INVESTIGATIONS OF CONSTRUCTIVE APPROACHES FOR EXAMINATION TIMETABLING AND 3D-STRIP PACKING

By

NAM PHAM, BSc

Thesis submitted to the University of Nottingham

for the degree of Doctor of Philosophy

School of Computer Science and Information Technology

SEPTEMBER 2011

Abstract

This thesis aims at designing search methods that can produce competitive solutions and to some extent, are of higher generality than the state of the art search/optimisation systems. Attaining this aim would underpin the next generation of automated systems with the goal being to require less specialist knowledge in solving complex optimisation problems. The main challenge in this project is to develop systems of higher generality which can intelligently select, evolve or combine search methods (heuristics) to operate upon a wider range of problems and problem instances. This research follows that direction and contributes to the goal of exploring the generality boundary of this new trend of automating the design of search systems.

The main contributions in this thesis are divided into two parts. The first part investigates different approaches to combine constructive heuristics which are capable of producing good solutions for timetabling problems. Chapter 3 presents a weighted graph model for the exam timetabling problem where vertices and edges store several extra-attributes to improve the process of finding difficult exams and selecting timeslots for them. Chapter 4 investigates sequential and linear combinations of vertex-selection heuristics that have emerged from the weighted graph model. The results on the Toronto exam timetabling benchmark are compared with those obtained from other approaches in the literature.

The second part of the research focuses on raising the level of generality for search methodologies by investigating the use of estimation of distribution algorithms into a proposed hyper-heuristic for several optimisation problems. Chapter 5 presents an extended framework for the best-fit strategy for the three-dimensional strip packing

problem. Chapter 6 proposes a hyper-heuristic based on estimation of distribution algorithms. Then we investigate the level of generality of the hyper-heuristic by applying it to different problem domains (graph colouring, exam timetabling, and 3D strip packing). Experimental evidence indicates that the hyper-heuristic can operate on a wide range of problems to produce some competitive results. We also demonstrate the capability of identifying the effectiveness of the low-level heuristics. This may facilitate the development of efficient automated search systems in future research.

Finally, Chapter 7 evaluates all the results obtained and summarises promising future research directions.

Acknowledgements

I would like to thank my supervisors Dr. Rong Qu and Prof. Edmund Burke for all the support and guidance which they have given me throughout my doctoral study. Also, thanks to Prof. Graham Kendall who gave me valuable comments and advice during the first and second year review of my doctoral degree. I also thank the School of Computer Science, University of Nottingham for providing excellent facilities for my research and the Graduate School of the University of Nottingham for their research training programme.

This doctoral programme would have not been possible without the financial support of the Engineering and Physical Sciences Research Council (EPSRC) and the School of Computer Science, University of Nottingham.

I would like to thank my family and friends for their encouragement.

Finally, thanks to all members of the ASAP group, School of Computer Science for their support and excellent working environment.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

This chapter gives an introduction to the research areas that are investigated in this thesis. We firstly provide the background for the exam timetabling and three-dimensional strip packing fields. We also explore the motivation for developing automated approximate methods as well as more generalised methods for the problems within these fields. Secondly, we summarise the scope and objectives of this research. This research consists of two major parts. In the first part, we propose an enhanced weighted graph model, which gives rise to some novel constructive heuristics for exam timetabling problem. We also target different approaches to combining the use of those constructive heuristics. The remaining part of the thesis concerns a hyper-heuristic based on estimation of distribution algorithms to generate good sequences of constructive heuristics. To our knowledge, this is the first time an estimation of distribution algorithms is employed as the high-level search technique for a hyper-heuristic. Both exam timetabling problem and three dimensional strip packing problem are used as case studies. Then, we present the contributions of this thesis and provide the list of publications and research papers under review. Finally, we outline the structure of this thesis.

1.1 Background and Motivations

Meta-heuristics (e.g. tabu-search, simulated annealing and evolutionary algorithms, etc.) have been shown to be a particularly effective class of search methodologies for many combinatorial optimisation problems (Glover and Kochenberger, 2003). Many of the best results reported for such problems are from meta-heuristic approaches. However, from a practical point of view, there are still challenging issues for practitioners to use meta-heuristics successfully. Particularly, they usually demand intensive knowledge resulting in higher costs of implementation and maintenance.

In practice, many users often employ simple heuristics of inferior performance which are much easier to implement and maintain. There is a demand from many small and medium companies for cheaper problem solvers which are capable of producing good-enough solutions instead of high-cost, complex and domain-intensive systems. To improve the performance of algorithms based on such simple heuristics, one popular approach is to combine several heuristics for better decision making. Another approach is to use *hyper-heuristics*. A hyper-heuristic has been defined as "an automated methodology for selecting or generating heuristics to solve hard computational search problems" (Burke et al., 2009). This thesis only focuses on hyper-heuristics that select heuristics. Such hyper-heuristics can be understood as *high-level* search methodologies that operate on a set of *low-level* simple heuristics. They aim at choosing appropriate low-level heuristics subject to situations. The success of hyper-heuristics depends on how they adapt to a problem solving situation based on non-domain specific information, and associate it with a number of given low-level heuristics. Examples of domain-independent criteria include solution evaluation, computational time, and previous choices of heuristics. Therefore, hyper-heuristics are often regarded as a search methodology of higher generality than most implementations of heuristics or meta-heuristics. That is, we can select different sets of low-level heuristics for different problems (or problem instances) and reuse the high-level search methodologies without making many changes. This can help facilitate the development of systems which can operate on a range of related problems. Moreover, such systems can be deployed by non-specialised users with little knowledge of the problem domain. We will investigate the application of our proposed search methods on two different types of problems, i.e. exam timetabling problem and three-dimensional strip packing problem. These two problems are chosen due to two reasons. Firstly, there are a number of benchmark datasets which we can use to measure our methods' effectiveness. Secondly, since the methods proposed in this research will be based on constructive heuristics, we aim at testing their applications on problems with different numbers of constraints. Constructive approaches are usually more applicable for problems with higher numbers of constraints due to the difficulty to design effective local search methods. Basically, the number of constraints for the three-dimensional strip packing problem is considered to be more than the exam timetabling problem. This is due to the constraints on boxes being placed strictly inside the container, i.e. boxes are not allowed to have intersection with several sides of the container.

Timetabling is a very active research field covering many different types of problems with different characteristics. Most of them can be described as scheduling some *events* to certain *times* concerning a number of requirements, known as *constraints* which must be satisfied either completely or as much as possible. In this research field, exam timetabling problem (ETP) is very popular as searching for high quality solutions is challenging. In practice, manually solving an exam timetabling problem is time-consuming and not an easy task especially for institutions with large numbers of students, exams and with room limitations, etc. This problem has been addressed by the research community for over 40 years (Carter and Laporte, 1996). Most of the research efforts focus on developing efficient computer programs and systems to search for good timetables.

Unlike timetabling problems, packing problems can be seen much more often in many aspects of daily life. For example, people use their packing skills to arrange foods into a fridge or to put clothes into a suitcase. The general form of such problems can be described as follows: given a number of *items*, we need to allocate them subject to some specific *constraints* into a number of available *resources* so that the usage of resources is minimised. This objective in packing problems in industry represents financial benefits especially when resources have high unit cost. Due to spatial awareness and intuition, packing problems tend to be solved effectively by humans. However, in industry, these problems often involve a great number of instances. Hiring many "human solvers" to solve those instances may not be feasible or cost-effective. This is one of the reasons that research on using computers to automate the packing process has received much attention.

The packing problem we try to solve in this research is the three-dimensional strip packing problem (3D-SPP) which involves packing a set of boxes into a three-dimensional container with fixed width and height, but unconstrained length. The sides of the boxes need to be parallel to the container's walls. The goal is to pack all of the boxes into the container so that the resulting length is minimised. Most of the research on strip packing in the literature has only addressed one- or two-dimensional packing. Although the techniques in such research can be used in many real world applications such as paper, metal, glass or other sheet material cutting, multi-processor scheduling and basic pallet loading, they cannot be directly applied to other industrial problems with another dimension such as three-dimensional block cutting (e.g. wood and marble), truck or air cargo loading, or multi-dimensional limited resource scheduling.

The ETP and 3D-SPP both belong to the class of Combinatorial Optimisation problems (Papadimitriou and Steiglitz, 1982). This class refers to a set of many problems with discrete variables to be optimised such as the travelling salesman problem, the quadratic assignment problem and scheduling problems. Both ETP and 3D-SPP are NP-hard as they are generalisations of the classical NP-hard problems i.e. the graph colouring problem and the bin packing problem, respectively (Garey and Johnson, 1979). A problem is assigned to the NP (nondeterministic polynomial time) class if it is solvable in polynomial time by a non-deterministic Turing machine. A problem is said to be NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problem. A problem which is both NP and NP-hard is called an NP-complete problem. For problems in the NP class with inputs of large sizes, it would take too long to find the best solution using exact methods. Much of the research efforts in the literature are devoted to developing heuristics, meta-heuristics and other inexact algorithms. Although there is no guarantee of optimality, with good designs, such algorithms are capable of producing solutions of good quality in a limited computational time.

1.2 Scopes and Objectives

Raising the level of generality of search systems is a major objective of this research. However, we are also interested in designing search methods that can produce high-quality solutions. These two objectives usually contradict each other. Most of meta-heuristics are successful after significant amounts of parameter tunings while such tunings are usually associated with a low level of generality. In the first part of this thesis, our objective is to design approaches that can produce as good as possible solutions for the ETP. In the second part, we investigate more on approaches that not only can generate good solutions, but also can reduce the amount of parameter tunings for different problems. We do not aim at creating a system that requires no parameter tuning. Instead, we are more interested in systems where parameter tunings require less expert knowledge in particular problem domains.

The main focus in this research is on simple constructive heuristics. We take into account the state-of-the-art heuristics in the literature for the problems and propose some novel constructive heuristics. On their own, heuristics have been shown to be useful in a number of ways. They tend to be easy to understand and implement approaches, and solutions are generated quickly.

Early research on the ETP focused on constructive sequential heuristics. The principal idea is to use a graph colouring model to formulate the exam timetabling problem where the aim is to colour (schedule) the most troublesome or difficult vertices (exams) as early as possible. Here, a *troublesome* or *difficult* vertex is the one that is most likely to lead to a poor timetable if its colouring is deferred until later in the process. Based on this idea, we present an enhanced weighted graph model which extends the conventional graph model. This model holds and keeps track of more information relevant to each vertex and colour. A natural by-product of this approach is the emergence of some new, promising constructive heuristics for the ETP.

For the 3D-SPP, we aim at extending the effective *best-fit* strategy in the literature (Allen et al., 2011). Within the extended strategy, we propose simple constructive heuristics that simulate human intuition for different packing situations encountered in practice. The objectives of these extensions are not only to improve the effectiveness of the best-fit strategy, but also to serve as supporting components for the hyper-heuristics afterwards.

Simple heuristics are fast and easy to implement; however, using them separately may not produce competitive results. In many decision-making scenarios, it is often better to take into account different factors than to rely on only one factor. Motivated by that observation, different ways of selecting and combining constructive heuristics are examined. Firstly, we investigate a sequential combination of several heuristics to select difficult vertices in the ETP. Each heuristic, except the first one in a sequence, acts as the tie-breaker for the previous heuristics in the combination. Secondly, we design an improved strategy that uses linear combinations on some selected vertex-selection heuristics with suitable weight settings. Thirdly, motivated by the demands of systems of high generality, we introduce a hyper-heuristic based on the idea of estimation of distribution algorithms.

In the hyper-heuristic research field, it is often understood that a hyper-heuristic approach is of higher generality than most implementations of heuristics/meta-heuristics. Although there will not be a hyper-heuristic with the capability of efficiently solving all problems, it is interesting to know what scope hyper-heuristics can have in terms of generality. To address this research question, our hyper-heuristic is applied to four variants of the concerning problems. These are the ETP, the ETP with only a hard constraint (known as the graph colouring problem (GCP)), the 3D-SPP without a stability constraint, and the 3D-SPP with a stability constraint.

Another challenging issue in hyper-heuristic research is to determine how we can provide a good set of low-level heuristics. We design our hyper-heuristic which has the capability of identifying 'good' or 'bad' heuristics by itself in different situations. An analysis and future research directions on this learning capability will be provided in this thesis.

1.3 Contributions

The following list includes the major contributions of this research:

- An enhanced weighted graph model is proposed. It holds and keeps track of more information relevant to each vertex and colour.
- New constructive heuristics for the ETP are introduced. Their design is based on additional information pertaining to vertices and edges within the enhanced weighted graph model.
- An extension of the best-fit strategy for the 3D-SPP is introduced which is more suitable within a hyper-heuristic context then the original strategy.
- New constructive heuristics within the extended best-fit strategy for the 3D-SPP are proposed.
- Two strategies to combine the use of several simple constructive heuristics for the ETP to select vertices and colours are examined, i.e. sequential combination strategy and linear combination strategy.
- A particular linear combination is presented. It shows good results over a set of exam timetabling benchmarks. The choice of heuristics and the weight settings for the linear combination are justified.

- A hyper-heuristic based on estimation of distribution algorithms is proposed. Its effectiveness is demonstrated over applications on four different variants of the ETP and the 3D-SPP.

We also list here the publications and papers under review of the forementioned contributions:

- Carrington, J.R., Pham, N., Qu, R. & Yellen, J. (2007) An Enhanced Weighted Graph Model for Examination/Course Timetabling. *Proceedings of 26th Workshop of the UK Planning and Scheduling.*

- Burke, E.K., Pham, N., Qu, R. & Yellen, J. (2012) Linear Combinations of Heuristics for Examination Timetabling. *Annals of Operations Research*, 19(1), 89-109.

- Burke, E.K., Pham, N. & Qu, R. (2011) A Hyper-heuristic based on Estimation of Distribution Algorithms for Examination Timetabling, Under review in *European Journal of Operational Research*.

- Burke, E.K., Pham, N. & Qu, R. (2011) A Univariate Marginal Distribution Algorithm-based Hyper-heuristic for Three-Dimensional Strip Packing Problems, Under review in *IEEE-Transactions on Evolutionary Computation*.

1.4 Thesis Overview

This thesis is presented in seven chapters. The first chapter presents the overall background, scopes and objectives of the research. Chapter 2 describes the ETP and the 3D-SPP encountered in this research. We present the methodologies and benchmarks used in the literature for these problems. In addition, this chapter also gives an overview of the state-of-the-art constructive hyper-heuristics in the literature for the investigating problems.

Chapters 3 and 4 concerns the ETP. Chapter 3 presents an enhanced weighted graph model for the ETP where vertices and edges store several extra-attributes to improve the process of selecting difficult exams and selecting timeslots for them. The chapter gives an overview of all features of the graph model and new vertex- and colour-selection heuristics that have arisen.

Based on the model presented in Chapter 3, Chapter 4 investigates the sequential and linear combinations of vertex-selection heuristics. We observe that the sequential combination approach is actually a special case of the linear combination approach. For the linear combinations, the weights of the heuristic combinations define specific roles that each simple heuristic contributes to decisions to select vertices. Justifications for the design of the combined strategy are included in this chapter. Experiments show promising results compared to other constructive approaches in the literature.

Chapters 5 and 6 mainly address the 3D-SPP. Chapter 5 presents an extended version of the best-fit strategy for the 3D-SPP and introduces several extensions for it. Particularly, we introduce a number of novel gap-filling and tie-breaking heuristics within the extended strategy. The objective of designing these heuristics is mainly to provide the hyper-heuristic in the next chapter with a variety of choices to select boxes in different situations. We present a procedure to group similar boxes together to reduce wasted spaces. We also propose an adjustment technique to improve compactness by moving boxes around their positions.

Chapter 6 presents a hyper-heuristic approach which uses an evolutionary algorithm to search for the best heuristic sequences. It is based on an estimation of distribution algorithm that evolves the probability distribution of heuristics used at different stages within the fittest sequences. The proposed hyper-heuristic can be understood as a semi-automated method that learns online during the problem solving for the problems at hand. We demonstrate the generality of this hyper-heuristic by comparing the experimental results from different problems and problem variants with the best results from other constructive approaches in the literature. The tested problems include: the ETP, the GCP and the 3D-SPP with and without a stability constraint. Some of the results obtained represent the best reported results in the literature. We also present observations on the capability of the hyper-heuristic in identifying effective and ineffective heuristics based on the probability distribution learnt after the evolutionary process. We suggest that this may help facilitate the development of efficient algorithms in future work.

Chapter 7 concludes the thesis by identifying and summarising research issues raised in this work and how they can be applied in future research.

Chapter 2 Literature Survey

This thesis concerns the design of automated constructive search methodologies for different variants of the exam timetabling problem and the three-dimensional strip packing problem. Specifically, the main contributions include simple constructive heuristics and different approaches to combine them. To understand the background of this research, this chapter gives descriptions; reviews related heuristic approaches (heuristics, meta-heuristics, hyper-heuristics) in the literature and presents benchmarks of the investigated problems.

2.1 Exam Timetabling Problems (ETP)

Many institutions encounter exam timetabling problems. Creating timetables manually is hard particularly for large educational institutions with hundreds or thousands of exams and students. Those challenges have motivated much research effort to exploit computer power to construct exam timetables and also to develop approaches applicable for many exam timetabling scenarios. This problem draws much research effort due to its difficulty. Many benchmark problems in the literature still have not been solved to optimality. The ETP concerns the assignment of examinations into a number of timeslots. The inputs of the problem can be stated as follows:

- E_i is a collection of N examinations ($i = 1,...,N$).
- T is the number of timeslots.
- t_k ($1 \leq t_k \leq T$) specifies the assigned timeslots for exam k ($k \in \{1,...,N\}$).
- $C = (c_{ij})_{N \times N}$ is a matrix where each record, denoted by c_{ij} ($i,j \in \{1,...,N\}$), represents the number of students taking both exams i and j.
- M is the total number of students.

The constraints for the exam timetabling problems can be classified into two types: *hard constraints* and *soft constraints*.

- *Hard Constraints* must be satisfied in a strict manner. Firstly, a student sitting in two exams at the same time (*student-conflict*) must not occur. Another hard constraint is that the number of students must be less than or equal to the seat capacity of the assigned room. This hard constraint is, however, not enforced. The exam timetabling scenario in this research concerns no room capacity and is also called the *uncapacitated* exam timetabling problem. A timetable that satisfies all hard constraints is called *feasible*. A timeslot that an exam can be assigned in without causing any hard constraint conflict is called a *valid* timeslot for that exam.

- *Soft Constraints* are not compulsory but the degree of satisfaction of soft constraints indicates the quality of solutions. The exam timetabling scenario in this research has an objective of minimising the penalty caused by students taking exams too close together (*exam-spread*).

