.

CHAPTER 7: CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER RESEARCH

7.1. Introduction

In this chapter the conclusion to the work presented in this thesis is drawn. The aim and objectives of this study were provided in detail in chapter 1. Chapter 2 provided background information on various metaheuristics that have proven popular over the years. Chapter 3 focussed on the Examination Timetabling Problem (ETP) and presented some of the best performing approaches for this problem domain. Chapter 4 outlined the proof by demonstration methodology adopted in this study. Chapter 5 introduced the SBPSS as the proposed method to solve the ETP. Chapter 6 presented the results obtained when the SBPSS was evaluated using the problems from the different benchmark sets.

In section 7.2, the objectives outlined in chapter 1 are revisited. For each stated objective a case is made on how the proposed approach has met that objective. Section 7.3 outlines the contributions to new knowledge by this work. Section 7.4 draws conclusions on the success of this study in meeting the aim and the defined objectives at the beginning in chapter 1. Lastly in section 7.5 the future work for this thesis is presented.

7.2. Revisiting objectives

This subsection focuses on how the objectives for this work have been met. The objectives have been outlined in section 1.3 in the chapter 1. These objectives are relisted here for discussion purposes. A write-up is also provided on how each objective has been met.

- **Objective 1: To demonstrate that the use of structure combined with behaviour in terms of the objective value to guide the search is a worthwhile and promising approach to solving the ETP.**

The literature survey revealed that behaviour in terms of the objective value was a popular approach used to guide the search in the solution space. The problem with this approach is that regions that are similar in behaviour may have different structure. Using behaviour alone may cause the search to pursue poor performing regions resulting in the opportunity to find better solutions being lost. The SBPSS is able to overcome this problem because it makes use of a search that combines structure with behaviour in terms of the objective value to find new regions in the search space. Each timetable represents a point in the search space and timetables that are similar in structure occupy the same region of the search space. The search finds new regions by delineating the timetables into regions using structure. This is achieved by organising timetables that are similar in structure into regions. The search then investigates each of the new found regions to find better quality timetables.

A separate study was undertaken to assess the effectiveness of using a search combined with structure and objective value instead of a search solely guided by the objective value. Four problem instances from the Carter benchmark set, namely, hec-s-92, ute-s-92, yor-f-83 and tre-s-92 were used for this investigation. It was thought that these problem instances provided a fair representation of the benchmark. Hec-s-92 has the highest conflict density whilst ute-s-92 has the least number of periods. Yor-f-83 and tre-s-92 are medium sized instances in terms of the number of examinations. All runs for all the problem instances tested produced feasible timetable solutions. Therefore, only the soft constraint cost was used to compare both approaches to search. The results are shown in Table 6.5. The results in the table indicate that using a search combined with structure leads to superior performance. The average scores obtained for all problem instances were better for the combined search. Also, the lower variance scores by the combined search indicate that this approach performs consistently better across all runs. The result was found to be statistically significant at all levels of significance.

- **Objective 2: To demonstrate that searching partial solution spaces whilst constructing solutions leads to good quality solutions to the ETP.**

An investigation of the effectiveness of searching the partial solution space whilst constructing timetables was undertaken (Refer to section 6.1.1). Four problem instances from the Carter benchmark set, namely, hec-s-92, ute-s-92, yor-f-83 and tre-s-92 were used for this investigation. A baseline was established by performing runs for all four instances without the use of search during construction. Thereafter, search during construction was introduced and the same number of runs were performed. All runs produced feasible timetables and the results of soft constraint costs are given in Table 6.1. The results clearly showed that improved results were produced for all problem instances when search was performed during solution construction as opposed to no search during construction. The results obtained were statistically significant at all levels of significance.

The interval at which search should be performed during construction was also investigated (refer to section 6.1.1). Four different scenarios were compared. First search was performed on every iteration during construction. In the second scenario search was performed on every 5th iteration. In the third scenario search was performed on every 10th iteration. In the last scenario search was performed on every 20th iteration. The results showed that searching on every iteration was overall the best approach. The results obtained were statistically significant at all levels of significance.