The objective function of the ETP in this research can be formulated as follows:

$$Min \frac{\sum_{i=1}^{N-1} F(i)}{M}$$

where:

$$F(i) = \sum_{j=i+1}^{N} c_{ij} \times P(t_i, t_j)$$

$$P(t_i, t_j) = \begin{cases} 2^5/2^{|t_i - t_j|} & if\ 1 \leq |t_i - t_j| \leq 5 \\ 0 & otherwise \end{cases}$$

subject to:

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} \times \lambda(t_i, t_j) = 0 \quad where\text{:}\ \ \lambda(t_i, t_j) = \begin{cases} 1 & if\ t_i = t_j \\ 0 & otherwise \end{cases}$$

$P(t_i, t_j)$ represents the penalty for assigning exams to two timeslots close together.

In this section, we first provide the references for some surveys on automated exam timetabling techniques. Then, we directly review the popular search methodologies in the literature for the ETP. Finally, we give a summary of the Toronto benchmark dataset.

2.1.1 Surveys on Exam Timetabling

There is a significant amount of exam timetabling research, especially in the last fifteen years. In the exam timetabling literature, many researchers proposed and developed solutions for specific schools or universities. These proposals were implemented by applying various techniques and then tested on some instances of real problems. Several surveys on automated exam timetabling problems have been published that classify timetabling problems and their methodologies.

An early survey by Carter (1986) presented an overview of practical applications for the exam timetabling problem from 1964 to 1984. These applications mainly used the model of graph colouring and worked on constructive heuristics to solve specific problems in particular schools. There was no comparison between the approaches in this period.

Carter and Laporte (1996) extended the survey and gave a classification of the algorithms. The four groups of techniques were reported as follows: cluster methods, sequential methods, generalised search strategies (i.e. meta-heuristics) and constraint based approaches. At that point in time, most algorithms solved different variations of the exam timetabling problems with simple constraints. To encourage more advanced research into exam timetabling problems, the authors published a set of test problems (Carter et al., 1996). This is a well-known benchmark nowadays for exam timetabling research and is also recognised under another name: the University of Toronto benchmark (shortly named as the Toronto dataset). Its details can be found in Section 2.1.6.

Burke et al. (1997) presented an overview of various techniques for exam timetabling problems and emphasised the increasing interest from researchers all over the world.

Schaerf (1999) conducted a survey on automated methods for educational timetabling problems and categorised them into: school, course and exam. The paper presented mathematical descriptions of the basic search and optimisation problems, variants of the problems, and solution approaches published in the literature. The basic search problem can be understood as the one which stops searching as soon as a feasible solution is found. The optimisation problem further requires the satisfaction of soft constraints to be optimised.

Burke and Petrovic (2002); Petrovic and Burke (2004) presented overviews of university timetabling problems and include automated methodologies for exam timetabling.

Lewis (2008) presented a survey of meta-heuristic techniques for university (including exam) timetabling problems. In this survey, meta-heuristic algorithms were loosely separated into three classes - one-stage optimisation algorithms, two-stage optimisation algorithms, and algorithms that allow relaxations. The author highlighted the importance of having meaningful comparisons between different algorithms on benchmark datasets.

Most recently, Qu et al. (2009b) published a comprehensive survey that summarised many recent search methodologies and their results in exam timetabling published since 1996.

2.1.2 Constructive Methods and Graph Colouring Heuristics

Early approaches in the literature solved the exam timetabling problems using a graph colouring model (Welsh and Powell, 1967; Garey and Johnson, 1979; Carter, 1986). The connection between a graph colouring model and basic timetabling (or scheduling) problems was firstly mentioned by Welsh and Powell (1967). This connection opened later research on graph colouring heuristics in timetabling (Mehta, 1981, 1982). The *graph (vertex) colouring problem* can be defined as follows. Given a graph, we need to find the smallest number of colours (the *chromatic number*) needed to obtain a feasible vertex colouring. A *feasible vertex colouring* is a colouring where adjacent vertices (two vertices joined by an edge) are assigned different colours. In a standard graph representation of an exam timetabling problem, exams to be scheduled are represented by vertices. The student-conflict hard constraint between two exams, indicating they should be assigned different timeslots, is represented by an edge between the corresponding vertices. Exams connected by an edge are called *neighbours*. If we associate each timeslot with a colour, then creating a conflict-free timetable is equivalent to constructing a feasible vertex colouring. Note that, for the approaches involving this model, we will use the terms *vertex* and *colour* interchangeably with *exam* and *timeslot*, respectively.

Based on the model, early research efforts were spent on improving a simple approximate algorithm that repeatedly executes the following two steps until a colouring is obtained.

1. Select an uncoloured vertex.

2. Find a colour for that vertex.

Most of the work in the exam timetabling literature focuses on designing constructive *vertex-selection heuristics* for step 1. In step 2, the colour chosen is usually one of the colours that cause minimum penalty.

Vertex-selection heuristics can be understood as simple ordering strategies to sort vertices by the degree of trouble they may cause if deferred until later. The first vertex in a particular ordering strategy can be considered as the most difficult vertex according to that strategy. The strength of this approximate algorithm lies mainly in the ability to produce a timetable rapidly. In addition, the ordering strategies are relatively simple and thus, easy-to-implement. In the literature, many local search/meta-heuristic methods integrated these heuristics to construct good initial timetables and iteratively improve them. We present, in the next sections, the techniques integrating graph colouring heuristics. An overview of such techniques can also be found in (Burke et al., 2004c).

Table 2.1 describes some of the most widely used ordering strategies. We also include a random ordering method in Table 2.1 for the purpose of reference in following sections. In the literature, there are also other variants of these strategies.

Heuristic	**Ordering Strategy**
Saturation Degree (Brelaz, 1979)	increasingly by the number of timeslots available for the exam in the timetable at the time
Largest Degree	decreasingly by the number of conflicts the exams have with the other exams
Largest Weighted Degree	the same as Largest Degree but each conflict (edge) is weighted by the number of students involved
Largest Enrolment	decreasingly by the number of enrolments for the exam
Random Ordering	randomly order the exams
Colour Degree	decreasingly by the number of conflicts the exam has with those scheduled at the time

Table 2.1 The most widely used ordering strategies for exam timetabling problems

Carter et al. (1996) integrated the use of the first five vertex-selection heuristics in Table 2.1 with backtracking to construct solutions. The backtracking strategy involved taking the exam with the least number of conflicts and reassigning it to another slot, and so on. The largest cliques were used to define the lower-bound on the number of timeslots required for the problem. The authors introduced and carried out experiments on the Toronto exam timetabling dataset. Each problem represents real-world timetabling data from different institutions. Observations showed that none of the heuristics was capable of outperforming all other heuristics over all tested problems.

Burke et al. (1998b) investigated the inclusion of random factors into some vertex-selection heuristics including Saturation Degree, Largest Degree and Colour Degree. Instead of selecting the first exam in an ordering strategy, two proposed strategies were: (1) *tournament selection*: generating a random subset of remaining exams and taking the first one ordered by the heuristic strategy; and (2) *bias selection*: instead of picking the first one, a random one is picked from the first *n* exams ordered by the heuristic strategy. The heuristic search techniques with random factors were tested on three instances of the Toronto dataset and were capable of producing better results compared to the method that uses vertex-selection heuristics with backtracking.

Carter and Johnson (2001) observed that real-world exam timetabling data often contains many cliques or even larger dense subsets of vertices that are almost cliques. They proposed a number of methods to modify the initialisation step to include larger subsets of exams by considering subsets that are almost cliques. The experiments were conducted on 11 instances in the Toronto benchmark. Improvements over previous approaches were reported.

Caramia et al. (2001, 2008) used a *greedy scheduler* to greedily create a feasible solution based on a varying priority associated with each exam. In addition, a *penalty decreaser* tries to perturb the solution without increasing the number of timeslots. In fact, the penalty decreaser may sometimes decrease the number of timeslots. These two processes are repeated until no improvement can be obtained. Then, a *penalty trader* is invoked which tries to trade penalties for timeslots. The author investigated different restarting schemes (*check pointing*) during the search and assignment of priority after a check pointing (*bridging priority*) to improve the performance. The algorithm produced some of the best results in the literature for some instances in the Toronto dataset.

Burke and Newall (2004) observed that using a single vertex-selection heuristic in the traditional approach of constructing timetables does not perform well in many cases. The effectiveness of a vertex-selection heuristic rather changes as the solution construction progresses. As an alternative, the authors investigated a process of adaptively changing ordering strategies. They introduced a heuristic modifier function that updates itself based on the performance of the associated heuristic in the previous steps. The value returned from this function was then added into the vertex difficulty estimation of the corresponding heuristic. Extensive experiments were conducted on 11 instances of the Toronto and the Nottingham dataset (see http://www.cs.nott.ac.uk/~rxq/data.htm). Most of the results were generally good and occasionally the approach improved the best results.

Asmuni et al. (2005); Asmuni et al. (2009) used fuzzy weights on a pair of ordering criteria to determine the difficulty of vertices. Promising results were reported on the Toronto benchmark dataset. Asmuni et al. (2007) extended the same fuzzy strategy to three ordering criteria and investigated the effect of altering fuzzy rules instead of fixing them. Corr et al. (2006) examined the application of neural networks to construct exam timetables. The difficulty of a vertex was assessed based on the returned values from three different orderings including the number of conflicts (degree), the number of students enrolled (enrolment) and the number of available slots left (saturation degree). The authors tested their methods on the Toronto dataset. The work demonstrated the capability to reduce the number of *unplaced* exams (those that cannot be assigned without causing a conflict) compared to approaches using single heuristics.

Abdul-Rahman et al. (2009) investigated the use of adaptive strategies that order the exams to be scheduled within a constructive approach. Firstly, a *constructor* uses either largest degree or saturation degree to assign exams into timeslots. Some exams might not be scheduled into a non-conflict timeslot. Then, an *analyser* uses different strategies to assign a higher level of difficulty to the unscheduled exams. The authors also used some strategies to shuffle the exams to be scheduled at each iteration. Good approximate solutions were observed by increasing the difficulty in certain ways. For one problem instance, the best result based on constructive heuristics at the time was improved.

Abdul-Rahman et al. (2011b) presented a constructive approach based on adaptive strategies that divides the set of exams into a difficult and an easy set. The difficult set consists of the exams that cause infeasibility during solution constructions. The authors also created a boundary set within the easy set. Their exams are merged or swapped with those in the difficult set using a strategy to improve solutions. If no improvement can be found, a roulette wheel selection strategy is employed to shuffle the current best examination ordering. The results were shown to be competitive to previously published constructive approaches. Abdul-Rahman et al. (2011a) improved their previous work by introducing a linear combination of heuristics with a heuristic modifier. A *difficulty_score* based on two strategies was used to determine the ordering of the examinations and the exam with the highest *difficulty_score* will be scheduled first. The approach was tested on the Toronto and the second international timetabling competition (ITC2007) datasets. The results are comparable to the previous approaches. It was observed that changing the weight settings for the linear combination has significant effects on the approximate solutions.

As stated before, the main strengths of the constructive approaches include the ability to quickly produce solutions and the use of many easy-to-implement heuristics. Research directions on this approach focus mainly on different techniques to measure the difficulty of vertices and adaptively change the ordering strategies during the solution construction. Part of this thesis's contributions follows these directions with the aim of raising the level of contribution for constructive approaches among other techniques for the ETP.

2.1.3 Local Neighbourhood Search

Apart from the simple constructive approaches that build a solution from scratch, the ETP (as well as many other combinatorial optimisation problems) can be solved by using local neighbourhood search. The basic idea of this approach is to repeatedly examine neighbours of the current solution to possibly move to a better one.

To explain how a local search approach and some meta-heuristics in the next sections solve a combinatorial optimisation problem and, we use the following notations:

- S = a set of candidate solutions and s = a solution
- $f(s)$ is an evaluation function applied to solution s
- an objective, either minimise or maximise $f(s)$ depending on the problem
- a neighbourhood function, $N(s)$, mapping any solution s in S to a set of 'related' s' solutions in S.

Local search is often an iterative algorithm that begins with an initial solution. Then in each iteration, it generates a number of solutions which are the neighbours of the current solution. Depending on the design, the algorithm can either keep the current solution or move to one of neighbours generated. A framework for general local search techniques is shown in Algorithm 2.1.

Algorithm 2.1 A Local Search Framework

```
initialise($s$)
while (not stop) do
    $s'$ := generate($s$)   // where $s' \in N(s)$
    if accept($s'$) then $s$ := $s'$
end while
```

There are two phases in this framework – an initialisation and an improvement phase. In the first phase, the *initialise(s)* procedure generates a starting solution. The improvement phase is iteratively repeated until stopping criteria are satisfied (the while loop). The *generate(s)* function returns a solution s' from the neighbourhood $N(s)$. The *accept(s')* function determines whether to replace the current solution s with the new solution s'.

Some examples of simple local search heuristics that correspond to different types of improvement include:

- Random Walk: It selects a solution at random from *N(s)*. On its own, this technique is the least effective. It is simply a way to perturb solution.
- Random Gradient Hill Climbing: a classic local search technique. At each step, a candidate solution, *s'* is selected at random from *N(s)*. If *f(s')* is an improvement over *f(s)*, *s'* will be accepted to replace *s*.
- Gradient Steepest Descent Hill Climbing: every solution in *N(s)* is evaluated and the solution with the best *f(s)* value is returned by the *generate(s)* function. If *f(s')* is an improvement over *f(s)* then *s'* is accepted to replace *s*, otherwise the algorithm terminates. With a large neighbourhood, computational complexity of this local search may become significant. A strategy to increase speed is to evaluate only a subset of *N(s)*.
- Next Gradient Hill Climbing: instead of iterating through all solutions in *N(s)*, we accept the first encountered *s'* during the iteration if *f(s')* is an improvement over *f(s)*.

Note that the notations and pseudo-codes in this section are for minimisation problems. The gradient steepest descent hill climbing will become the gradient steepest ascent hill climbing for maximisation problems.

Although they are simple, local search techniques are stuck in local optima in many situations, where all neighbouring solutions are worse than or equal to the current solution. They cannot guarantee to obtain the global optimum. The performance of a local search is dependent on how it decides to accept new solution states, the type of neighbourhood structure (single- or multi-neighbourhood and the size of the neighbourhood), efficient neighbourhood search techniques, and the initial solution (Ribeiro and Hansen, 2001).

In exam timetabling problems, many publications generate neighbours of a solution by applying a simple *atomic move*, i.e. move an exam to a different timeslot. Burke and Bykov (2008) proposed a different approach to improve exam timetables, namely the *late acceptance strategy*. In that strategy, the decision to move to a new solution is decided by the comparison between that new solution and the solution several steps earlier in the search. Their approach yielded some of the best results for the Toronto dataset.

Nevertheless, the simple local search techniques described in this section are rarely used on their own. Instead, we often see them being specially altered to escape local optima or being hybridised with other meta-heuristics (Burke et al., 1996; Burke and Newall, 1999; Schaerf and Di Gaspero, 2001; Merlot et al., 2003; Kendall and Hussin, 2005a, 2005b). They are employed to compare performance against other new algorithms (Corne and Ross, 1995; Burke and Newall, 2003).

2.1.4 Meta-heuristics

Meta-heuristics represent a class of heuristic techniques that have been successfully applied to solve a wide range of combinatorial optimisation problems over the years (Reeves, 1995; Osman and Kelly, 1996; Osman and Laporte, 1996; Glover and Kochenberger, 2003). The introduction of meta-heuristics was based on the demand for designing more effective techniques that can be applied to a larger number of applications in comparison with simple heuristics at the time. Glover and Laguna (1997) defined a meta-heuristic as follows:

"A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule."

For comparison purpose between our research and other applications for the ETP and the 3DSPP, we only focus on the following meta-heuristics in this research, i.e. tabu-search, simulated annealing, great deluge, greedy randomised adaptive search procedure, genetic algorithms and memetic algorithms. There are many other meta-heuristics; however, they are not the subject of this research.

Tabu-Search

First proposed by Glover (1986), tabu-search can be seen as a very effective meta-heuristic to solve many combinatorial optimisation problems, especially in the scheduling and timetabling fields. Many success stories of using tabu-search can be found in (Glover, 1986; Glover and Laguna, 1993; Osman and Laporte, 1996; Voß et al., 1999; Voß, 2001). Tabu-search was then clearly defined by Glover and Laguna (1997) as follows:

"Tabu-search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality."

Tabu-search can be understood as an extension of the steepest descent method. It overcomes the possibility of being stuck at local optima by incorporating adaptive memory and responsive exploration. During the exploration process, assuming that *s* represents the current solution, tabu-search accepts either improving or non-improving moves to a subset *N'(s)* of the neighbourhood *N(s)*. The move, however, has to have the best evaluation compared to all other solutions in *N'(s)*. The main reason to allow searching only in *N'(s)* is to reduce the time complexity of the search. In addition, it is very likely that searching in *N(s)* will direct the solution back to *s*, thus leading to cycles around a solution. Tabu-search uses a *tabu list* to determine *N'(s)*. During the search, some moves satisfying some *tabu restriction* criteria will be put into the tabu list and prohibited for a number of iterations (*tabu tenure*).

However, the restriction sometimes prohibits breakthrough moves. A *breakthrough* move can be understood as a move to a solution which is better than the currently-known best solution. Therefore, in tabu-search, an *aspiration criterion* is employed to allow solutions of sufficient quality to escape the tabu restriction. An appropriate design for aspiration criteria plays an important role in guiding the search process. The aspiration criteria can be time dependent, time independent, or aspiration by default. Reeves (1995) discussed other aspiration criteria by objective, search direction, and influence.

There are two different types of using memory to restrict some moves. A typical tabu-search uses a short-term memory to restrict a limited number of iterations prior to the current iteration (*recency-based restriction*). There is also a different type of restriction which uses *longer-term memory* to prohibit moves in a certain frequency over a larger number of iterations (*frequency-based restriction*). Tabu-search using longer term memory can be found in (Glover and Laguna, 1997; Taillard et al., 2001).

Many research studies have shown that the settings of the tabu tenure, the cardinality of *N'(s)* and the aspiration function are the deciding factors for a successful tabu-search. The pseudo-code for a tabu-search is presented in Algorithm 2.2. Starting with an initial solution *s* and an empty tabu list *T*, we repeatedly find the best neighbour *s'* of *s* which either is not in the tabu list or satisfies an aspiration criterion, add *s'* into the tabu list to restrict the backward move, release expired moves in the tabu list, and replace *s* with *s'*. The stopping conditions for tabu-search could simply be a maximum number of iterations or a maximum number of iterations since the last improvement.

Algorithm 2.2 Tabu-search Algorithm

```
initialise($s$)
$T$ := null
while (not done) do
    candidate_list := null
    for $s'$ in $N(s)$ do
        if ($s' \notin T$) or (aspiration($s'$) = true) then
            add $s'$ into the candidate_list
        end if
    end for
    $s'$ := get the best candidate solution in candidate_list
    add $s'$ into $T$
    release expired moves in $T$
    update_best_solution($s'$)
    $s := s'$
end while
```

Glover and Laguna (1997) investigated two important strategies used in tabu-search: *intensification* and *diversification*. These two strategies are wisely chosen based on the situation in a particular iteration. While *intensification* strategies involve tweaking parameters to focus the search in the area of high quality solutions, *diversification* strategies encourage the search to explore unvisited regions in the solution space. These strategies may become particularly useful if the landscape of the search is understood in advance.

Tabu-search is a very effective meta-heuristic in exam timetabling. Di Gaspero and Schaerf (2001) investigated a family of tabu-search based techniques to solve the exam timetabling problem. An objective function with varying weights on hard and soft constraints (shifting penalty mechanism) was employed with the purpose of exploring a different search landscape. The work used the atomic local move (see Section 2.1.3 for a definition). To decide which exam to move, they maintain two violation lists. One consists of exams that violate either hard or soft constraints, the other one consists of those violating only hard constraints. Various strategies using the shifting penalty mechanism and the two violation lists were also studied. The experiments were carried out on the Toronto and Nottingham datasets. This technique could beat the approach integrating constructive heuristics with clique initialisation and backtracking (Carter et al., 1996) in one instance - sta83 I. Di Gaspero (2002) further improved their work by employing a multi-neighbourhood strategy. He combined tabu-search with different neighbourhoods. The combinations were categorised into local search that focused on optimising the objective function (*recolour*), perturbing the current solution (*shake*) and obtaining more improvement (*kick*). The *recolour* and *shake* algorithms were repeatedly applied until no further improvement is achieved. After that, the *kick* was used to improve the solution obtained. The authors applied their approach on seven instances in the Toronto dataset and showed improvement from their previous work. In three instances, they could outperform the results from (Carter et al., 1996).

White and Xie (2001) developed a tabu-search called OTTABU to generate exam timetables using data provided by the University of Ottawa. The initial solution was generated using the constructive vertex-selection heuristic - largest enrolment. That initial solution was improved using tabu-search with the atomic local move. In addition to a recency short-term memory, the tabu-search also used a frequency long-term memory to improve the solution quality. If both short-term and long-term memory were used, only the role of long-term memory turned out to be significant. An analysis was carried out to estimate the appropriate size of the long-term memory. White and Xie (2004) expanded their research and included comparisons between these approaches on the Toronto dataset. The results were competitive compared to (Carter et al., 1996; Di Gaspero and Schaerf, 2001). The experimental results showed that employing memory can significantly improve tabu-search on real-world problesms.

Paquete and Stützle (2002) investigated a tabu-search for exam timetabling that added priorities to objectives. Two distinct strategies were proposed. Firstly, solutions were compared by the objective function of the higher priority objective and ties were broken by using the next lower priority objective. Secondly, solutions must satisfy all constraints associated with decreasing priority objectives. A tabu-search was used in which the tabu tenure was varying during the search based on the number of constraint violations at the time. While the first strategy produced more consistent results, the second strategy could obtain some comparable results with (Carter et al., 1996). The author observed that larger size problems require longer tabu tenure to increase diversity.

Simulated Annealing

Annealing is the process of cooling material in a heat bath. The material is heated to high energy where there are frequent state changes. It is then gradually cooled to a low energy state where state changes are rare. Simulated annealing (SA) emulates this physical process whereby the material is slowly cooled until a steady state is reached. Simulated annealing can be understood as an extension of the simple random gradient descent algorithm. Kirkpatrick et al. (1983) suggested that simulated annealing could be used to search for solutions in an optimisation problem whose objective is to converge to an optimum state.

Algorithm 2.3 Simulated Annealing Algorithm

```
initialise($s$)
$k$ := 0
while ($k < kmax$) do
    $t$ := temperature($k$ / $kmax$)
    $s'$ := $N(s)$
    if $P(f(s), f(s'), t)$ < random() then
        $s$ := $s'$
        update_best_solution($s'$)
    end if
    $k$ := $k$ + 1
end while
```

SA repeatedly considers neighbours s' of the current solution s and probabilistically decides between changing to s' or staying with s. Typically, improving moves are always accepted while worsening moves can be accepted probabilistically based on a function P of the temperature t and evaluation difference between $f(s')$ and $f(s)$. The temperature is proportionate with the probability of acceptance, i.e. high temperature means high acceptance probability and vice versa. The temperature is gradually reduced as the algorithm proceeds. Acceptance probability is calculated as $exp(-\delta/t)$ where δ is the magnitude of $f(s') - f(s)$ (the difference between the current and new solution evaluation function) and *t* is the current value of the temperature parameter. The pseudo-code in Algorithm 2.3 presents a simulated annealing that iterates through *kmax* iterations. The temperature is calculated as a function of the remaining number of iterations.

Aarts and Korst (1989) showed the importance of setting the cooling schedule (start temperature, end temperature, temperature reduction) and the neighbourhood structure to the result of SA. The temperature is set at a high value at the beginning to allow more worse moves and then it is slowly reduced to reach equilibrium.

Simulated annealing is a popular meta-heuristic among various optimisation problems. An introduction to simulated annealing techniques can be found in (Dowsland, 1995; Aarts et al., 2005) .

Early SA approaches for the exam timetabling problem include (Thompson and Dowsland, 1996, 1998). The authors solved the problem in two stages. The first stage generated a feasible exam timetable which was then improved on the soft constraints using simulated annealing. The authors further investigated the Kempe chain neighbourhood. Instead of allowing moving a single exam, a chain of exams were considered. Their experiments observed that the solution quality depends not only on the cooling schedule and neighbourhood but also on the way it is sampled. The system was reported to be successfully applied in Swansea University.

Bullnheimer (1998) used a model for quadratic assignment problems to formulate a practical exam timetabling problem. The soft constraint prefers timetables with longer study time between exams for students. He used simulated annealing to search for exam timetables based on two neighbourhood structures (four different timeslot moves or randomly pick an exam to move to another random timeslot). The system was used to build timetables at the University of Magdeburg.

Great Deluge

Dueck (1993) introduced *great deluge* algorithm which is similar to a certain extent to the simulated annealing algorithm. The major difference between the great deluge algorithm and the SA algorithm is that SA has probability in accepting worsening moves. The basic idea of the great deluge is to accept worsening moves if the solution quality is better than a certain level *B* (or threshold). This level is set as the evaluation of the initial solution and gradually reduced by a decay factor ΔB. The decay factor and an estimation of desired quality represent the parameters in this approach. The pseudo-code presented in Algorithm 2.4 demonstrates a great deluge algorithm over *kmax* number of iterations. In the first phase, a solution s is initialised and *OR* is set as a desired quality of the final solution. The level is initialised as the evaluation of *s* while the decay factor is set as (*B* - *OR*) / *kmax*. The algorithm repeatedly finds neighbour *s'* of *s*, accepts to move to *s'* if the evaluation of *s'* is below the current threshold and reduces the threshold by a decay factor after each iteration.

Burke et al. (2004a) investigated the great deluge algorithm for the ETP. The first feasible solution was required to define the threshold, so it was obtained by running the saturation degree graph colouring heuristic for a number of times and the best solution (mostly feasible) is taken. Experiments were conducted on the Toronto and the Nottingham datasets and the method produced some of the best results compared to approaches in (Carter et al., 1996; Di Gaspero and Schaerf, 2001). The authors also implemented a simulated annealing approach but found that the results obtained were inferior to using the great deluge.

Algorithm 2.4 Great Deluge Algorithm

```
initialise($s$)
$s_{best} := s$
set the optimal rate for $OR$
$B := f(s)$  // Set the initial level
$\Delta B := (B - OR) / kmax$
$k := 0$
while ($k < kmax$) do
    $s' := N(s)$

    if $f(s') < f(s_{best})$ then
        update_best_solution($s'$)
        $s := s'$
    else
        if $f(s') \leq B$ then
            $s := s'$
        end if
    end if
    $B := B - \Delta B$
    $k := k + 1$
end while
```

Yang and Petrovic (2005) developed a case based reasoning methodology to select an appropriate hybridisation of great deluge meta-heuristics with sequential exam-selection heuristics integrated with other techniques, including clique detection and backtracking. Some of the best known results for the Toronto dataset were obtained using this methodology.

Greedy Randomised Adaptive Search Procedure

Greedy randomised adaptive search procedure (GRASP) is a multi-start iterative procedure consisting of two phases: *construction* and *local improvement* (Resende and Ribeiro, 2003). The construction phase greedily builds solutions on which the local improvement will be applied. The process is repeatedly applied and the best solution during the local improvement phase is recorded. In the GRASP, the construction phase is designed to ensure the variability of solutions. First, this phase ranks all candidate elements based on the dynamic benefit of selecting an element. Then, well-ranked candidate elements are often placed in a *restricted candidate list* (RCL) and randomly or greedily selected to build up solutions. This construction phase can be viewed as a good sampling of initial solutions that lie in promising areas of the search space. The pseudo-code for the GRASP is shown in Algorithm 2.5.

Algorithm 2.5 Greedy Randomised Adaptive Search Procedure

```
k := 0
while (k < kmax) do
    s := Ø
    evaluate the incremental cost of the candidate elements
    while (s is not a complete solution) do
        build the restricted candidate list (RCL)
        randomly select an element e from the RCL
        add e into s
        re-evaluate the incremental costs
    end while

    while (s is not locally optimal) do
        s' := N(s)
        if  f(s') < f(s) then
            s := s'
        end if
    end while

    update_best_solution(s)
    k := k + 1
end while
```

A survey of using GRASP on many operational research problems and industrial applications can be found in (Resende and Ribeiro, 2003, 2005).

Casey and Thompson (2003) developed an enhanced GRASP algorithm and applied it to exam timetabling problems. The authors focused on tuning parameters to find the best constructive heuristic and the size *n* for the RCL. Exams were ordered according to one of the constructive vertex-selection heuristics with the saturation degree being the most effective. In each iteration, an exam was selected from the RCL using a roulette wheel selection and is assigned to the first available period. Then the backtracking technique with a tabu list was applied if no feasible period for the selected exam was found. The improvement phase was implemented with a neighbourhood of Kempe chains. To encourage the diversity in the searching process, some techniques were used including a simulated annealing and memory function. The method was tested on the Toronto dataset and produced robust results across all instances. It could improve best results in some instances.

Genetic Algorithms

The meta-heuristics discussed so far all belong to the class of methods that search from a single solution. There is another class that has also attracted significant research interest in exam timetabling (and indeed many other applications): *population-based meta-heuristics*. We present a review on some successful techniques for the exam timetabling problems in this class.

Genetic algorithms (GAs) are evolutionary algorithms which generate solutions to optimisation problems using techniques inspired by natural evolution. GA work can be seen as early as in (Fraser, 1957) and (Bremermann, 1962). After that, the success of GAs was popularised in many publications e.g. (Holland, 1975) and (De Jong, 1975). Introductions to genetic algorithms and their various applications can be found in (Goldberg, 1989; Sastry et al., 2006).

Algorithm 2.6 A Genetic Algorithm Framework

```
$t$ := 0
generate initial population $P(0)$
evaluate $P(0)$
**while** (not done) **do**
    select a set of promising solutions $Parents(t)$ from $P(t)$
    generate offspring $O(t)$ from $Parents(t)$ using genetic operators
    evaluate $O(t)$
    incorporate $O(t)$ and $P(t)$ into $P(t + 1)$
    $t := t + 1$
**end while**
```

GAs represent a meta-heuristic that operates on a population of strings (*chromosomes* or *genotypes*) which encode candidate solutions (*individuals*). Each string consists of a number of encoding characters (*genes*). Traditionally, solutions are represented in binary, i.e. strings of *0*s and *1*s, but other encodings are also possible. The evolutionary process usually starts by generating a random initial population. Then, the fitness of every individual in the population is evaluated. The selection procedure is then applied to select individuals with preference on the ones with higher fitness values. This selection procedure influences the search direction towards promising areas in the search space. The offspring are then created by applying genetic operators e.g. *crossover* (taking parts of two selected parents) and *mutation* (randomly making changes in some genes). After evaluating offspring, a procedure is used to determine which individuals of the current population and the set of offspring get to survive to the next generation. Commonly, the algorithm terminates when a certain number of generations or a certain fitness level is reached. Usually, controlling parameters of a simple genetic algorithm include the population size, the frequency of crossover and the mutation rate. A general framework for GAs can be found in Algorithm 2.6.

Ross et al. (1998) developed a GA for exam timetabling problems. They used a simple direct representation for a timetable, i.e. a sequence of integer numbers where the i^{th} number represents the assigned timeslot for exam i.

Burke et al. (1998a) analysed the initialisation phase of a GA. Specifically, the diversity of solutions generated from several graph colouring heuristics were examined. A random factor is used in the initialisation process. That was to either select exams based on a particular ordering strategy from a randomly generated subset of exams to be scheduled or select randomly an exam from the best *n* exams according to an ordering strategy.

Erben (2001) developed a GA for the exam timetabling problem with only hard constraints. In the algorithm, genes of exams with the same timeslot were grouped together. Special crossover and mutation operators were designed for the group encoding. Although the computational time was small, the results obtained by this approach were not impressive.

Cote et al. (2005) investigated an evolutionary algorithm for the exam timetabling problem. Instead of using the recombination operator, the approach used two local search operators, i.e a tabu-search and a simplified variable neighbourhood descent with Kempe chain. The evaluation function used a ranking procedure based on Pareto strength. The authors tested the algorithm on the Toronto and some other datasets. The approach was generally robust and obtained competitive results on a number of benchmark problems against other methods in the literature. The paper also provided a review on all the state-of-the-art approaches on the Toronto dataset at the time.

Ülker et al. (2007) used grouping representation to tackle the graph colouring problem and exam timetabling problem, i.e. vertices (or exams) with the same colour (or timeslot) are grouped together. The authors employed *Linear Linkage Encoding* scheme (Du et al., 2004) for groups and tested several new crossover operators on both graph colouring and the exam timetabling benchmarks. The results obtained by these crossover operators could match the results obtained by well-known crossovers investigated in (Galinier and Hao, 1999).

Memetic Algorithms

Memetic Algorithms (MA) (Moscato, 1989, 1999) represent a class of evolutionary algorithms. MA was named from the term "meme" by Dawkins (1990). The main difference between GA and MA is that genes are basically not changed during the evolutionary process while memes can change. Typically, MA includes knowledge of the problem in the form of heuristics, local search techniques, or truncated exact methods. Most memetic algorithms in the literature are the combination of genetic algorithms with a local improvement technique on every individual after a generation.

A review on MA for exam timetabling problems can be found in (Burke and Landa Silva, 2004).

Burke et al. (1996) developed a memetic algorithm for the exam timetabling problem. They investigated a hill climbing operator and different mutation operators for reassigning a set of exams. Hill climbing could improve the quality of solutions more than mutation operators, but the computational time was also higher. The authors further investigated the initialisation strategies and the diversity in the populations of memetic algorithms (Burke et al., 1998a). The trade-off between solution quality and diversity can potentially benefit memetic algorithms. Burke and Newall (1999) further studied a multi-stage algorithm to solve the exam timetabling problem. The approach used different constructive vertex-selection heuristics. It assigned each time a subset of the most difficult exams. To avoid infeasibility, backtracking and look-ahead techniques were applied. The approach fixed all exams assigned in the previous stage and applied the above memetic algorithm to the current stage. High quality solutions were obtained especially for large problem instances. Running time was improved significantly compared to the previous approach.

2.1.5 Hyper-heuristics

In the last decade or so, hyper-heuristics have emerged as a major area of study (Burke et al., 2003). Burke et al. (2009) gave a formal definition of a hyper-heuristic as follows.

"Hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems."

Unlike many existing search methodologies in the literature that work in the search space of solutions, hyper-heuristics work in the search space of heuristics.

Research motivation for hyper-heuristics originated from the goal of providing generic methodologies for combinatorial optimisation problems. Cowling et al. (2000) described hyper-heuristics as a class of *knowledge poor* methods with the aim of selecting appropriate heuristics from a pre-defined set of heuristics during the search. The success of hyper-heuristics depends on how they adapt to a problem solving situation based on no (or in some cases, very little) problem-domain-specific information. Examples of domain-independent criteria include solution evaluation, computational time, and previous choices of heuristics. Therefore, hyper-heuristics are often regarded as a search methodology of higher generality compared to most implementations of heuristics or meta-heuristics. They can help facilitate the development of systems which can operate on a range of related problems. In fact, most of the developed hyper-heuristics in the literature turned out to be cheaper to implement and easier to use when compared against other knowledge intensive meta-heuristics (Burke and Kendall, 2005). Moreover, such systems can be deployed by even non-specialised users with little knowledge of the problem domain.

Hyper-heuristic research was categorised in (Burke et al., 2009) into two main categories, namely *heuristic-selection* and *heuristic-generation*. Heuristic-selection approaches involve high-level search methodologies that intelligently choose from a given set of simple low-level heuristics to construct or improve a solution. Heuristic-generation approaches focus on generating novel heuristics whose components are building blocks or parts of known heuristics. This research focuses only on the heuristic-selection direction with the objective of better utilising the simple constructive heuristics. A comprehensive survey of the heuristic-generation approaches can be found in (Özcan et al., 2008; Burke et al., 2012). Hereafter, the term *hyper-heuristics* is used with the assumption that we are referring to heuristic-selection methods. In the heuristic-selection category, hyper-heuristics can be considered as either constructive or perturbative. On the one hand, general constructive methods build a complete solution from scratch through a definite number of decision steps. Hyper-heuristics based on constructive heuristics can be understood as methodologies that repeatedly select suitable constructive heuristics from a given set and apply them to the partial solution being constructed. On the other hand, hyper-heuristics based on perturbative heuristics, in general, start with a complete solution, and then select one from a given set of neighbourhood structures and acceptance criteria to iteratively improve that solution. In general, two different types of perturbative heuristics can be identified as in (Özcan et al., 2008) which are *mutational heuristics* and *hill climbers* . The main difference between the two types is that hill climbers aim to obtain a better candidate solution at each step while mutational heuristics are not expected to do so.

A popular framework for hyper-heuristics is presented in Figure 2.1. In this framework, there are two search layers, namely the high-level search and the low-level search, separated by a domain barrier. The high-level layer includes a search method or learning mechanism that manipulates the selection of heuristics in the low-level layer. The set of low-level heuristics can include simple constructive/local search heuristics or more complex meta-heuristics. They are methods that work directly upon problem solutions. The domain barrier between these two layers is employed to raise the level of generality. It can be seen as a restriction for the high-level search techniques on using domain-dependent information.