The SBPSS follows this strategy; it searches the partial timetable space during construction and has two phases, namely, a construction and a deconstruction phase. In the construction phase complete timetables are constructed incrementally by assigning exams to partial timetables. Search is performed on every iteration after a new exam has been scheduled. Problem dependant search operators are called on each iteration. In the deconstruction phase, new partial timetables are produced by removing random exams from completed timetables. The SBPSS then reconstructs new timetables by rescheduling the exams that have been previously removed in the new partial timetables. No search takes place in the deconstruction phase. However, this phase enables the search to be restarted at a different location in the partial solution space. By removing random examinations from a completed timetable, a new partial timetable is produced. In the next construction cycle the search uses this new partial timetable as a new starting point in the partial solution space.

- **Objective 3: To combine structure-based search together with partial solution space in a structure-based partial solution space to solve the ETP.**

The proposed approach was evaluated using both the capacitated and uncapacitated version of the ETP. The Carter benchmark set which represented the uncapacitated version of the ETP. The ITC2007 benchmark set and the Yeditepe benchmark set represented the capacitated version of the problem. SBPSS successfully solved the problems set for both of the versions. The SBPSS found the best result for two of the problem instances in the Carter benchmark set. It found the best result for four of the problem instances from the ITC2007 benchmark set and found the best results for all of the problem instances for the Yeditepe benchmark set. Overall the performance of the SBPSS was satisfactory and its results compared well with the state of the art for all benchmark sets.

7.3. Contributions to new knowledge

The research conducted presents a novel approach to solving the ETP. It incorporates a multi-point search approach with a structure-based search that operates in the solution spaces of partial timetables. The approach performs well for both the capacitated and uncapacitated version of the ETP.

7.4. Conclusion

The research conducted has met all the objectives that were set out at the beginning in Chapter 1. The use of a structured-based search combined with behaviour in terms of the objective value does improve performance. The searching of solution spaces of partial timetables leads to good quality timetables. The proposed approach is able to successfully solve all the problem instances in all benchmark sets that represent both the capacitated and uncapacitated version of the ETP. It found the best results for two of the problem instances in the Carter benchmark set. It found the best result for four of the instances in the ITC2007 benchmark set and the best result for all the instances in the Yeditepe benchmark set. Finally, it compares favourably to other state of the art methods for all benchmark sets.

7.5. Future Work

The SBPSS has been shown to work well for the ETP. However, some observations have been made with regards to performance as follows:

- All work done in this study made use of manual parameter tuning. Joshi and Basal [108] state that the choice of parameter values impacts the performance of an approach and finding the good parameter values is a difficult task. Huang et al. [109]

note that manual parameter tuning is time-consuming and often biased. Also, one set of parameter values may work well for a problem instance but less effective for another problem instance. This is often referred to as the no free-lunch theorem on optimisation [110]. One solution is to make use of automatic parameter tuning approaches. One approach is use a set of training data to find suitable parameter values to apply to the problem at hand. Bellio et al. [88] used to this approach to find good results for ETP. Another approach is the F-Race method [111] where competing candidate set of parameter values are compared to each other using statistical testing and the poor performing candidate set of parameter values are discarded until a single set of values is found that performs well. In future work it may be useful to consider an automatic parameter tuning approach as a way to to realise an improvement in performance.

- One of the observations made was that due to the stochastic nature of the SBPSS infeasible initial timetables would sometimes be constructed. In some cases the removal of exams from feasible timetables during the deconstruction phase would cause infeasible timetables to be reconstructed in the construction phase. This is not ideal because the existence of infeasible points and regions in the solution space causes the SBPSS to waste valuable resources working in these poor regions. Future work may investigate ways of overcoming this challenge. One solution would be to add a repair mechanism in the construction process to enable the SBPSS to produce only feasible solutions. As a result, the effectiveness of the search may be improved because the search will only take place in feasible regions.
- Another observation made was that as the algorithm progressed structural diversity among the timetables was decreased because the timetables become more similar in structure. This results in a loss of information for the search process thereby

reducing its effectiveness. The SBPSS has no mechanism in place to measure structural diversity in the regions. As a result, resources are wasted by the search spending too much time in regions which have points that are very similar. Future studies may consider ways to overcome this issue. One way would be to use Euclidean distance to measure diversity of points in a region. In this way the search can concentrate on regions that are more diverse.

In an order to improve the generality of the approach, the SBPSS may be applied to other problems. Two directions may be pursued here;

- The SBPSS was only applied to the ETP. This problem falls in the broader category of timetabling problems. Other variations such as the School Timetabling Problem exist. Future studies may consider applying the SBPSS to other timetabling problems to develop the SBPSS as a method that works well for all timetabling problems.