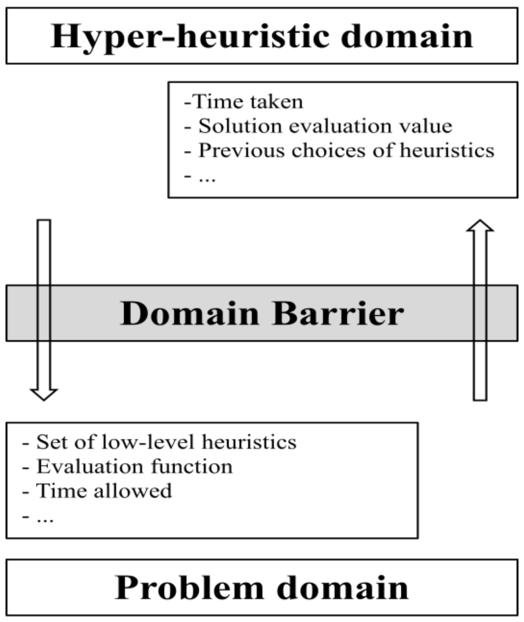


Figure 2.1 A hyper-heuristic framework

Although the term *hyper-heuristic* was firstly used by (Denzinger et al., 1996), the idea of hyper-heuristics can be traced back to the early 60s (Fisher and Thompson, 1961, 1963; Crowston et al., 1963); in production scheduling. During the 90s, hyper-heuristics attracted a significant rise in research interest, mostly in the job-shop scheduling field (Storer et al., 1992, 1995; Fang et al., 1993, 1994; Dorndorf and Pesch, 1995; Norenkov and Goodman, 1997; Hart and Ross, 1998; Hart et al., 1998). They mainly encoded the heuristic choices in the high-level search as sequences of heuristics. In these sequence representations, heuristics are sequentially applied either to build the current partial solution or to improve a complete solution. In the last decade, exam timetabling problems have been tackled by many hyper-heuristic approaches both constructively and perturbatively.

Hyper-heuristics based on Constructive Heuristics

There have been two major directions in constructive hyper-heuristic research for exam timetabling. The first one searches for good sequences (or permutations) of heuristics and applies them sequentially. The second one uses different techniques to find good combinations of ordering criteria to measure exam difficulty.

Terashima-Marín et al. (1999) introduced the first hyper-heuristic for the ETP which works on the search space of constructive vertex-selection heuristics. The research was motivated by observations that some vertex-selection heuristics work better for solving certain problem instances. Instead of following the state-of-the-art evolutionary algorithms at the time to search for actual timetables, the authors evolved heuristic choices for constructing timetables. Their approach used a non-direct chromosome representation for the construction process. In particular, it used heuristic A to choose an exam and heuristic B to place it in a timeslot until a certain condition C first holds. After that, the process continues using heuristic D to choose and heuristic E to place an exam, and so on. The method showed promising results on the Toronto dataset at that time.

Ahmadi et al. (2003) proposed several constructive heuristics and used a weighted decision function to solve a capacitated exam timetabling problem. The set of low-level heuristics included 6 vertex-selection heuristics, 2 colour-selection heuristics, and 3 room-selection heuristics. For each of those three types of heuristics, a random ordering strategy was also included. Then, the author used a variable neighbourhood search algorithm to find good combinations of weighted heuristics. The only experiment was conducted on a real dataset from the University of Nottingham due to its rich set of constraints.

Asmuni et al. (2005, 2009) investigated the potential of implementing a fuzzy system for solving exam timetabling problems. Fuzzy models were applied to combine the following three heuristics: largest degree (LD), largest enrolment (LE), and saturation-degree (SD). The model took any two of the three heuristics as inputs and associated with each heuristic a *grade of membership* based on a provided *membership function*. Then, the model employed rule set matrices to return a variable *examweight* that measures the overall difficulty of exams. For example, one rule of the fuzzy system was "IF LD is high AND LE is high THEN *examweight* is very high". The authors used a tuning phase to adjust the membership function until the best possible system performance was achieved. The proposed system was tested on 12 instances of the Toronto exam timetabling benchmark and the results showed improvement in one instance (*yor-83-I*) compared to other constructive approaches. Asmuni et al (2007) extended the same fuzzy strategy to three ordering criteria and investigated the effect of altering fuzzy rules instead of fixing them.

Ross et al. (2004); Ross and Marín-Blázquez (2005); applied a messy genetic algorithm (Goldberg et al., 1990) hyper-heuristic based on constructive vertex-selection heuristics to the capacitated exam timetabling problem. The messy GA was used to search for a set of labelled points in a simplified problem-state-description space and a heuristic is associated with a point. Given a problem state, the hyper-heuristic idea was to find its nearest labelled point, applying the associated heuristic to extend the growing solution towards a complete solution. The low-level heuristics consisted of 16 vertex-selection, 20 colour-selection, and 5 room-selection heuristics. Fast and simple algorithms were generated that offered good performance over a range of exam timetabling problems.

Burke et al. (2005); Burke et al. (2006) used a knowledge discovery technique, case-based reasoning (Leake, 1996) as a heuristic selector and applied it to exam timetabling problems. A set of four vertex-selection heuristics: largest-degree, largest degree with tournament selection, colour degree, and saturation degree was used in the low-level search. In (Burke et al., 2006), tabu-search was employed to discover the most relevant features used in evaluating the similarity between problem solving situations in case-based reasoning. The objective was to choose the best heuristics from the most similar previous problem-solving situations to construct good solutions for the problem at hand. In (Burke et al., 2005), different ways of hybridising the low-level graph heuristics (with and without case-based reasoning) were compared for solving the Toronto dataset. It was observed that employing knowledge-based techniques rather than randomly/systematically hybridising heuristics in a hyper-heuristic framework presented better results.

Burke et al. (2007) developed a hyper-heuristic framework based on commonly used graph colouring heuristics. The set of low-level heuristics included largest degree, largest weighted degree, colour degree, largest enrolment, saturation degree and a random ordering heuristic. The hyper-heuristic used a sequence representation in which each element corresponds to one low-level heuristic. The heuristics in a sequence were sequentially applied to schedule one exam into the growing solution. Tabu-search was employed as the high-level search method to search for fittest sequences. This work addressed the existence of two different search spaces in a constructive hyper-heuristic (a heuristic combination search space and a solution search space). Competitive results were obtained on course and exam timetabling benchmark instances. Within the same framework, Qu and Burke (2005) employed variable neighbourhood search on two different sets of neighbourhoods. Each neighbourhood structure involves randomly changing a small number of heuristics in a heuristic sequence. An automated heuristic construction approach was presented in (Qu et al., 2009a) to adaptively hybridise the Saturation Degree heuristic with the Largest Weighted Degree heuristic, at different stages of the solution construction for exam timetabling. Promising results for the Toronto dataset were obtained using these hybridisations in a short running time.

Qu and Burke (2009) provided a formal definition of an extended version for their graph-based framework. Three other high-level search methods were experimented with and comparisons were made with the previously implemented tabu-search (a steepest descent method, an iterated local search and a variable neighbourhood search). The authors suggested that the quality of the solution obtained depends little on the choice of neighbourhood structures. In addition, iterative techniques such as iterated local search and variable neighbourhood search were found to be more effective for traversing the heuristic combination search space than tabu-search. The authors observed that the heuristic search space is likely to be smooth and contain large plateaus (i.e. areas where different heuristic sequences can produce similar quality solutions). Qu and Burke (2009) also investigated hybridisations of their hyper-heuristic framework with local search operating on the solution space. The motivation was from the observation that not all problem solutions can be mapped into a heuristic combination. The hybridisations produced competitive results compared with state-of-the-art approaches on the Toronto benchmark. Ochoa et al. (2009) further investigated this graph-based framework and studied a fitness landscape analysis on the search space of heuristic combinations. Similar to the observation in (Qu and Burke, 2009), the study showed a high level of neutrality in its landscapes. The most promising feature suggested by the study was that the landscape has the shape of a globally convex or big valley structure. It indicates that an optimal heuristic combination is likely to be surrounded by many local minima. The study also confirms a positional bias in the heuristic combination search space.

Heuristics in the early stage of building up a solution tend to be of greater significance than those in the later stages.

Pillay and Banzhaf (2007) investigated a genetic programming based (GP-based) hyper-heuristic system for exam timetabling problems. The approach used a similar heuristic combination representation i.e. sequences of heuristics. However, the length of a sequence was varying with an upper bound limit. Each element in a sequence was a single character representing one of five low-level heuristics: largest degree, largest enrollment, largest weighted degree, saturation degree, and highest cost (decreasingly order exams by their highest possible proximity cost caused by a feasible assignment). Genetic programming was applied to evolve good sequences. Experiments showed that this evolutionary system outperformed previous hyper-heuristics on a number of instances in the Toronto dataset. Pillay (2008) extended this work by investigating the performance of the system using three different heuristic combination representations: fixed length (FHC), variable length (VHC), and *n* – times (NHC). For the NHC representation, each heuristic in a combination is associated with a number representing the number of times that heuristic must be applied in order. The results suggested that the FHC representation did not perform as well as the other two representations. Nevertheless, the performance of VHC and NHC was varying and problem dependent.

Pillay and Banzhaf (2009) developed an alternative approach to combine two or three low-level heuristics as tie-breakers in each decision to select an exam. Pillay (2009) used a genetic programming approach to evolve functions that calculate the difficulty of assigning an exam during the timetable construction process. Each approach could obtain improvements to some instances in the Toronto dataset compared to other constructive approaches.

Hyper-heuristics based on Perturbative Heuristics

Perturbative hyper-heuristic publications have started to appear only recently. Bilgin et al. (2007) employed different heuristic selection and move acceptance mechanisms and used their combinations within hyper-heuristics. The authors experimented with many different combinations on a set of exam-timetabling problems. The comparison results showed that the combination (*Choice Function – Monte Carlo*) outperforms others. The *Choice Function* selects heuristics by analysing their single or pairing performance (improvement and execution time). It also considers the overall performance, i.e. to focus or broaden the search based on the improvement rate. The *Monte Carlo* move acceptance mechanism was actually a simulated annealing method, i.e. all improving moves are accepted while non-improving moves can be accepted based on a probability that is decreased over time.

Ersoy et al. (2007) investigated *hyperhill-climbers* which adapt the hyper-heuristic mechanism into the hill climbing method within memetic algorithms. The authors investigated a set of deterministic, adaptive and self-adaptive hyperhill-climbers on the Toronto exam timetabling problem instances. The performance of the self-adaptive hyperhill-climbers was found to be better than the performance of the adaptive ones.

Özcan et al. (2009) proposed a late acceptance strategy and compared its combination with a different heuristic selection mechanism. The *late acceptance strategy* only accepts a move to a solution *s'* if *s'* is improved from the previously visited solution L-step backwards from the current solution. The results on an exam timetabling problem showed that randomly selecting heuristics at each step performs best with the late acceptance strategy.

Özcan et al. (2010) developed a Reinforcement Learning and Great Deluge hyper-heuristic and applied it to a capacitated exam timetabling problem at Yeditepe University. The hyper-heuristic selects heuristics using their utility values that were adaptively maintained according to their performance. The Great Deluge technique was used as the move acceptance mechanism. The authors presented their experimental results with different settings. This approach improved performance over a Simple Random - Great Deluge hyper-heuristic on the tested problems.

Burke et al. (2010) investigated a set of Monte Carlo based selection hyper-heuristics. Similar to the idea of simulated annealing, the Monte Carlo move acceptance methods allow acceptance of worsening moves based on a parametric probability function. The additional use of reheating within simulated annealing in this research was shown to be a very promising hyper-heuristic component. The authors also compared and analysed the results obtained from combining different heuristic selection and move acceptance mechanisms.

2.1.6 Summary of the Toronto Dataset

The University of Toronto benchmark dataset consists of 13 real-world exam timetabling problems firstly introduced by (Carter et al., 1996) (available at http://www.cs.nott.ac.uk/~rxq/data.htm). Since its introduction, it has attracted much research effort from the timetabling research community. During the years, researchers have reported the best results obtained along with the development of advanced algorithms. This dataset still remains an interesting challenge as optimal solutions for all instances have not been found yet. Therefore, we test our methods in the next chapters on this dataset to compare results against many other existing methodologies. Table 2.2 shows characteristics of the instances. Two versions of the dataset have been circulated under the same name over the last ten years. We used the naming convention provided in (Qu et al., 2009b). This dataset follows the formulation for the ETP provided in Section 2.1.

Qu et al. (2009b) provided an extensive survey on all search methodologies with associated best reported results for this dataset.

In Table 2.2, the *density* represents the ratio of the number of edges that have at least one common student to the total number of edges of the conflict matrix. Note that there are also other exam timetabling benchmark datasets in the literature e.g. the University of Nottingham and the University of Melbourne Benchmark datasets (available at http://www.cs.nott.ac.uk/~rxq/data.htm), or the exam timetabling benchmark datasets from the second International Timetabling Competition (available at http://www.cs.qub.ac.uk/itc2007/). However, in this thesis, we focus only on the Toronto dataset because there is a significant number of hyper-heuristic approaches and approaches based on constructive graph colouring heuristics to compare with our approaches.

Instances	**No. of Exams**	**No. of Students**	**Enrolments**	**Density**	**Timeslots**
car91 I	682	16925	56877	0.13	35
car92 I	543	18419	55522	0.14	32
ear83 I	190	1125	8109	0.27	24
hec92 I	81	2823	10632	0.42	18
kfu93 I	461	5349	25113	0.06	20
lse91	381	2726	10918	0.06	18
pur93	2419	30029	120681	0.03	42
rye92	482	11483	45051	0.07	23
sta83 I	139	611	5751	0.14	13
tre92	261	4360	14901	0.18	23
uta92 I	622	21266	58979	0.13	35
ute92	184	2749	11793	0.08	10
yor83 I	181	941	6034	0.29	21

Table 2.2 Details of the Toronto exam timetabling benchmark (Carter et al., 1996; Qu et al., 2009b)

2.2 Three-dimensional Strip Packing Problems (3D-SPP)

In three-dimensional strip packing problems, a number of rectangular boxes, specified by their width, length and height, need to be packed into a three dimensional rectangular container with unlimited length. The objective is to find a *feasible packing* of all the boxes into the container such that the length of the packing is minimised. A feasible packing must obey the following rules: (i) there is no overlap between boxes, and (ii) all boxes should be placed fully within the container. The 3D-SPP can work with only rectangular boxes (3D rectangular strip packing problems) or boxes of any shape (3D irregular strip packing problems). In this research, we focus on the 3D rectangular strip packing problems and the abbreviation 3D-SPP refers to this class. It is clearly the case that to solve the 3D-SPP, boxes should be packed parallel to the edges of the container.

(Wäscher et al., 2006) recently proposed a new classification of cutting and packing problems. According to that classification, the 3D-SPP considered here is referred to as the open dimension problem (ODP), or more precisely as three-dimensional rectangular open dimension problem with one variable dimension (3D-R-ODP).

Problem	**Applications**
2D Strip Packing	• Paper, metal, glass or other sheet material cutting • Pallet loading • Multi-processor scheduling
3D Strip Packing	• Marble, foam, wood or other block cutting • Truck, air cargo load planning • Multi-dimensional limited resource scheduling

Table 2.3 Real-life applications for strip packing problems

Strip packing problems are very popular in the cutting and packing research field. Many practical real-life applications of strip packing can be seen in Table 2.3. While two-dimensional strip packing problems (2D-SPP) have received significant research interest in the literature, the number of publications addressing the 3D-SPP is still relatively low. One reason for this is because the 3D-SPP is particularly difficult due to a large number of constraints. However, it represents a suitable context for work on constructive heuristics. The 3D-SPP is identical to the 2D-SPP when the width (or height) of each box is exactly 1. By solving the 3D-SPP efficiently, we also provide solutions for industrial applications of the 2D-SPP.

For problems concerning loading, there are additional constraints to restrict the placements of the boxes. We list in Table 2.4 the practical constraints that might be applicable for the 3D-SPP. These practical constraints are taken from a publication addressing issues in developing approaches for the three-dimensional container loading problem (3D-CLP) (Bischoff and Ratcliff, 1995).

Constraint	**Description**
Orientation Constraints	Restrict some rotations of boxes from the maximum six rotations.
Box Strength Constraint	Limit the weights put on top of a box. A simple form of this constraint can be: "stack no more than *x* box high".
Handling Constraint	Large boxes might be required to be put on container's floor and heavy boxes need to be positioned below a certain height.
Stability Constraint	Require the bottom surface of a box to be fully supported by either container's floor of top surfaces of other boxes.
Load Stability Constraint	Restrict the movement of boxes during transportation.
Grouping Constraint	Require similar boxes to stay close together.
Multi-Drop Constraint	Position boxes in order of dropping time to reduce the work of loading and unloading a large part of cargo at each drop.
Separation Constraint	Separate boxes containing goods that can severely affect other goods (e.g. chemical stuff and food)
Complete Shipment Constraint	Require boxes belonging to the same shipment (e.g. computer tower and screen) to stay close together.
Shipment Priority Constraint	Some boxes are shipped to customers of higher priority. Those boxes should be positioned close to the container's door.
Arrangement Complexity Constraint	Require packing patterns to be easily handled when unloading.
Weight Distribution Constraint	Require the centre of gravity of the container to be close to the geometrical mid-point of the container's floor. (e.g. The situation often happens if the container is lifted onto a ship).

Table 2.4 Practical constraints for the 3D-SPP (Bischoff and Ratcliff, 1995)

This research investigates the 3D-SPP with two of the most widely studied constraints: *orientation constraint* and *stability constraint*. In particular, experiments will be carried out on a benchmark dataset where the orientation constraint is always applied while the stability constraint is optional. Thus, we divide the problems into two different classes - 3D-SPP with and without the stability constraint.

The strip packing problem is NP-hard as it is a generalisation of the classical bin packing problem. The classical *bin packing problem* involves finding a minimum number of one-dimensional bins of fixed capacity to contain a number of one-dimensional objects with given sizes. The NP-hardness of the bin packing problem can be found in (Garey and Johnson, 1979). Research effort for the 3D-SPP, therefore, has concentrated mostly on designing approximate algorithms which can result in good, but not necessarily optimal solutions within reasonable computing time. Due to the high number of constraints in a 3D-SPP, it is particularly hard to design an effective local move. Most of the successes have been obtained from constructive methods. We review below different approaches in the literature for the 3D-SPP. Several methods were actually proposed for the 2D-SPP. However, such methods in the simplified problem (2D-SPP) can give, or have already given, inspiration for designing approaches for the 3D-SPP. Note that the descriptions of methodologies (e.g. tabu-search, simulated annealing, genetic algorithms, and hyper-heuristics, etc.) are not repeated as they can be found in Section 2.1.

2.2.1 Approximate Algorithms and Constructive Heuristics

Approximate algorithms for the strip packing problems can be separated into *off-line* and *on-line* algorithms. In this research, we concern only off-line algorithms, i.e.

algorithms which have full knowledge of the input. Most of these algorithms are for the two-dimensional problems. However, some of the ideas can still be applicable for the three-dimensional problems. These algorithms can be found in (Csirik and Woeginger, 1996; Lodi et al., 2002). Some of the most popular approximate algorithms and heuristics for strip packing problems are presented as follows.

For the 2D-SPP, Baker et al. (1980) introduced the bottom-left heuristic (BL), which orders the objects according to their areas. The objects were placed at the top right corner of the container and repeatedly pushed down and then left as much as possible. This method was improved by Chazelle (1983) and called bottom left fit (BLF): each object was located at the most bottom place and then pushed left as much as possible. Hopper (2000) presented BLD which was an improved strategy of BL, where the objects were ordered using various criteria (height, width, perimeter and area) and the algorithm selected the best result obtained.