- Another direction would be to consider using SBPSS to solve problems in other domains. For example, the Traveling Salesman Problem is described as the problem of finding the shortest route to take when visiting a set of towns. It is also an optimisation problem and using a structure-based search will be a novel approach to solve this problem. This would develop the SBPSS as a method that also works well across multiple problem domains.

REFERENCES

[1] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, “A survey of search methodologies and automated system development for examination timetabling,” *J. Sched.*, vol. 12, no. 1, pp. 55–89, Feb. 2009.

[2] N. Pillay and W. Banzhaf, “A developmental approach to the uncapacitated examination timetabling problem,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5199 LNCS, Springer Berlin Heidelberg, 2008, pp. 276–285.

[3] C. Rajah and N. Pillay, “A Study of Cell Depletion in the Developmental Approach for the Uncapacitated Examination Timetabling Problem,” in *Annual Conference of the Operations Research Society of South Africa (ORSSA 2013)*, 2013, pp. 102–111.

[4] E. Burke, R. Qu, and A. Soghier, “Adaptive selection of heuristics within a GRASP for exam timetabling problems,” *Proc. Multidiscip. Conf. Sched. Theory Appl. MISTA 2009*, no. August, pp. 409–423, 2009.

[5] C. Blum and A. Roli, “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[6] F. Glover, M. Laguna, F. Glover, and M. Laguna, “Tabu Search Principles,” in *Tabu Search*, Springer {US}, 1997, pp. 125–151.

[7] I. H. Osman and G. Laporte, “Metaheuristics: A bibliography,” *Ann. Oper. Res.*, vol. 63, pp. 513–623, 1996.

[8] A. Lodi, S. Martello, and D. Vigo, "Neighborhood Search Algorithm for the Guillotine Non-Oriented Two-Dimensional Bin Packing Problem," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Springer US, 1999, pp. 125–139.

[9] G. R. Raidl, "A unified view on hybrid metaheuristics," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 4030 LNCS, pp. 1–12.

[10] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "A brief survey on hybrid metaheuristics," in *Bioinspired Optimization Methods and their Applications - Proceedings of the 4th International Conference on Bioinspired Optimization Methods and their Applications, BIOMA 2010*, 2010, pp. 3–16.

[11] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *J. Stat. Phys.*, vol. 34, no. 5–6, pp. 975–986, Mar. 1984.

[12] M. A. S. Elmohamed, P. Coddington, and G. Fox, "A comparison of annealing techniques for academic course scheduling," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1408, pp. 92–112.

[13] D. Kusumawardani, A. Muklason, and V. A. Supoyo, "Examination timetabling automation and optimization using greedy-simulated annealing hyper-heuristics algorithm," in *Proceedings of 2019 International Conference on Information and Communication Technology and Systems, ICTS 2019*, 2019, pp. 164–169.

[14] N. Leite, F. Melício, and A. C. Rosa, “A fast simulated annealing algorithm for the examination timetabling problem,” *Expert Syst. Appl.*, vol. 122, pp. 137–151, May 2019.

[15] G. Dueck, “New optimization heuristics; The great deluge algorithm and the record-to-record travel,” *J. Comput. Phys.*, vol. 104, no. 1, pp. 86–92, 1993.

[16] L. Di Gaspero and A. Schaerf, “Tabu search techniques for EXAMINATION TIMETABLING,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001, vol. 2079 LNCS, pp. 104–117.

[17] H. D. Lawal, I. A. Adeyanju, E. O. Omidiora, O. T. Arulogun, and O. I. Omotosho, “University Examination Timetabling Using Tabu Search,” *Int. J. Sci. Eng. Res.*, vol. 5, no. 10, pp. 785–788, 2014.

[18] G. M. White and B. S. Xie, “Examination timetables and tabu search with longer-term memory,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001, vol. 2079 LNCS, pp. 85–103.

[19] G. Kendall and N. M. Hussin, “A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2005, vol. 3616 LNCS, pp. 270–293.

[20] T. A. Feo and M. G. C. Resende, “Greedy Randomized Adaptive Search Procedures,” *J.*

Glob. Optim., vol. 6, no. 2, pp. 109–133, 1995.

[21] M. G. C. Resende and C. C. Ribeiro, “Greedy randomized adaptive search procedures: Advances and extensions,” in *International Series in Operations Research and Management Science*, vol. 272, Springer New York LLC, 2019, pp. 169–220.