Burke et al. (2004b) proposed a best fit heuristic (BF) that uses a dynamic ordering for the rectangles to be placed. The algorithm went through the available places from the most bottom-left one, and selected for each place the rectangle that best fits in it. If all rectangles cannot fit the considered place, the algorithm ignores that place and considers the next one.

Karabulut and Inceoglu (2004) extended the BLF method to the Deepest BLF (DBLF) method for three-dimensional packing problems. In that method, a box was moved to the deepest available position, and then as far as possible to the bottom, and then as far as possible to the left.

Allen et al. (2011) developed a three-dimensional best fit heuristic (3BF) for the 3D-SPP which is an extension of the best-fit heuristic BF. The 3BF was a constructive approach that gradually arranges boxes into the container. A similar idea with the BF was applied to find boxes that fit as much of the deepest gap in the container as possible. When more than one box may fit the same biggest gap, one of four proposed tie-breaking heuristics was applied. If the gap cannot be filled by any box, it was ignored and the construction process continues with the next deepest gaps.

Some other heuristic approaches based on different ideas of BL and BF were proposed by George and Robinson (1980) and Bischoff and Mariott (1990) for the 3D-SPP. In both cases, a greedy technique was used and loading was carried out by a layer-building (or wall building) method: the given container was filled in vertical layers, which follow one after the other in the longitudinal direction of the container. In this algorithm, each layer was divided into several horizontal strips placed one on top of the other; a single strip was constituted by several boxes placed sequentially and parallel to the container width. Bischoff and Mariott (1990) introduced some heuristics that combine a 2D-packing procedure with the heuristic from (George and Robinson, 1980) to fill the layers. Both articles reported that the solution quality of a layer-building approach depends mainly on the suitable selection of the layer depths.

Bortfeldt and Mack (2007) developed a heuristic derived from a branch-and-bound approach to build layers for the 3D-CLP proposed by Pisinger (2002). Two adaptations to the 3D-SPP were investigated which are the open container and closed container approaches. The open container approach assumes an infinite length container whilst the closed container approach solves the problem by gradually reducing the container’s length until no packing solution is found. The authors showed the success of reusing the best parts of the previous best found solutions in the closed container approach (SPBBL-CC4). Experiments on the 3D-SPP were reported which shows improving average volume utilisation on the SP-BR dataset introduced by Bischoff and Ratcliff (1995).

2.2.2 Meta-heuristics

Bortfeldt and Gehring (1999) presented a TS and a GA algorithm for the 3D-SPP. Both methods were derived from the corresponding algorithms for the 3D-CLP. Two different approaches were investigated for each algorithm. The first approach was based on a single container with a large container length, while the second approach solved a sequence of 3D-CLP instances with decreasing container lengths. Both the TSA and the GA were subjected to parallelisation.

Allen et al. (2011) investigated the integration of 3BF with tabu-search and named the method 3BF+TS. The best solution found by 3BF was used as an input for a tabu-search in the next phase. The first certain number of boxes in the packing solution was kept the same while the remaining boxes were processed using the DBLF. The authors used tabu-search to search for different ordering of input boxes for the DBLF. The local move was to simply swap two boxes in the list of remaining boxes. Results obtained on the SP-BR benchmark without using a stability constraint were competitive in comparison with results obtained in (Bortfeldt and Mack, 2007).

2.2.3 Hyper-heuristics

To our knowledge, hyper-heuristics have not been applied directly to the 3D-SPP. We present here some hyper-heuristic approaches on the 2D-SPP of which the ideas are also applicable for the 3D-SPP.

Terashima-Marín et al. (2006) developed classifier system and messy genetic algorithm hyper-heuristics to solve 2D-regular cutting stock problems. The representation of a problem state remained the same. It consists of information on the number of remaining objects (figures/items) and their size ranges. The heuristic for each problem state, however, was represented by a combination of two different heuristics: one for selecting objects, one for placing the object into a position. The set of heuristics consisted of 8 heuristics for selecting the objects: next-fit, first-fit, best-fit, worst-fit, almost worst fit, first fit decreasing, and Djang and Fitch heuristics. Two heuristics for object placement were bottom-left, and improved-bottom-left. Both the classifier system and the messy genetic algorithm were evaluated in experiments and both produced competitive results (Terashima-Marín et al., 2007). The authors also reported the generalisation on a test phase of generated rules after a training phase. The set of trained rules were applied to test cases and produced very competitive results, better than using the best single heuristic. Terashima-Marín et al. (2008) investigated further on the messy genetic algorithm approach for both 2D-regular and 2D irregular bin-packing problems. Experiments on different parameter settings were conducted. Rules learnt from the training phase could produce some promising results for unseen problems especially when compared with the best results from single heuristics.

Garrido and Riff (2007a, 2007b) also proposed a hill-climbing and a genetic-algorithm-based hyper-heuristic for solving the 2D strip packing problems online. The methods searched for packing strategies for a particular instance instead of having a learning phase and a test phase. The representation of chromosomes had varying length that consists of a number of the following block (a, b, c, d). The meaning of a block was to use a^{th} greedy heuristic, use the b^{th} ordering heuristic, and apply the dth rotation heuristic to locate c objects. The crossover operator was not allowed to use a crossing point in the middle of any block. The four most competitive single heuristics were chosen to form the set of low-level heuristics including: best-fit, bottom-left-fill, a recursive heuristic (HR) that locates the objects capable of covering large area on the bottom left corner (Zhang et al., 2006), and a heuristic BFDH* (a variation of BFDH (Mumford-Valenzuela et al., 2003)) that locates an object with the longest width (from all rotations). The results obtained on several testing instances were very competitive; they even outperformed the state-of-the-art specialised algorithms for such benchmark instances.

2.2.4 Summary of the SP-BR Dataset

In this thesis, the work on the 3D-SPP will be evaluated on a popular dataset (SP-BR) in the literature. This dataset consists of 1000 problem instances which were initially introduced as benchmarks for the three-dimensional container loading problem by Bischoff and Ratcliff (1995). These instances can be downloaded from the OR-library (http://people.brunel.ac.uk/~masterjjb/jeb/info.html). The dataset has been widely used for the 3D-SPP by assuming an indefinite container length. Orientation constraints for all box types are specified clearly in these instances. In the literature, publications address this dataset either with or without an addition of a stability constraint. The 1000 three-dimensional strip packing instances are divided into 10 test cases, namely SP-BR01 to SP-BR10, each with 100 instances. The characteristics of these test cases vary from weakly heterogeneous SP-BR01 to strongly heterogeneous SP-BR10.

We also verify our approach on large instances SP-BR01-XL to SP-BR10-XL which are generated based on the original instances. These instances can be downloaded from http://www.cs.nott.ac.uk/~rxq/benchmarks.htm. The generating process is implemented exactly the same as in (Bortfeldt and Mack, 2007). The container width, height and the orientation constraints for all box types remain the same. However, the number of boxes per type is increased by multiplying it by a factor $1000/N_{box}$, where N_{box} represents the total number of boxes in the original instance. As the resulting numbers of boxes are truncated to integer numbers, we may need to increase the number of boxes of the last type so that the total number of boxes reaches 1000.

The objective of each instance is to find a feasible arrangement that requires a minimum container length. However, for each test case of 100 instances, an average of the required container lengths gives little knowledge on the performance of an approach. A more informative objective is used which is to maximise the volume utilisation (VU), calculated as follows:

$$volumeUtilisation = optimalLength\ /\ actualLength \tag{2.1}$$

The *actualLength* is the minimum container length obtained. The *optimalLength* can be estimated by using a lower bound length shown in Equation 2.2:

$$lowerBoundLength = \left\lfloor \left(\sum_{b \in B} volume(b) \right) / (containerWidth \times containerHeight) \right\rfloor \tag{2.2}$$

where B is the set of all boxes.

2.3 Chapter Summary

This chapter presented the problem descriptions, the heuristic approaches developed over the last decades and benchmark datasets for both the ETP and the 3D-SPP. The complexity of the approaches grows along with the increasing power of computing hardware. The mostly practical early heuristics were to constructively build solutions whilst in more recent research, attention has been paid to more powerful meta-heuristic techniques such as hill climbing, tabu-search, simulated annealing, and evolutionary algorithms, etc. There has been a tendency to hybridise different approaches which can significantly improve performance. Recently, hyper-heuristics have emerged as an alternative direction to produce methodologies of higher generality in timetabling as well as in cutting and packing fields. The performance on problem benchmarks was relatively well studied by many approaches. In the subsequent chapters, several approaches working on simple constructive heuristics for both the ETP and the 3D-SPP are proposed and compared against some approaches listed in this chapter.

Chapter 3 An Enhanced Weighted Graph Model for the ETP

This chapter presents our first contribution in this research. Most of the approaches based on constructive heuristics for the ETP employ a number of traditional graph colouring heuristics, e.g. largest degree, largest weighted degree, saturation degree, etc. These heuristics use some information associated with vertices and edges to measure the difficulty level of a vertex if it is deferred until later in the colouring process. Our main aim is to design new constructive heuristics which may estimate the difficulty level more precisely, possibly at the cost of longer running time. Thus, we introduce in this chapter an enhanced weighted graph model which adds more information into each vertex or edge. The model is designed to be adaptable to a variety of exam timetabling scenarios. Within this model, some novel vertex- and colour-selection heuristics arise naturally. We also introduce a possible improvement for the exam timetabling, i.e. a partitioning technique. With the aim of evaluating the model's techniques on the Toronto benchmark instances in Chapter 4, some of the model’s features are set to be suitable for this benchmark only.

3.1 The Graph Colouring Model

We use the following notations to define a common graph. A graph *G* consists of a set *V* of vertices and a set *Edge* of edges. A colouring of a graph *G* is a function *c* which assigns a colour *c(v)* to each vertex $v \in V$.

Conventionally, exam timetabling problems can be solved using the graph colouring model. Each vertex in the graph corresponds to an exam to be scheduled and each colour corresponds to a different timeslot. Accordingly, assigning colour *c* to vertex *v* represents that the exam corresponding to *v* is scheduled in the timeslot corresponding to *c*.

3.2 Description of an Enhanced Weighted Graph Model

The enhanced weighted graph model described in this section incorporates more problem specific information at the input and maintains even more information pertaining to the partial colouring during the solution process as comparing to the existing approaches. In this weighted graph model, each vertex and each edge are *weighted* with several attributes, some of them hold information from the problem instance and others hold information that helps guide the colouring process.

In our enhanced model, associated with each vertex is the set of students who must take that exam. Two vertices are joined by an edge, and are said to be adjacent or neighbours, if it is undesirable to schedule the corresponding exams in the same timeslot. Each edge carries information that tells us how undesirable it would be for the corresponding exams to be scheduled in the same timeslot or in timeslots near each other. In particular, each edge has two attributes: the set of students taking both exams (*intersection subset*); and a positive integer indicating the conflict severity if the exams are scheduled in the same timeslot. This second attribute is tied to the size of the intersection subset.

To illustrate our model, suppose there are four available timeslots, 0, 1, 2, and 3 for five exams, E_1, E_2, E_3, E_4, and E_5. The set of students taking each of the exams is as follows:

E_1: {a, b, c, d, e, f, g, h, i, j}
E_2: {k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}
E_3: {a, e, k}
E_4: {b, c, d, x, y, z}
E_5: {a, c, e, g, i, j}

Each edge in the graph shown in Figure 3.1 has the subset of students enrolled in both exams corresponding to the endpoints of that edge.

In general it may be undesirable to assign the same timeslot (colour) to a given pair of adjacent exams for a variety of reasons. For this example, however, we consider the popular condition which is two vertices are adjacent only if there is at least one student taking both exams.

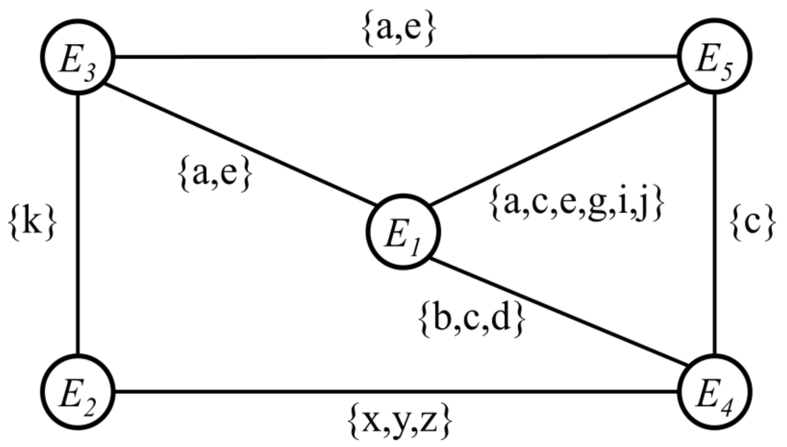


Figure 3.1 Student intersections for pairs of exams

For our example, we may set the conflict severity equal to 1, 5, or 25, according to the size of the intersection subset. In particular, we set the conflict severity to 1 if the intersection size is 1 or 2, to 5 if the intersection size is 3 or 4, and to 25 if the intersection size is 5 or greater (see Figure 3.2). We emphasize that these thresholds for conflict severity are arbitrarily chosen here. If a conflict-free timetable is a requirement, as it is in the Toronto problems then all conflict severities can simply be set to one since all conflicts are regarded as equally bad.

Of course, as mentioned, there will be many situations in which the conflict severity depends on other factors. In these situations, an edge might exist even when it corresponds to an empty intersection of students.

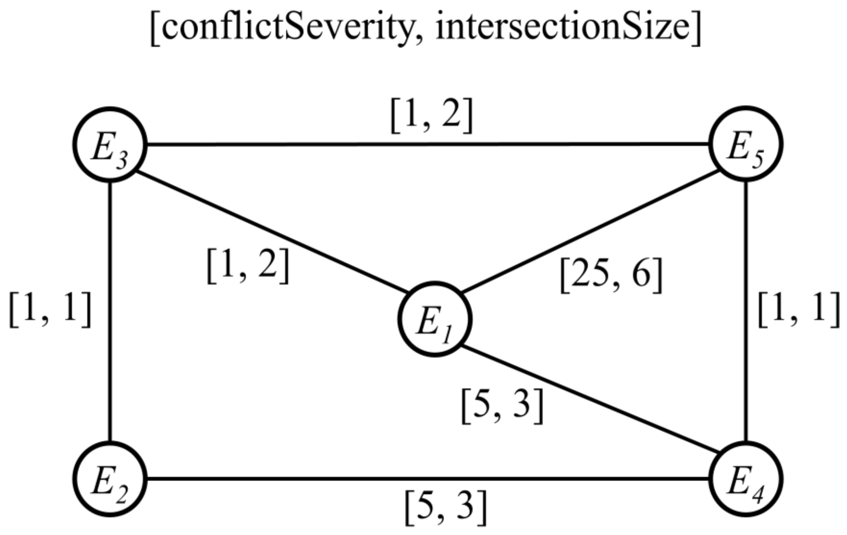


Figure 3.2 Additional edge attributes

The proximity penalty of assigning colours t_i and t_j to the endpoints of an edge is a function of how close t_i and t_j are and the size of the intersection c_{ij}. Section 2.1 presents the function to calculate the proximity penalty for the Toronto dataset. Our implementation uses the same function for comparison purposes with the Toronto benchmark results. However, if the timeslots are specified by a day, or a start time and duration, then our colour attributes can easily be modified to allow for the appropriate change in the proximity evaluation function.

The overall objective is to produce colourings (timetables) with minimum total conflict (zero may be required) and minimum total proximity penalty.

Knowing the conflict severity and size of the intersection for each edge makes it straightforward to keep track of the two kinds of penalties as the colouring progresses. When a vertex gets coloured *t*, that colour becomes less desirable (or forbidden) to its neighbours, as do colours in proximity with colour *t* (i.e. timeslots close together with the considered timeslot *t*).

Our model keeps track of these two kinds of colour undesirability as follows. Each vertex *v* has a *colour-penalties vector* that indicates the undesirability of assigning each colour to that vertex with respect to conflict penalty and proximity penalty. That is, the component of the colour penalties vector corresponding to colour *t* has two values, one is the conflict penalty incurred if *v* is coloured *t*, and the other is the resulting proximity penalty.

Using our example and a simplified proximity function, we illustrate how the colour-penalties vectors change as the graph is coloured. Suppose that any two colours t_i and t_j of the colours 0, 1, 2, and 3 are *within proximity* if they differ by 1, then the proximity penalty incurred when the colours of the endpoints of an edge differ by 1 equals the intersection size. Suppose further that the colour-penalties vectors for all of the vertices are initialised with [0, 0] for all of their colour components. Figure 3.3 shows the result of colouring vertex E_1 with colour 1.

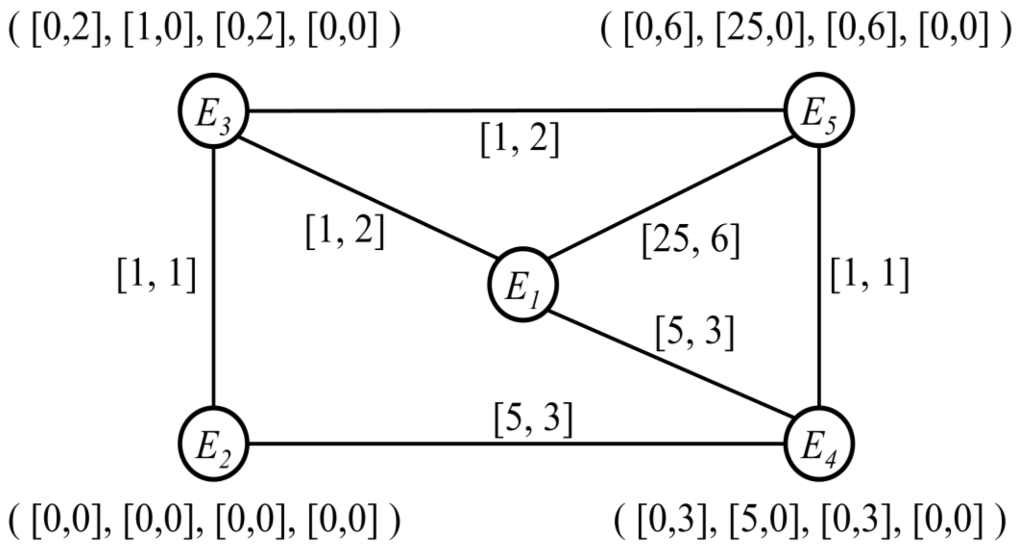


Figure 3.3 Colour-penalties vectors after E_1 is coloured 1

As each vertex is coloured, its adjacent vertices' colour-penalties vectors are updated. The ease with which we are able to keep track of both hard and soft constraints as the colouring progresses creates new opportunities for the use of more sophisticated heuristics tied to this readily accessible information.

3.2.1 The Basic Constructive Approach

We use a similar constructive approach as used for the classical graph colouring problem. It consists of two steps - select a vertex and then colour that vertex. These two steps are repeated until all vertices are coloured. Notice that the model will easily accommodate more computation-intensive approaches involving backtracking and local improvement. However, we keep our focus on simple constructive heuristics. Thus, we choose to concentrate on producing fast and essentially one-pass colourings. Here, one-pass colouring is understood as obtaining a solution without applying backtracking or local improvement during the construction process.

3.2.2 Summary of the Model Features and Parameters

We employ the same notations as used in the ETP formulation in Section 2.1. An edge $e = (i, j)$ connecting exams E_i and E_j exists only if $c_{ij} > 0$. The set *Edge* consists of all the edges in the graph. In preparation for the discussion of heuristics, we list the key features and parameters on which the heuristics are based. The two edge attributes, conflict severity and intersection size, give rise to two different versions of the traditional concept of weighted degree of a vertex.

- *Conflict severity* (of an edge) – a measure of how undesirable it is to assign the same colour to both endpoints of the edge (see examples in Figures 3.1 and 3.2). In general, this would depend on several factors, and it could be set interactively by the end-user. However, for the Toronto dataset, all conflict severities are set to 1 as every conflict is of equal weight and prohibited.

$$cs(e) = \begin{cases} 1 & if\ c_{ij} > 0\ and\ t_i = t_j \\ 0 & otherwise \end{cases}$$

- *Intersection size* (of an edge) – the size of the intersection of the two sets corresponding to the endpoints of the edge. In exam timetabling, this is simply the number of students taking both exams (see examples in Figures 3.1 and 3.2).

$$is(e) = c_{ij}$$

- *Average intersection size* - the average of the intersection sizes of all edges in a graph.

$$ais(Edge) = \frac{\sum_{\forall e \in Edge} is(e)}{countEdge(Edge)}$$

- *Conflict degree* (of a vertex) – the sum of the conflict severities of the edges incident on the vertex.

$$cd(v) = \sum\nolimits_{e \in IncidentEdge(v)} cs(e)$$

- *Intersect degree* (of a vertex) – the sum of the intersection sizes of the edges incident on the vertex.

$$id(v) = \sum\nolimits_{e \in IncidentEdge(v)} is(e)$$

- *Bad-conflict edge* – an edge whose conflict severity exceeds a specified *threshold* value. If a conflict-free timetable (i.e., a feasible colouring) is required, then this threshold is set to zero, e.g. the Toronto problem instances.

$$bce(e) = \begin{cases} 1 & if\ cs(e) > T_{ce} \\ 0 & otherwise \end{cases}$$

- *Bad-intersect edge* – an edge whose intersection size exceeds a specified threshold $T_{ie} = ais(Edge) * ie$, where *ie* is a multiplier parameter.

$$bie(e) = \begin{cases} 1 & if\ is(e) > T_{ie} \\ 0 & otherwise \end{cases}$$

- *Conflict penalty* (for the colour assignment of a vertex) – a measure of how undesirable it is to assign that colour to the vertex. This will depend on the colour assignments of the vertex's neighbours and the conflict severities of the relevant edges, but it could also depend on other factors (e.g., professor, room, or equipment constraints).

- *Proximity* (of two colours) – a measure of how close together (in the case of exam timetabling) or spread apart (for course timetabling) the two colours are. This is often a secondary objective to optimise in exam timetabling and is typically referred to as a *soft constraint*. For the Toronto instances, the proximity is measured as follows:

$$d(t_i, t_j) = |t_i - t_j|$$

- *Proximity penalty* (for the colour assignment of two endpoints of an edge) – the proximity penalty resulting from the colour assignments of those two vertices. For the Toronto instances, the following function is applied to measure that proximity penalty.

$$P(t_i, t_j) = \begin{cases} 2^5 / 2^{d(t_i, t_j)} & if\ 1 \le d(t_i, t_j) \le 5 \\ 0 & otherwise \end{cases}$$

- *Colour-penalties vector* (of a vertex) – indicates, for each colour, the conflict penalty and proximity penalty of assigning that colour to the vertex. When a vertex is coloured, the colour-penalties vector of each of that vertex's neighbours must be updated accordingly.

- *Bad-conflict colour* (for a vertex) – a colour whose conflict penalty for that vertex exceeds a specified threshold T_{cc} (for the Toronto instances, $T_{cc} = 0$ since feasible colourings are required).

- *Bad-proximity colour* (for a vertex) – a colour whose proximity penalty for that vertex exceeds a specified threshold $T_{pc} = ais(Edge) * ev * pc$, where $ev = (62 - 114)/(x(x-1))$ is the expected value of the proximity weight and *pc* is a multiplier parameter (see Appendix A for a derivation of *ev* and an explanation of its use).

- *Bad-colour* (for a vertex) – either a bad-conflict colour or a bad-proximity colour for that vertex.

The thresholds for badness are easily adaptable to the requirements of the problem, and, in a decision support system, they could be specified by the end-user interactively. Chapter 4 provides some studies of the effect that the values of the thresholds have on the quality of the solutions.

3.3 Primitive Heuristics

As addressed, vertex selection and colour selection are the two key components of the simple, constructive algorithm. Our implementation uses nine 'primitive' heuristics for selecting the next vertex to be coloured and four to select a colour for that vertex.

3.3.1 Nine Primitive Vertex-Selection Heuristics

The colouring strategies are based on the classical and intuitive idea that the most troublesome vertices should be coloured first; see Table 2.1 for some commonly used heuristics. We use variations of those, and we introduce some new ones that focus more on the number of bad edges and the number of bad colours. Some of these new heuristics rely on the information kept in each vertex's colour-penalties vector, while others use information tied to the edges incident on each vertex. The primitive heuristics on which our vertex selectors are based are:

0. *Maximum number of bad-conflict edges to uncoloured neighbours* – vertices having the most bad-conflict edges among their incident edges to uncoloured neighbours. For the Toronto dataset, this heuristic represents the typical largest uncoloured degree heuristic.

1. *Maximum number of bad-conflict colours* – vertices having the most bad-conflict colours. For the Toronto dataset, this heuristic reduces to largest saturation degree (see Figure 3.4 for an example).

2. *Maximum number of bad-proximity colours* – vertices having the most bad-proximity colours (see Figure 3.4 for an example).

3. *Maximum number of bad colours* – a consolidation of heuristics 1 and 2; a bad colour is one whose conflict penalty or whose proximity penalty exceeds its respective threshold (see Figure 3.4 for an example).

4. *Maximum conflict sum* – vertices with the largest sum of their conflict colour penalties. For the Toronto dataset, heuristic 4 is similar to heuristic 1 (saturation degree) because all conflict severities are set to 1.

5. *Maximum proximity sum* – vertices with the largest sum of their proximity colour penalties (see Figure 3.4 for an example).

6. *Maximum conflict degree to uncoloured neighbours* – vertices whose incident edges to uncoloured neighbours have the largest sum of the conflict severities. For the Toronto dataset, heuristic 6 is identical to heuristic 0 (largest uncoloured degree) because all conflict severities are set to 1.

7. *Maximum number of bad-intersect edges to uncoloured neighbours* – vertices having the most bad-intersect edges among their incident edges to uncoloured neighbours (see Figure 3.5 for an example).

8. *Maximum intersect degree to uncoloured neighbours* – vertices whose incident edges to uncoloured neighbours have the largest sum of the intersection sizes (see Figure 3.5 for an example).

The motivation to introduce some new heuristics here originates from the intuition of measuring the difficulty of a vertex. For example, we observe that heuristic 7 may be better at evaluating the difficulty of a vertex than its sum counterpart, heuristic 8. To illustrate, suppose that the edge weights in Figure 3.5 represent intersection size and that all neighbours of vertices *v*1 and *v*2 are uncoloured. Then heuristic 8 would select *v*1, whereas, for any bad-intersect-edge threshold greater than one, heuristic 7 would select *v*2, which seems to be more difficult. A similar observation can be made for heuristic 2 versus heuristic 5 (see Figure 3.4). Experimental justifications on these situations are provided in Chapter 4.

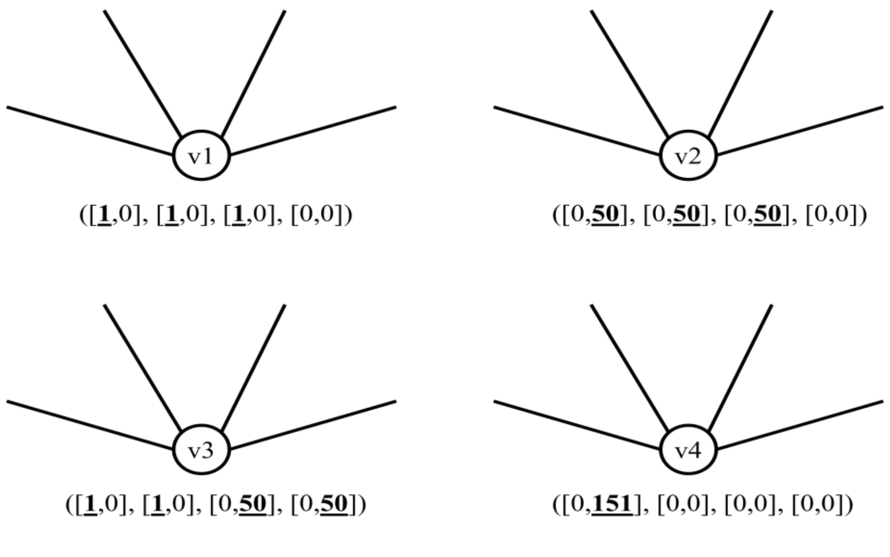


Figure 3.4 An example of heuristics that select difficult vertices based on the information retained in the colour-penalties vector. Heuristic 1 will select *v*1, heuristic 2 will select *v*2, heuristic 3 will select *v3*, and heuristic 5 will select *v*4, if $T_{pc} < 50$

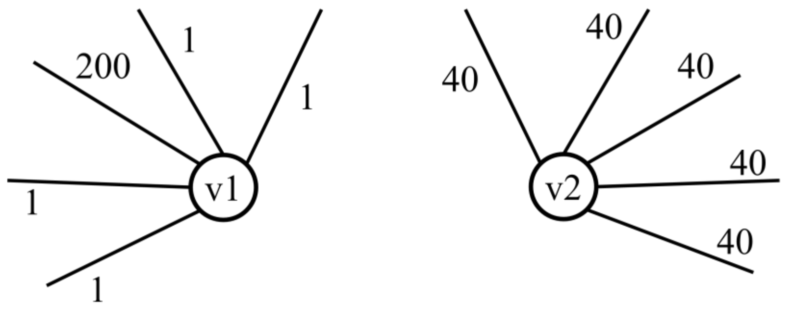


Figure 3.5 Heuristic 8 will select *v*1 while heuristic 7 will select *v*2 if $T_{ie} < 40$

3.3.2 Four Primitive Colour-Selection Heuristics

Given a vertex *v* that has been selected, the primitive heuristics that we use to choose a colour for *v* are:

0. *Minimum conflict penalty* – a colour that has the minimum conflict penalty for vertex *v*.
1. *Minimum proximity penalty* – a colour that has the minimum proximity penalty for vertex *v*.
2. *Least bad for neighbours with respect to conflict penalty* – a colour which when assigned to *v* causes the fewest good-to-bad conflict penalty switches for the uncoloured neighbours of *v*.
3. *Least bad for neighbours with respect to proximity penalty* – a colour which when assigned to *v* causes the fewest good-to-bad proximity penalty switches for the uncoloured neighbours of *v*.

Figure 3.6 shows an example to illustrate the use of different colour-selection heuristics. The example presents a particular exam timetabling scenario with the following assumptions:

- There are totally 4 colours (timeslots), i.e. 0, 1, 2, 3.
- Any 2 colours t_i and t_j are within proximity if they differ by 1. This leads to the proximity penalty, incurred when the colours of the 2 endpoints of an edge differ by 1, equals the intersection size of the edge.
- Colour-penalties vectors of E_1, E_2 represent the current situation of the partial colouring. Edge attributes are as shown in Figure 3.6.
- Thresholds $T_{cc} = 0$ (conflict is not allowed) and $T_{pc} = 8$.
- E_1, E_2 are uncoloured while E_3, E_4, E_5 received colours 1, 0, 2, respectively.
- Vertex E_1 is being selected for colouring.

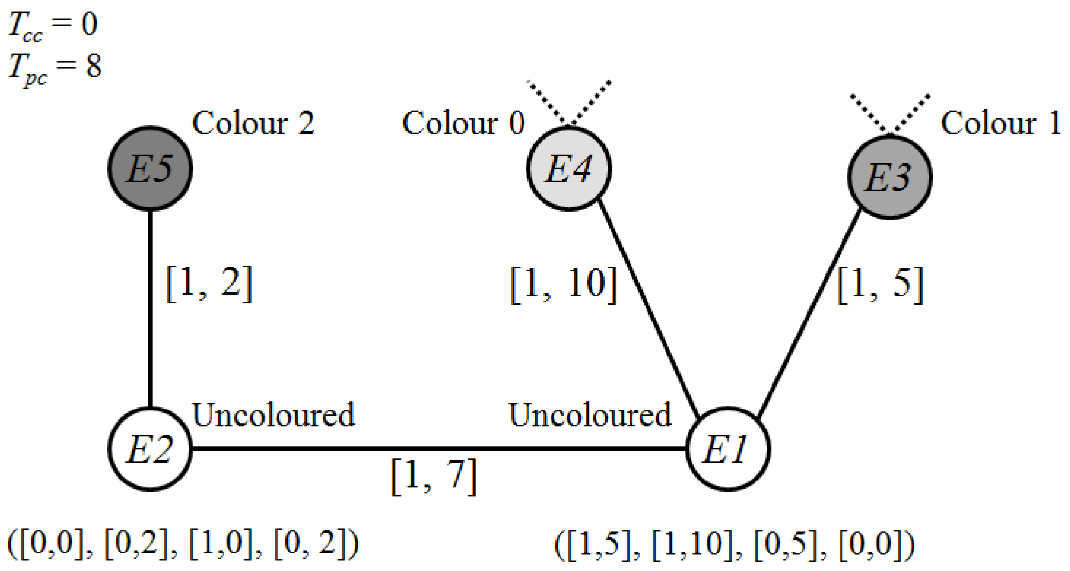


Figure 3.6 An example to illustrate the use of different colour-selection heuristics

Vertex E_1 cannot receive colours 0 or 1 due to the prohibition of conflict in this exam timetabling scenario. We list below the decisions made by the four colour-selection heuristics:

- Heuristic 0 will select either colour 2 or 3 for E_1 as both colours cause no conflict penalty.
- Heuristic 1 will select colour 3 for E_1 as its resulting proximity penalty is 0 compared to a proximity penalty of 5 if E_1 receives colour 2.
- Heuristic 2 will select colour 2 which causes no good-to-bad conflict penalty switch for E_1's uncoloured neighbours because in the current partial solution, colour 2 was already a bad-conflict colour for E_2. The selection of colour 3 in this case will increase one good-to-bad conflict penalty switch in E_1's uncoloured neighbours (i.e. E_2).
- Heuristic 3 will select colour 3 which causes no good-to-bad proximity penalty switches for E_1's uncoloured neighbours. The selection of colour 2 here will cause two good-to-bad proximity penalty switches on colours 1 and 3 since the proximity penalties of these two colours will increase from 2 to 9 (i.e. greater than T_{pc} with the value of 8).

3.4 Switching Selectors in the Middle of a Colouring

Another feature of this model is the ability to switch from one heuristic to another at various stages of the colouring. Including this feature was motivated by the general observation that the effectiveness of a heuristic is likely to change as the colouring progresses. The primitive vertex-selection heuristic 1 is perhaps the simplest illustration of this behaviour. As we mentioned earlier, this heuristic is essentially the traditional saturation degree, which has proven to be among the most preferred heuristics for classical graph colouring. However, applying heuristic 1 at the very early stages of a colouring will produce a huge number of ties. Moreover, early in a colouring, the only vertices with any bad-conflict colours will tend to be those few that have neighbours that have already been coloured. Thus, until several vertices are coloured, the order in which they are selected will tend toward a simple breadth-first order and not be an effective predictor of the difficult-to-colour vertices.

Accordingly, the primitive heuristic used early in the colouring process should be based on the weights of incident edges (e.g., heuristic 0). Then, after a designated number of vertices have been selected and coloured, we switch to heuristic 1 when it is more likely to be a stronger predictor of the difficulty of a vertex.

3.5 Vertex Partitioning

Our final innovation involves a preprocessing step that partitions the vertex set and allows us to reduce the amount of computation without incurring additional conflict penalties. The preprocessing is based on the following simple observation. If v is a vertex with degree less than k, and v initially has k colours available, then v can safely be left until last to colour, since it will always have at least one non-conflict colour available, independent of how its neighbours are coloured and of how heavy the edge-weights are between v and its neighbours.

The preprocessing uses an iterative partitioning algorithm that places all vertices whose colouring can be done last into the easiest-to-colour subset, say s_1. Next, for each vertex in s_1, we calculate a reduced (quasi-) degree of each of its neighbours and put all vertices whose reduced degree is less than the number of colours available into the next-easiest-to-colour subset, s_2. Again, as long as a vertex in s_2 is coloured before any of its neighbours in s_1, it can safely be left uncoloured until its other neighbours are coloured. The process continues until no additional vertices can be removed from the 'hardest' subset and the vertices in that last subset of the partition must be coloured first using the specified selection criteria. The pseudo-code for the vertex partitioning can be found in Algorithm 3.1 where each *VS* represents a vertex subset found over the process. The process starts by initialising a degree vector d of which each element represents the number of a vertex's neighbours. Then, the process repeatedly construct each easy subset by comparing the degree (or quasi-degree) $d[v]$ of an unassigned vertex v with the total number of available colours T, and adds the vertex into the current easy subset if $d[v] < T$. After all easy-to-colour vertices are found, we update the degree vector by reducing the degree of those vertices' unassigned neighbours. Until no easy-to-colour vertex can be found, the remaining set of vertices represents the hardest-to-colour subset.

As long as the subsets are done in order (last to first), vertices in all subsets except for the hardest one can be selected arbitrarily with no possibility of incurring a conflict penalty. One simply chooses an available colour, whose existence is guaranteed by the construction. Thus, in a fairly sparse graph, computation can be considerably reduced.

Another potential advantage to this partitioning strategy is that the vertex-selection process after the hardest subset has been coloured can be based solely on proximity considerations. However, this strategy is applicable only for the uncapacitated exam timetabling problem without the room constraint.