[22] J. P. Hart and A. W. Shogan, “Semi-greedy heuristics: An empirical study,” *Oper. Res. Lett.*, vol. 6, no. 3, pp. 107–114, Jul. 1987.

[23] M. H. Gashti and H. Javanshir, “Greedy Randomized Adaptive Search Procedures for a Single Product Network Design Model,” *IJRRAS*, vol. 6, no. 4, pp. 419–423, 2011.

[24] M. J. F. Souza, N. Maculan, and L. S. Ochi, “A GRASP-Tabu Search Algorithm for Solving School Timetabling Problems,” in *Metaheuristics: Computer Decision-Making. Applied Optimization*, Springer, Boston, MA, 2003, pp. 659–672.

[25] M. Paris and C. C. Ribeiro, “Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment,” *INFORMS J. Comput.*, vol. 12, no. 3, pp. 164–176, 2000.

[26] R. K. Ahuja, J. B. Orlin, and A. Tiwari, “A greedy genetic algorithm for the quadratic assignment problem,” *Comput. Oper. Res.*, vol. 27, no. 10, pp. 917–934, 2000.

[27] W. De, S. Rocha, M. Claudia, S. Boeres, and M. C. Rangel, “A GRASP Algorithm for the University Timetabling Problem,” *Pract. Theory Autom. Timetabling*, pp. 29–31, 2012.

[28] P. Hansen and N. Mladenović, "Variable neighborhood search," in *Handbook of Heuristics*, vol. 1–2, 2018, pp. 759–787.

[29] E. K. Burke, A. J. Eckersley, B. McCollum, S. Petrovic, and R. Qu, "Hybrid variable neighbourhood approaches to university exam timetabling," *Eur. J. Oper. Res.*, vol. 206, no. 1, pp. 46–53, Oct. 2010.

[30] P. Alefragis, C. Gogos, C. Valouxis, and E. Housos, "A multiple metaheuristic variable neighborhood search framework for the Uncapacitated Examination Timetabling Problem," *Pract. Theory Autom. Timetabling*, vol. 1, pp. 13–21, 2021.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems*. 2019.

[32] H. R. Ahmed and J. I. Glasgow, "Swarm Intelligence: Concepts, Models and Applications," *Queen's Univ. Sch. Comput. Tech. Reports*, 2012.

[33] S. L. Yadav and A. Sohal, "Study of the various selection techniques in Genetic Algorithms," *Int. J. Eng. Sci. Math.*, vol. 11, no. 2, pp. 198–204, 2007.

[34] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *Int. J. Mach. Learn. Comput.*, vol. 7, no. 1, pp. 9–12, 2017.

[35] T. P. Hong, H. S. Wang, W. Y. Lin, and W. Y. Lee, "Evolution of appropriate crossover and mutation operators in a genetic process," *Appl. Intell.*, vol. 16, no. 1, pp. 7–17, 2002.

[36] K. Sastry, D. E. Goldberg, and G. Kendall, "Genetic algorithms," in *Search*

Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Second Edition, 2014, pp. 93–118.

[37] N. Pillay and W. Banzhaf, “An informed genetic algorithm for the examination timetabling problem,” *Appl. Soft Comput.*, vol. 10, no. 2, pp. 457–467, Mar. 2010.

[38] M. Dorigo and T. Stützle, “The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances,” in *Handbook of Metaheuristics. International Series in Operations Research & Management Science*, vol. 57, F. Glover and Kochenberger G.A., Eds. Springer, Boston, MA, 2003, pp. 250–285.

[39] V. Maniezzo and A. Carbonaro, “An ANTS heuristic for the frequency assignment problem,” *Futur. Gener. Comput. Syst.*, vol. 16, no. 8, pp. 927–935, Jun. 2000.

[40] M. Eley, “Ant algorithms for the exam timetabling problem,” *Pract. Theory Autom. Timetabling VI*, pp. 167–180, 2007.

[41] W. Deng, J. Xu, and H. Zhao, “An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem,” *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[42] M. Dorigo and T. Stützle, “Ant colony optimization: Overview and recent advances,” in *In: Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham. https://doi.org/10.1007/978-3-319-91086-4_10*, vol. 272, 2019, pp. 311–351.

[43] T. Stützle and H. Hoos, “Improvements on the Ant-System: Introducing the {MAX}-

{MIN} Ant System," in *Artificial Neural Nets and Genetic Algorithms*, Springer Vienna, 1998, pp. 245–249.