Algorithm 3.1 Vertex Partitioning

```
initialise the degree vector $d$
$i := 0$
**repeat**
    $i := i + 1$
    $VS_i := \emptyset$
    **for** ($v \in V$) **do**
        **if** ($d[v] < T$) **then**
            $V := V \setminus \{v\}$
            $VS_i := VS_i \cup \{v\}$
        **end if**
    **end for**
    **if** ($VS_i \neq \emptyset$) **then**
        **for** ($v \in VS_i$) **do**
            **for** (($v, u) \in Edge$ and $u \in V$) **do**
                $d[u] := d[u] - 1$
            **end for**
        **end for**
    **end if**
**until** $VS_i = \emptyset$
$VS_i := V$
```

3.6 Chapter Summary

The weighted graph model presented in this chapter represents a new direction for further research on simple constructive heuristics for exam timetabling problems. Some features of this model in our current implementation can show its applicability in various exam timetabling scenarios (not limited only to the Toronto benchmark). The model can handle pre-coloured vertices, that is, exams that must be assigned to certain timeslots. Furthermore, if certain timeslots are forbidden for a particular exam, then this can easily be handled by setting an initial nonzero penalty for the relevant colour. As we noted earlier, each colour, which represents a timeslot, can have attributes associated with fairly general information, like start time, duration and/or finish time instead of having only a single timeslot value as for the Toronto benchmark problems. Although it is not a subject of this research, this model can be easily extended or integrated with other techniques to develop more advanced and powerful algorithms. Based on this model, we further investigate different approaches in Chapter 4 to better utilise the simple heuristics to construct exam timetables. However, this model and the novel associated heuristics will not be used in Chapter 6 within a hyper-heuristic context. The reason is because our hyper-heuristic only targets heuristics which are fast enough instead of those that require a certain number of trials to find a good parameter settings (i.e. bad proximity colour parameter *pc* or bad intersect edge parameter *ie*).

Chapter 4 Combinations of Heuristics for the ETP

In Chapter 3, we presented a weighted graph model and the primitive constructive heuristics associated with it. Algorithms using single heuristics are easy to implement, but in many cases, may not produce highly competitive results. It is often better to take into account different factors than to reply on only one factor. In this chapter, we focus on using combinations of a number of primitive heuristics, instead of using a single heuristic, to better select vertices and colours. Two different strategies to combine heuristics are investigated, namely sequential and linear combinations. Both strategies can be applied on vertex- and colour-selection heuristics; however, we focus mainly on the combinations of vertex-selection heuristics in this thesis. Whilst sequential combinations represent the use of a sequence of heuristics as tie-breakers, linear combinations use weights to define specific roles that each simple heuristic contributes to the ordering strategy. We include specific explanations for the design of our strategies and present the experimental results on 12 instances of the Toronto dataset. Some competitive results are obtained using linear combinations when compared with other constructive methods.

4.1 Sequential Combination Strategy (SCS)

This way of combination allows the use of any number of primitive heuristics to form compound vertex selectors and compound colour selectors. A compound vertex selector starts with one of the nine primitive vertex-selection heuristics listed in Section 3.3.1. As discussed before, the ETP is tackled in this thesis using the approximate algorithm which repeatedly incurs a vertex-selection heuristic and a colour-selection heuristic. A vertex-selection heuristic will use a particular strategy to identify which vertex is likely to cause trouble if deferred until later. Typically, there will be several vertices identified as the most difficult with respect to that heuristic. This subset of vertices is then narrowed down by applying a second primitive heuristic, and so on. Thus, a compound vertex selector consists of a sequence of primitive heuristics, where all but the first one in the sequence is regarded as a tie-breaker for the ones before it. Once the subset of vertices is pared down by the combination of heuristics, a vertex is chosen from the subset (typically the first one in the list). Compound colour selectors are similarly constructed from the four primitive colour-selection heuristics listed in Section 3.3.2.

The pseudo-codes for the SCS for selecting a vertex and the SCS for selecting a colour are presented in Algorithms 4.1 and 4.2. These two strategies are basically the same except that a vertex is selected based on the maximum difficulty while a colour is selected based on the minimum trouble (caused to the vertex itself or to the vertex's neighbours). Both processes iterate through the lists of primitive heuristics in a corresponding compound selector *H*. *V* and *C* can be understood as the sets of candidate vertices and colours, respectively while *rV* and *rC* represent the interim vertex and colour sets which store the remaining most difficult vertices and most troublesome colours, respectively after applying one more heuristic in the compound selectors. *V* and *C* are updated accordingly to *rV* and *rC* at each iteration, respectively. The processes typically reduce the size of *V* and *C* in each iteration. At the end, if there are still more than one candidate vertices or colours, we simply pick the first one in the sets. Such vertex or colour should be selected as early as possible according to this SCS.

Algorithm 4.1 Sequential Combination Strategy for Selecting a Vertex

```
Input:  a compound vertex selector $H$ (with vertex-selection heuristics)
        a list of all uncoloured vertices $V$

$i$ := 1
while ($i$ < $H$.size) do
    max_difficulty := 0
    for ($v \in V$) do
        $d$ := measure the difficulty of vertex $v$ by using heuristic $H_i$
        if ($d$ > max_difficulty) then
            max_difficulty := $d$
            $rV$ := {$v$}
        else
            if ($d$ = max_difficulty) then
                $rV$ := $rV \cup$ {$v$}
            end if
        end if
    end for
    $V$ := $rV$
    $i$ := $i$ + 1
end while
return $V_1$
```

We use the following two groups of compound vertex selectors:

*vs*1: 0 7 8 1 2 5 | 1 0 2 5 7 8 | 2 5 7 8

*vs*2: 0 7 8 3 5 | 3 0 7 8 2 5 | 2 5 7 8

The numbers refer to the primitive vertex-selection heuristics introduced in Section 3.3.1. The separation for the set of vertices is illustrated in Figure 4.1.

Algorithm 4.2 Sequential Combination Strategy for Selecting a Colour

```
Input:  a compound colour selector $H$ (with colour-selection heuristics)
        a list of all available colours $C$

$i$ := 1
while (i < $H$.size) do
    min_trouble := $+\infty$
    for ($c \in C$) do
        $d$ := measure the trouble caused by colour $c$ by using heuristic $H_i$
        if ($d$ < min_trouble) then
            min_trouble := $d$
            $rC$ := $\{c\}$
        else
            if ($d$ = min_trouble) then
                $rC$ := $rC \cup \{c\}$
            end if
        end if
    end for
    $C$ := $rC$
    $i$ := $i$ + 1
end while
return $C_1$
```

The vertical lines separate the three compound selectors that form each group. The first compound selector in a group is applied to the hardest subset until a certain number of vertices (the *switching point*) have been selected and coloured. Then, the second compound selector is applied to the rest of the hardest subset. Finally, the third selector, which consists of the four proximity-related primitive heuristics, is applied to the remaining (non-hard) vertices.

$$\underbrace{\text{Early Stage | Remaining Stage}}_{\text{Vertices in the Hardest Subset}}\text{ | Dealing with Easy Subsets}$$

Figure 4.1 Vertex set separation

The reason for designing *vs*1 and *vs*2 are presented as follows. At the early stage (before the switching point), we colour a vertex based on a heuristic using the information retained in its incident edges. This is due to that not many vertices adjacent to the considering vertex has been coloured, thus information in the colour-penalties vector may not represents the true difficulty of a vertex. For the vertices in the later part of the hardest subset (after the switching point), we use heuristics based on the same idea with saturation degree. Therefore, heuristics 1 and 3 took the leading roles. While heuristic 1 is identical to the saturation degree for the Toronto dataset, heuristic 3 can be considered as an advanced version of the saturation degree because it also takes into account the bad-proximity colours. We always use heuristic 8 as the tie-breaker for heuristic 7 because they represent different ideas to measure a vertex's difficulty by using the same information (see Figure 3.4 for an example). Similarly, heuristic 5 is always used as the tie-breaker for heuristic 2 because they operate on the same information (see Figure 3.5 for an example). Although the main components of these vertex selectors may include up to six heuristics, an observation in Section 4.3 will show that there are, in many cases, redundant heuristics.

We used the following two groups of two compound colour selectors:

*cs*1: 0 1 2 3 | 0 1 3

*cs*2: 0 2 3 1 | 0 3 1

The first compound selector in each group was applied to the entire subset of hardest-to-colour vertices, and the second one was applied to the rest of the vertices (see Section 3.3.2 for the primitive colour-selection heuristics). The first colour-selection heuristic is always heuristic 0 to make sure that the selected colour causes no conflict if it is possible. Then, two colour selectors use two different strategies to order the heuristics in two opposite ways. While *cs*2 prefers the colours that cause less conflict and proximity to the vertex's neighbours first, *cs*1 prefers colours that cause minimum proximity penalty for the vertex itself first. For easy subsets, the same strategies are applied for *cs*1 and *cs*2, but colour-selection heuristic 2 is unnecessary as for the vertex's neighbours, there is always at least one available colour.

4.2 Experimental Results for the SCS

All experiments in Section 4 are conducted on a PC Pentium 4, 3.4 GHz processor with 2GB memory. We test SCS on the Toronto benchmark with the following settings:

$$\textit{switching point} = 1..(\textit{nExam} \times 20\%);$$

$$pc = 1..120 \text{ with an increment of } 0.1$$

$$ie = 1..2 \text{ with an increment of } 1$$

This section presents the results of applying the SCS on the Toronto benchmarks. We set the number of colours equal to the number of timeslots in the Toronto dataset. Table 4.1 presents our best results obtained from several parameter settings. The partitioning technique appears to improve solution quality most of the time. Except for the “sta83 I” problem instance, all results in column 2 of the table were produced using the partitioning pre-processing.

The Settings column gives the values of the switching point, the multipliers - *pc* (the parameter for bad-proximity colours) and *ie* (the parameter for bad-intersect edges), and indicates the vertex and colour selectors used to produce the given result. The last column gives the best results reported in the literature published in (Qu et al., 2009b) which used different constructive methods.

The results from Table 4.1 demonstrate that, for vertex selection, 10 of the 12 best results were achieved using *vs*2. Similarly, using *cs*1 obtains better results than using *cs*2 in 10 out of 12 instances. Changing threshold values for badness and changing the switch point between the first and second compound vertex selector clearly affects the performance of the algorithm.

Problem	**SCS's best results**	**Settings switching point | *pc* | *ie* | *vs* | *cs***	**Best reported**
car91 I	5.22	29 | 52.1 | 1 | *vs*2 | *cs*1	4.97
car92 I	4.40	41 | 66.8 | 2 | *vs*2 | *cs*1	4.32
ear83 I	39.28	36 | 46.4 | 1 or 2 | *vs*2 | *cs*1	36.16
hec92 I	12.35	16 | 4.9 | 1 or 2 | *vs*1 | *cs*1	10.8
kfu93 I	18.08	32 | 45.2 | 1 or 2 | *vs*2 | *cs*1	14.0
lse91	12.05	11 | 58.6 | 1 or 2 | *vs*2 | *cs*1	10.5
rye92	10.21	17 | 51.5 | 2 | *vs*2 | *cs*1	7.3
sta83 I	163.05	5 | 18.3 | 1 | *vs*2 | *cs*2	158.19
tre92	8.62	6 | 79.8 | 2 | *vs*2 | *cs*1	8.38
uta92 I	3.62	38 | 28.9 | 1 or 2 | *vs*1 | *cs*1	3.36
ute92	30.60	36 | 65.6 | 1 or 2 | *vs*2 | *cs*2	25.8
yor83 I	42.05	10 | 120 | 2 | *vs*2 | *cs*1	40.66

Table 4.1 SCS's results with the corresponding settings for the Toronto benchmark. PC refers to the parameter for bad-proximity colours. IE refers to the parameter for bad-intersect edges

The totals for the proximity penalty obtained from this approach are on average 13% worse than the best ones reported; however, this approach still holds promise, particularly in view of the fact that it is, at the moment, an one-pass algorithm without any backtracking or local improvement. The best results reported in the last column of Table 4.1 were by different meta-heuristic approaches cited in the literature. A notable point is that no single algorithm outperformed others on all problems tested here. In the next section, a more generalised approach is presented which shows improving performance.

4.3 Linear Combination Strategy (LCS)

The sequential combination strategy uses sequences of primitives as tiebreakers to form compound vertex selectors. However, we observe in an experiment that many tiebreakers in the sequences are often unnecessary when the set of candidates are narrowed down to a single vertex rapidly. Table 4.2 presents the performance of the compound vertex selector *vs*1 (see Section 4.1) with a particular set of parameters (*pc=50*, *ie=2*, *switchingPoint=1*), and compound colour selector: *cs*1 (see Section 4.1) applied to the hardest vertex subset of the 12 Toronto problem instances.

	car91 I (507)	**car92 I (392)**	**ear83 I (157)**	**hec92 I (70)**	**kfu93 I (196)**	**lse91 (124)**	**rye92 (189)**	**sta83 I (78)**	**tre92 (193)**	**uta92 I (458)**	**ute92 (89)**	**yor83 I (176)**
Primitive 0	112	102	40	30	66	33	74	11	44	95	30	46
Primitive 7	341	259	103	32	117	85	102	22	132	296	52	117
Primitive 8	38	18	6	2	2	1	3	0	5	38	2	4
Primitive 1	16	13	8	6	11	5	10	13	12	29	5	9
Primitive 2	0	0	0	0	0	0	0	0	0	0	0	0
Primitive 5	0	0	0	0	0	0	0	0	0	0	0	0
Unnarrowed	0	0	0	0	0	0	0	32	0	0	0	0

Table 4.2 The depth level of primitive heuristics called for the hardest vertex subset using *vs*1. Values in parentheses denote the size of the hardest vertex subsets

Observation results in Table 4.2 show the number of times the set of the most troublesome vertices is narrowed down to one vertex after applying a particular heuristic in the sequence. In the majority of cases, this happens only after the first two heuristics are used. As a result, the remaining primitive heuristics in the sequence play no role in the vertex selection. The problem instance sta83 I is unique in that there are 32 times in which the set of troublesome vertices is not narrowed down to one vertex after applying six primitive heuristics in the sequence.

This observation motivates a new strategy that combines primitive heuristics more effectively. The compound vertex selectors are now *(weighted) linear combinations* of the primitive vertex-selection heuristics.

To identify the most troublesome vertices, we apply a heuristic evaluation function *f* to each of the uncoloured vertices, and the vertex having the largest evaluation value is chosen. In the first implementation, our function *f* is a linear combination of seven characteristics (parameters) tied to the partial colouring of the weighted graph. Due to the use of the Toronto dataset for experiments, where non-conflict timetables are compulsory, vertex-selection heuristics 4 and 6 share the same ordering strategies with vertex-selection heuristics 1 and 0, respectively. Therefore, for each uncoloured vertex *v*,

$$f(v) = a_0x_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_5x_5 + a_7x_7 + a_8x_8, \qquad (4.1)$$

where a_i are nonnegative weights and

x_0 = number of bad-conflict edges to *v*'s uncoloured neighbours.

x_1 = number of bad-conflict colours counted from the colour-penalties vector of *v*.

x_2 = number of bad-proximity colours counted from the colour-penalties vector of *v*.

x_3 = number of bad colours counted from the colour-penalties vector of *v*.

x_5 = sum of the proximity penalties over all colours stored in the colour-penalties vector of *v*.

x_7 = number of bad-intersect edges to *v*'s uncoloured neighbours.

x_8 = intersect degree to *v*'s uncoloured neighbours.

This linear-combination approach is adaptable to colour selection as well, although in this research, we focus only on the vertex-selection process. The pseudo-code for the LCS for selecting a vertex is presented in Algorithm 4.3. This strategy simply applies the function *f* in Equation 4.1 to all vertices and selects the vertex that returns the maximum value of *f* as the most difficult vertex. As mentioned, this vertex should be coloured as early as possible due to the trouble it might cause if deferred until later.

Algorithm 4.3 Linear Combination Strategy for Selecting a Vertex

```
Input:    a list of all uncoloured vertices

max_difficulty := 0;
$v_{most\_difficult}$ := 0;
for ($v \in V$) do
    $d$ := $f(v)$; // see Equation 4.1
    if ($d$ > max_difficulty) then
        max_difficulty := $d$;
        $v_{most\_difficult}$ := $v$;
    end if
end for
return $v_{most\_difficult}$
```

4.4 Justification of Heuristic Selections and Weight Settings for the LCS

The main task in designing linear combinations of primitive heuristics is to set the *weight vector* [a_0, a_1, a_2, a_3, a_5, a_7, a_8]. This section focuses on describing one approach to tackle this task and justify the decisions by experiments on the 12 Toronto benchmark instances. In all experiments in this section, we always use the vertex partitioning step described in Section 3.5. The compound colour selector *cs*1: 0 1 2 3 | 0 1 3 is applied as it showed its efficiency in the SCS experiments. The multiplier parameters for the bad-proximity colour threshold and the bad-intersect edge threshold are applied for all experiments in this section and are varied as follows:

- *pc* = 5 to 75 with increments of 2.
- *ie* = 0.5 to 5 with increments of 0.5.

The results obtained from the algorithm give the average proximity penalty over all students. We include in each experiment the best results from other constructive methods drawn from the survey paper (Qu et al., 2009b). That is for the purpose of observing the progress of our method when new heuristics are added into the linear combinations.

The new vertex-selection heuristics proposed in Chapter 3 are based on the same idea with traditional heuristics in the literature. In the following experiments, we step-by-step observe the effectiveness of each heuristic in combination with others. However, the idea is to create linear combinations based on the idea of combining saturation degree and largest weighted degree, which has been very popular in ETP research. The other heuristics may act as tie-breakers.

4.4.1 The Effectiveness of using Heuristic 3 – MaxBadColours

In seven characteristics deduced from the vertex-selection heuristics, four of them are based on the information retained in the vertices' colour-penalties vectors. Accordingly, we classify them as *colour-penalties-based vertex-selection heuristics*.

$x1$ = number of bad-conflict colours counted from the colour-penalties vector of v (saturation degree).

$x2$ = number of bad-proximity colours counted from the colour-penalties vector of v.

*x*3 = number of bad colours counted from the colour-penalties vector of *v*.

*x*5 = sum of the proximity penalties over all colours stored in the colour-penalties vector of *v*.

In this group, we claim that heuristic *3* should play a role in linear combinations because it is essentially a consolidation of heuristics 1 and 2. Heuristic 2 is analogous to the saturation degree (heuristic 1) but is based on proximity instead of conflict.

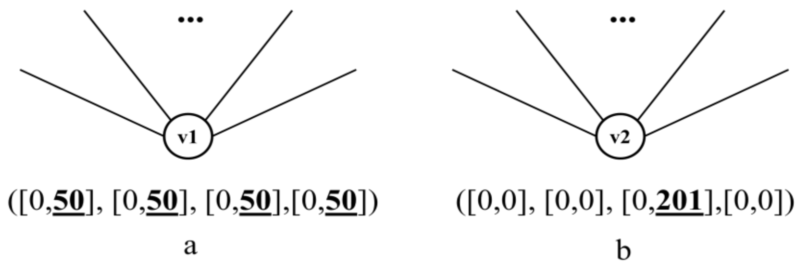


Figure 4.2 Colour-penalties vectors of an example where heuristic 2 is probably better than heuristic 5. Heuristic 2 would select vertex *v*1 and heuristic 5 would select vertex *v*2, if the bad-intersect edges threshold $0 \le T_{pc} < 50$

In Section 3.3.1, we also suggested that heuristic 2 may be better at evaluating the trouble level of a vertex than its sum counterpart, heuristic 5. An extreme example in Figure 4.2 shows the colour-penalties vectors of two vertices *v*1 and *v*2. Suppose that there are four available colours for each vertex (represented by the four component-pairs in each colour-penalties vector), the first value in each component represents the conflict penalty and the second represents the proximity penalty. If we set a bad-proximity-colour threshold $0 \le T_{pc} < 50$, heuristic 2 would return four bad-proximity colours (underlined in Figure 4.2a) from *v*1's evaluation and as a result, favour *v*1 over vertex *v*2, which has only one bad-proximity colour (underlined in Figure 4.2b).