[44] B. Bullnheimer, R. Hartl, and C. Strauss, "A New Rank Based Version of the Ant System - A Computational Study," *Manage. Sci.*, vol. 7, no. JANUARY 1999, pp. 25–18, 2016.

[45] F. Djannaty and A. R. Mirzaei, "Enhancing max-min ant system for examination timetabling problem," *Int. J. Soft Comput.*, vol. 3, no. 3, pp. 230–238, 2008.

[46] R. Abounacer, J. Boukachour, B. Dkhissi, and A. E. H. Alaoui, "A Hybrid Ant Colony Algorithm for the Exam Timetabling Problem," *Rev. Africaine la Rech. en Inform. Mathématiques Appliquées*, vol. 12, pp. 15–42, 2016.

[47] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks - Conference Proceedings*, 1995, vol. 4, pp. 1942–1948.

[48] S. Larabi Marie-Sainte, "A survey of Particle Swarm Optimization techniques for solving university Examination Timetabling Problem," *Artif. Intell. Rev.*, vol. 44, no. 4, pp. 537–546, 2015.

[49] A. Akhtar, "Evolution of Ant Colony Optimization Algorithm -- A Brief Literature Review," *arXiv*, 2019.

[50] Shu-Chuan Chu, Yi-Tin Chen, and Jiun-Huei Ho, "Timetable Scheduling Using Particle Swarm Optimization," in *First International Conference on Innovative Computing, Information and Control*, 2006, pp. 324–327.

[51] C. Blum and A. Roli, “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison,” *ACM Computing Surveys*, vol. 35, no. 3. pp. 268–308, 2003.

[52] R. Battiti and G. Tecchiolli, “The Reactive Tabu Search,” *ORSA J. Comput.*, vol. 6, no. 2, pp. 126–140, 1994.

[53] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” in *Readings in Computer Vision*, M. Fischler and O. Firschein, Eds. Elsevier, 1987, pp. 606–615.

[54] C. Gogos, P. Alefragis, and E. Housos, “An improved multi-staged algorithmic process for the solution of the examination timetabling problem,” *Ann. Oper. Res.*, vol. 194, no. 1, pp. 203–221, Feb. 2012.

[55] D. Abramson, M. Krishnamoorthy, and H. Dang, “Simulated annealing cooling schedules for the school timetabling problem,” *Asia-Pacific J. Oper. Res.*, vol. 16, no. 1, pp. 1–22, 1999.

[56] S. Abdullah, S. Ahmadi, E. K. Burke, M. Dror, and B. McCollum, “A tabu-based large neighbourhood search methodology for the capacitated examination timetabling problem,” *J. Oper. Res. Soc.*, vol. 58, no. 11, pp. 1494–1502, 2007.

[57] J. M. Thompson and K. A. Dowsland, “Variants of simulated annealing for the examination timetabling problem,” *Ann. Oper. Res.*, vol. 63, pp. 105–128, 1996.

[58] S. F. Galan and O. J. Mengshoel, “Generalized crowding for genetic algorithms,” in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference,*

GECCO '10, 2010, pp. 775–782.

[59] D. Gupta and S. Ghafir, "An Overview of methods maintaining Diversity in Genetic Algorithms," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 5, pp. 56–60, 2012.

[60] K. de Jong, "Adaptive System Design: A Genetic Approach," *IEEE Trans. Syst. Man. Cybern.*, vol. SMC-10, no. 9, pp. 566–574, 1980.

[61] C. R. Reeves, "Genetic algorithms and grouping problems," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3. p. 297, 2001.

[62] M. Linder and I. Sekaj, "Parallel genetic algorithms," in *Mendel*, 2011, pp. 9–15.

[63] M. Nowostawski and R. Poli, "Parallel genetic algorithm taxonomy," *Int. Conf. Knowledge-Based Intell. Electron. Syst. Proceedings, KES*, pp. 88–92, 1999.

[64] S. Pappu, K. T. Talele, and J. Mandviwala, "Application of Parallel Genetic Algorithm for Exam Timetabling Problem," *Int. J. Sci. Eng. Res.*, vol. 3, no. 9, pp. 1–4, 2012.

[65] D. Whitley, S. Rana, and R. B. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *J. Comput. Inf. Technol.*, vol. 7, no. 1, pp. 33–47, 1999.

[66] A. L. Corcoran and R. L. Wainwright, "A parallel island model genetic algorithm for the multiprocessor scheduling problem," in *Proceedings of the ACM Symposium on Applied Computing*, 1994, vol. Part F1294, pp. 483–487.

[67] V. Kolonias, G. Goulas, C. Gogos, P. Alefragis, and E. Housos, “Solving the examination timetabling problem in GPUs,” *Algorithms*, vol. 7, no. 3, pp. 295–327, Jul. 2014.

[68] A. Umbarkar and M. Joshi, “Review of Parallel Genetic Algorithm Based on Computing Paradigm and Diversity in Search Space,” *ICTACT J. Soft Comput.*, vol. 03, no. 04, pp. 615–622, 2013.

[69] M. Dorigo and T. Stützle, “The Ant Colony Optimization Metaheuristic,” in *Ant Colony Optimization*, 2018.

[70] A. M. Mohsen, “Annealing Ant Colony Optimization with Mutation Operator for Solving TSP,” *Comput. Intell. Neurosci.*, vol. 2016, 2016.

[71] T. Stützle and H. H. Hoos, “MAX-MIN Ant System,” *Futur. Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, 2000.

[72] E. Chen and X. Liu, “Multi-Colony Ant Algorithm,” in *Ant Colony Optimization - Methods and Applications*, 2011.

[73] S. Cheng and Y. Shi, “Diversity control in particle swarm optimization,” in *IEEE SSCI 2011 - Symposium Series on Computational Intelligence - SIS 2011: 2011 IEEE Symposium on Swarm Intelligence*, 2011, pp. 110–118.

[74] Y. Shi and R. C. Eberhart, “Parameter selection in particle swarm optimization,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1447, pp. 591–600.

[75] M. Baghel, Shikha Agrawal, and S. Silakari, “Survey of Metaheuristic Algorithms for Combinatorial Optimization,” *Int. J. Comput. Appl.*, vol. 58, no. 19, pp. 21–31, 2012.

[76] O. Abayomi-Alli, A. Abayomi-Alli, S. Misra, R. Damasevicius, and R. Maskeliunas, “Automatic Examination Timetable Scheduling Using Particle Swarm Optimization and Local Search Algorithm,” in *Data, Engineering and Applications*, 2019, pp. 119–130.

[77] I. X. Tassopoulos and G. N. Beligiannis, “A hybrid particle swarm optimization based algorithm for high school timetabling problems,” *Appl. Soft Comput. J.*, vol. 12, no. 11, pp. 3472–3489, 2012.

[78] E. L. Lawler, “Combinatorial Optimization : Networks and Matroids,” *Comb. Optim. networks matroids*, 1976.

[79] A. Wren, “Scheduling, timetabling and rostering - A special relationship?,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996, vol. 1153, pp. 46–75.

[80] E. Burke, D. de Werra, and J. Kingston, “Applications to Timetabling,” in *Handbook of Graph Theory*, CRC Press London, 2013, pp. 530–562.

[81] A. Schaerf, “Survey of automated timetabling,” *Artif. Intell. Rev.*, vol. 13, no. 2, pp. 87–127, 1999.

[82] E. K. Burke and Y. Bykov, “A late acceptance strategy – the new general-purpose optimisation metaheuristic,” *Memory*, no. February, pp. 1–29, 2009.

[83] M. W. Carter, G. Laporte, and S. Y. Lee, “Examination timetabling: Algorithmic strategies and applications,” *J. Oper. Res. Soc.*, vol. 47, no. 3, pp. 373–383, Mar. 1996.

[84] A. Muklason, A. J. Parkes, E. Özcan, B. McCollum, and P. McMullan, “Fairness in examination timetabling: Student preferences and extended formulations,” *Appl. Soft Comput. J.*, vol. 55, pp. 302–318, 2017.

[85] E. K. Burke and Y. Bykov, “An adaptive flex-deluge approach to university exam timetabling,” *INFORMS J. Comput.*, vol. 28, no. 4, pp. 781–794, 2016.

[86] N. Leite, F. Melício, and A. C. Rosa, “A shuffled complex evolution algorithm for the examination timetabling problem,” in *Studies in Computational Intelligence*, vol. 620, 2016, pp. 151–168.

[87] N. Leite, C. M. Fernandes, F. Melício, and A. C. Rosa, “A cellular memetic algorithm for the examination timetabling problem,” *Comput. Oper. Res.*, vol. 94, pp. 118–138, Jun. 2018.

[88] R. Bellio, S. Ceschia, L. Di Gaspero, and A. Schaerf, “Two-Stage Multi-Neighborhood Simulated Annealing for Uncapacitated Examination Timetabling,” *Comput. Oper. Res.*, vol. 132, p. 105300, Aug. 2021.

[89] A. K. Mandal and M. N. M. Kahar, “Solving examination timetabling problem using partial exam assignment with great deluge algorithm,” in *I4CT 2015 - 2015 2nd International Conference on Computer, Communications, and Control Technology, Art Proceeding*, 2015, pp. 530–534.

[90] M. Caramia, P. Dell'Olmo, and G. F. Italiano, "Novel Local-Search-Based Approaches to University Examination Timetabling," *INFORMS J. Comput.*, vol. 20, no. 1, pp. 86–99, Feb. 2008.

[91] B. McCollum *et al.*, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS J. Comput.*, vol. 22, no. 1, pp. 120–130, Feb. 2010.

[92] T. Müller, "{ITC}2007 solver description: a hybrid approach," *Ann. Oper. Res.*, vol. 172, no. 1, pp. 429–446, Oct. 2009.

[93] C. Gogos, P. Alefragis, and E. Housos, "A multi-staged algorithmic process for the solution of the examination timetabling problem," in *7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008*, 2008.

[94] Y. Bykov and S. Petrovic, "A Step Counting Hill Climbing Algorithm applied to University Examination Timetabling," *J. Sched.*, vol. 19, no. 4, pp. 479–492, 2016.

[95] A. J. Parkes and E. Özcan, "Properties of yeditepe examination timetabling benchmark instances," in *PATAT 2010 - Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, 2010, pp. 531–534.

[96] R. Qu and E. K. Burke, "Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems," *J. Oper. Res. Soc.*, vol. 60, no. 9, pp. 1273–1285, 2009.

[97] C. Blum and M. López-Ibáñez, "Ant colony optimization," in *Intelligent Systems*, 2016.

[98] D. Henderson, S. H. Jacobson, and A. W. Johnson, “The Theory and Practice of Simulated Annealing,” in *Handbook of Metaheuristics*, 2006, pp. 287–319.

[99] N. Abu-Baker, “(PDF) Research Methods in Computer Science,” 2018. [Online]. Available: https://www.researchgate.net/publication/323867128_Research_Methods_in_Computer_Science. [Accessed: 30-Apr-2021].

[100] S. Demeyer, “Research methods in computer science,” in *IEEE International Conference on Software Maintenance, ICSM*, 2011, pp. 25–30.

[101] V. Ramesh, R. L. Glass, and I. Vessey, “Research in computer science: An empirical study,” *J. Syst. Softw.*, vol. 70, no. 1–2, pp. 165–176, 2004.

[102] C. Johnson, “What is Research in Computing Science?,” 2006. [Online]. Available: http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html. [Accessed: 30-Apr-2021].

[103] H. Performance, “Center for High Performance Computing,” vol. 14, pp. 1–6, 2003.

[104] S. K. Smit and A. E. Eiben, “Parameter tuning of evolutionary algorithms: Generalist vs. specialist,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6024 LNCS, no. PART 1, pp. 542–551.

[105] A. E. Eiben and S. K. Smit, “Parameter tuning for configuring and analyzing evolutionary algorithms,” *Swarm and Evolutionary Computation*, vol. 1, no. 1. pp. 19–

31, 2011.

[106] F. Dobslaw, “Recent Development in Automatic Parameter Tuning for Metaheuristics,” *Proc. 19th Annu. Conf. Dr. Students-WDS*, pp. 54–63, 2010.

[107] C. Gogos, G. Goulas, P. Alefragis, V. Kolonias, and E. Housos, “Distributed scatter search for the examination timetabling problem,” in *PATAT 2010 - Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, 2010, pp. 211–223.

[108] S. K. Joshi and J. C. Bansal, “Parameter tuning for meta-heuristics,” *Knowledge-Based Syst.*, vol. 189, Feb. 2020.

[109] C. Huang, Y. Li, and X. Yao, “A Survey of Automatic Parameter Tuning Methods for Metaheuristics,” *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 201–216, Apr. 2020.

[110] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

[111] O. Maron and A. Moore, “Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation,” *Adv. Neural Inf. Process. Syst.*, vol. 6, pp. 59–66, 1993.