On the other hand, heuristic 5 always selects vertex $v2$ with the proximity penalty sum of 201 before $v1$ whose proximity penalty sum equals 200.

The experimental results shown in Table 4.3 reinforce these claims. The experiment used the following groups of linear combinations:

$vs1$*:* $x3$ | $x2$

$vs2$*:* $x1$ | $x2$

$vs3$*:* $x2$ | $x2$

$vs4$*:* $x5$ | $x2$

$vs5$*:* $x1 + .00001x2$ | $x2$

$vs6$*:* $x1 + .00001x5$ | $x2$

These groups are chosen to compare heuristics that measure a vertex's difficulty based on information retained in the colour-penalty vectors, i.e. $x1$, $x2$, $x3$, $x5$. We would also compare heuristic 3 ($vs1$) with saturation degree with tie-breakers (i.e., $vs5$ and $vs6$).

The vertex selector to the left of the vertical line is applied to the hardest vertex subset (first), and the one to the right is then applied to all the easy subsets. In $vs5$ and $vs6$, we set a small enough weight (*.00001*) so that heuristics 2 and 5 will act as tiebreakers for heuristic 1. For the easy subsets, we always choose heuristic 2 to select the most troublesome vertices since it concerns only proximity and is favoured over its sum counterpart 5. $vs1$ which uses vertex-selection heuristic 3 could obtain better results in 9 out of 12 instances.

Problem	***vs*1**	***vs*2**	***vs*3**	***vs*4**	***vs*5**	***vs*6**	**Best reported**
car91 I	**5.23**	5.44	infeasible	infeasible	5.34	5.34	4.97
car92 I	**4.47**	4.85	infeasible	infeasible	4.66	4.66	4.32
ear83 I	**38.05**	44.1	infeasible	infeasible	38.99	39.88	36.16
hec92 I	**12.28**	infeasible	infeasible	infeasible	12.69	13.82	10.8
kfu93 I	infeasible	infeasible	infeasible	infeasible	**18.68**	infeasible	14.0
lse91	**12.23**	12.81	infeasible	infeasible	12.91	13.33	10.5
rye92	**10.6**	11.51	infeasible	infeasible	11.02	11.49	7.3
sta83 I	168.63	164.94	infeasible	infeasible	**163.04**	165.11	158.19
tre92	**8.68**	9.89	infeasible	infeasible	9.52	9.14	8.38
uta92 I	**3.45**	3.75	infeasible	infeasible	3.55	3.6	3.36
ute92	**29.39**	32.75	infeasible	infeasible	30.21	31.2	25.8
yor83 I	infeasible	infeasible	infeasible	infeasible	**42.2**	infeasible	40.66

Table 4.3 Comparisons on different groups of linear combinations of coloured-penalties-based heuristics. Bold font indicates the best results obtained among the linear combinations. Parameter settings are presented in Section 4.4

4.4.2 Linear Combinations of Heuristics 3 and 7

The remaining three primitive vertex-selection heuristics are classified as *edge-based vertex-selection heuristics* since they are based on the (static) information retained in each vertex's adjacent edges to determine its difficulty.

$x0$ = number of bad-conflict edges to uncoloured neighbours (degree).

$x7$ = number of bad-intersect edges to uncoloured neighbours.

$x8$ = intersect degree to uncoloured neighbours (weighted degree).

This section mainly compares three edge-based vertex-selection heuristics, i.e. heuristics 0, 7 and 8. We demonstrate an example of extreme cases to see how these three heuristics work. Then, we provide experimental results on the Toronto dataset to understand how these heuristics perform in combination with heuristic 3 (max bad-colours).

For the Toronto instances, heuristics 0 and 8 represent the traditional largest uncoloured degree and largest weighted degree, respectively. Heuristic 7, introduced within the weighted graph model in the previous chapter, involves a threshold to determine when an intersection size of an edge is considered 'bad'. We suggested in Section 3.3.1 that counting bad-intersect edges of a vertex (heuristic 7) may be better at evaluating the difficulty of the vertex than adding up all the intersection sizes of the incident edges (as carried out by heuristic 8). The example shown in Figure 4.3 illustrates this claim.

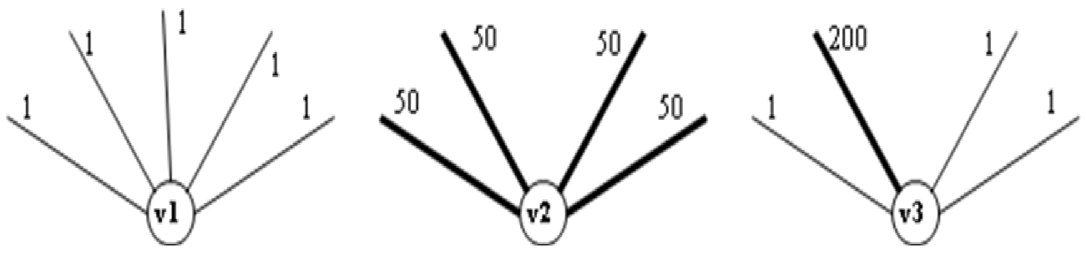


Figure 4.3 Vertex-selection strategies for edge-based primitive heuristics 0, 7, 8. Heuristic 0 would select vertex *v*1, heuristic 8 would select vertex *v*3, and heuristic 7 would select heuristic *v*2 if the bad-intersect edges threshold $1 \leq T_{ie} < 50$

We provide an experiment using the following groups of vertex selectors:

*vs*1*: x3* | *x*2

*vs*2*: 10000x3* + *x*7 | *10000x*2 + *x*7

*vs*3*: 10000x*3 + *x*8 | *10000x*2 + *x*8

*vs*4*: 100x*3 + *10x*7 | *100x*2 + *10x*7

*vs*5*: 10x*3 + *100x*7 | *10x*2 + *100x*7

The very large weight of *10000* in the linear combinations *vs*2 and *vs*3 is to set heuristics 7 and 8 as tiebreakers. Linear combinations *vs*4 and *vs*5 test different weighting scheme, i.e. (100, 10), (10,100) for the combinations between heuristics 3 and 7. These are the two main heuristics forming the core of linear combinations in this chapter. As before, the linear combinations for the easy subsets always use heuristic 2 instead of heuristic 3 since heuristic 2 is based only on proximity.

The results shown in Table 4.4 support our claim that heuristic 7 may also be better than heuristic 8 when used in linear combinations with heuristic 3. In addition, we observe that the higher weight setting for *x*3 than *x*7 is likely to produce better timetables.

Problem	***vs*1**	***vs*2**	***vs*3**	***vs*4**	***vs*5**	**Best reported**
car91 I	5.23	5.21	5.19	**5.09**	infeasible	4.97
car92 I	4.47	**4.29**	4.39	4.32	infeasible	4.32
ear83 I	38.05	37.65	38.71	**36.7**	39.93	36.16
hec92 I	**12.28**	12.52	12.38	12.52	12.93	10.8
kfu93 I	infeasible	**16.93**	18.21	**16.93**	18.4	14.0
lse91	12.23	**11.46**	11.49	**11.46**	12.98	10.5
rye92	10.6	9.83	10.04	**9.74**	infeasible	7.3
sta83 I	168.63	160.26	162.6	160.26	**159.61**	158.19
tre92	8.68	8.57	8.58	**8.5**	9.44	8.38
uta92 I	3.45	3.47	3.54	**3.44**	infeasible	3.36
ute92	29.39	**28.41**	29.46	**28.41**	29.44	25.8
yor83 I	infeasible	41.1	41.61	**40.74**	40.84	40.66

Table 4.4 Comparisons on different groups of linear combinations between colour-penalties-based and edge-based vertex-selection heuristics. Bold font indicates the best results obtained among the linear combinations. Parameter settings are presented in Section 4.4

4.4.3 The Effect of Including Heuristic 0 in Linear Combinations

Heuristic 0 applied to the Toronto problems is essentially the traditional largest uncoloured degree. As we suggested earlier regarding the dynamic nature of a heuristic's effectiveness as the colouring progresses, applying heuristic 1 (saturation degree) at the beginning of the process has no distinguishing effect, whereas heuristic 0 (largest degree) does. We investigate this behaviour using the following groups of vertex selectors:

*vs*1*: 100x*3 + *10x*7 | *100x*2 + *10x*7

*vs*2*: 100x*3 + *10x*7 + *0.1x*0 | *100x*2 + *10x*7

*vs*3*: 100x*3 + *10x*7 + *1x*0 | *100x*2 + *10x*7

*vs*4*: 100x*3 + *10x*7 + *10x*0 | *100x*2 + *10x*7

*vs*5*: 100x*3 + *10x*7 + *0.1x*0 | *100x*3 + *10x*7 | *100x*2 + *10x*7*, switching point = 10*

With *vs*1, *vs*2, *vs*3 and *vs*4, we examine the linear combinations of the core combination (i.e. heuristic 3 and heuristic 7) with different weight settings for heuristic 0, i.e. (0, 0.1, 1 and 10). We also examine the effectiveness of adding heuristic 0 only in the early part of colouring (before a switching point as in *vs*5).

As before, the vertical line separates the linear combinations for different stages of a colouring. For each group, the last linear combination is applied to the easy vertex subsets. It always replaces heuristic 3 by its proximity analogue heuristic 2. For vertex selector *vs*5, the first and second combinations are applied to the hardest subset before and after the specified switching point. For each of the selectors from *vs*1 to *vs*4, the first linear combination is applied to the entire hardest subset.

The results shown in Table 4.5 show a preference to use a switching point. Vertex selector *vs*5 obtains the best solutions in seven problem instances. Notice that the inclusion of heuristic 0 was particularly effective only when being applied before the switching point.

Problem	*vs*1	*vs*2	*vs*3	*vs*4	*vs*5	Best reported
car91 I	**5.09**	5.15	5.18	5.41	**5.09**	4.97
car92 I	4.32	4.33	**4.31**	4.49	4.32	4.32
ear83 I	**36.7**	37.38	37.38	41.37	**36.7**	36.16
hec92 I	12.52	12.27	12.47	12.62	**12.09**	10.8
kfu93 I	16.93	**16.58**	17.66	infeasible	16.98	14.0
lse91	**11.46**	11.57	11.5	11.61	**11.46**	10.5
rye92	**9.74**	9.83	10.39	12.18	**9.74**	7.3
sta83 I	160.26	159.37	160.41	163.83	**158.95**	158.19
tre92	8.5	**8.47**	8.51	8.98	8.5	8.38
uta92 I	**3.44**	**3.44**	**3.44**	infeasible	**3.44**	3.36
ute92	**28.41**	28.83	28.83	30.16	28.68	25.8
yor83 I	40.74	40.74	**40.67**	41.43	40.74	40.66

Table 4.5 Comparisons on different groups of linear combinations with regards to the inclusion of heuristic 0 and switching points. Bold font indicates the best results obtained among the linear combinations. Parameter settings are presented in Section 4.4

4.4.4 Using Heuristic 5 and/or 8 as Tiebreakers

Given the effectiveness of using heuristic 0 as a tiebreaker, it was natural to consider the effect of using heuristics 5 and 8 in a similar way. Table 4.6 shows the results for the following groups of selectors:

*vs*1*: 100x*3 + *10x*7 | *100x*2 + *10x*7

*vs*2*: 100x*3 + *10x*7 + *.00001x*5 | *100x*2 + *10x*7 + *.00001x*5

*vs*3*: 100x*3 + *10x*7 + *.00001x*8 | *100x*2 + *10x*7 + *.00001x*8

*vs*4*: 100x*3 + *10x*7 + *.00001x*8 | *100x*3 + *10x*7 + *.00001x*5 | *100x*2 + *10x*7 + .00001x*5*, switching point = 10*

where *vs*1, *vs*2 and *vs*3 compare the linear combinations of heuristic 3 and heuristic 7 with a tie-breaker (i.e. heuristic 5 or heuristic 8). Vertex selector *vs*4 has a switching point where before the switching point, the edge-based heuristic 8 is the tie-breaker while after the switching point, the colour-penalties-based heuristic 5 is the tie-breaker. As before, the reason for that difference between two sides of the switching point is because the colour-penalties-based heuristics may not provide a precise measure of difficulty at early stage of colouring. Similar to other experiments, heuristic 2 replaces heuristic 3 for the easy subsets.

The comparison of *vs*1 with three selectors that involve no, one or both heuristics 5 and 8 as shown in Table 4.6 shows no clear advantage towards using any of them.

Problem	***vs*1**	***vs*2**	***vs*3**	***vs*4**	**Best reported**
car91 I	**5.09**	5.12	5.19	5.12	4.97
car92 I	4.32	**4.29**	4.32	**4.29**	4.32
ear83 I	36.7	36.73	**36.55**	36.73	36.16
hec92 I	12.52	12.52	12.52	**12.38**	10.8
kfu93 I	**16.93**	17.11	17.12	17.17	14.0
lse91	11.46	11.49	11.54	**11.4**	10.5
rye92	**9.74**	9.77	9.79	9.77	7.3
sta83 I	160.26	**160.07**	160.53	**160.07**	158.19
tre92	8.5	8.56	**8.39**	8.56	8.38
uta92 I	3.44	**3.38**	3.44	**3.38**	3.36
ute92	**28.41**	29.34	29.24	29.34	25.8
yor83 I	40.74	**40.38**	40.74	**40.38**	40.66

Table 4.6 Comparisons on different groups of linear combinations with regards to the inclusion of heuristic 5 and/or heuristic 8 acting as tiebreakers. Bold font indicates the best results obtained among the linear combinations. Parameter settings are presented in Section 4.4

4.4.5 Improving the Linear Combination Strategy

Based on the experimental results described in the previous sections, we focus here on one specific vertex selector. The vertex selector we present here consists of a linear combination of heuristics 3, 5, and 7 applied to the entire hardest vertex subset, and a linear combination of heuristic 2, 5, and 7 applied to all of the easy subsets. We use the same weight vector, $[a, a_5, a_7]$, for both linear combinations. Thus, our vertex selector has the form

$$ax_3 + a_5x_5 + a_7x_7 \mid ax_2 + a_5x_5 + a_7x_7$$

The characteristics of the four primitive heuristics are repeated below.

*x*2 = number of bad-proximity colours counted from the colour-penalties vector of *v*.

*x*3 = number of bad colours counted from the colour-penalties vector of *v*.

*x*5 = sum of the proximity penalties over all colours stored in the colour-penalties vector of *v*.

*x*7 = number of bad-intersect edges to uncoloured neighbours.

4.5 Experimental Results for the LCS

We test the vertex selector in Section 4.4.5 using each of the eight sets of values for [a, a_5, a_7] shown in Table 4.7:

(1000, 0.00001, 1)	(100, 0.00001, 15)
(1000, 0, 1)	(1000, 0, 15)
(100, 0.00001, 10)	(100, 0.00001, 50)
(1000, 0, 10)	(1000, 0, 50)

Table 4.7 Sets of weight settings

Also, for each linear combination, we vary the two threshold parameters, *pc* and *ie*, over a large range of values as follows:

- *pc* = 5 to 90 with increments of 0.1 (851 different values).
- *ie* = 0.5 to 5.5 with increments of 0.1 (51 different values).

The compound colour selector *cs*1: 0 1 2 3 | 0 1 3, introduced in Section 4.1, is used here.

4.5.1 The Effect of Vertex Partitioning as a Pre-processing Step

All the results (up to this point) in this chapter were produced by assuming the use of the vertex partitioning pre-processing. Here, we investigate the effect of using vertex partitioning with the improved vertex selector. Table 4.8 shows the best results obtained for each problem instance with and without vertex partitioning. Vertex partitioning produced better results for seven problem instances (bold font) and was equal or only slightly inferior for the remaining five problem instances. These results show that vertex partitioning is a promising approach to improve exam timetabling solutions.

Problem	**Our best results with partitioning**	**Our best results without partitioning**
car91 I	5.05	5.03
car92 I	**4.22**	4.24
ear83 I	36.07	36.06
hec92 I	**11.71**	12.12
kfu93 I	16.02	16.02
lse91	**11.15**	11.28
rye92	9.47	9.42
sta83 I	**158.86**	158.96
tre92	**8.37**	8.39
uta92 I	**3.37**	3.38
ute92	28.18	27.99
yor83 I	**39.53**	39.73

Table 4.8 Comparison on groups of linear combinations with and without using the pre-processing step of vertex partitioning

4.5.2 The Overall Performance of Linear Combinations of Primitive Vertex-selection Heuristics

Table 4.9 lists the lowest proximity penalty obtained for a feasible solution on each of the 12 Toronto problem instances. It indicates the use of vertex partitioning (2 options), the weight vectors (8 options), and the values of the threshold multipliers *pc* (851 options) and *ie* (51 options) used to obtain each best result for our vertex selector. For each problem instance, the total number of sets of parameters examined is 694416. The average execution time for one set of parameters for each problem instance is reported.

Problem	Settings (used vertex partitioning, weight vector for heuristics [3(2), 5, 7], threshold multipliers)	Avg. Exec. Time per set of parameters (seconds)	LCS Best Results	SCS Results	Best constructive reported	Best cited
car91 I	No, [100, .00001, 50], *pc*=79, *ie*=3.3	156.5	5.03	5.22	4.97	4.5
car92 I	Yes, [1000, .00001, 1], *pc*=61.5, *ie*=5	106.61	**4.22**	4.40	4.32	3.93
ear83 I	No, [1000, .00001, 1], *pc*=43, *ie*=0.7	7.22	**36.06**	39.28	36.16	29.3
hec92 I	Yes, [100, 0, 15], *pc*=35.4, *ie*=0.5	1.38	11.71	12.35	10.8	9.2
kfu93 I	Yes, [100, .00001, 50], *pc*=86.8, *ie*=3.1	14.89	16.02	19.04	14.0	13.0
lse91	Yes, [1000, .00001, 1], *pc*=30, *ie*=4.5	8.49	11.15	12.05	10.5	9.6
rye92	No, [100, 0, 50], *pc*=31, *ie*=3.7	26.08	9.42	10.21	7.3	6.8
sta83 I	Yes, [100, 0, 10], *pc*=11.1, *ie*=0.7	1.52	158.86	163.05	158.19	157.0
tre92	Yes, [100, .00001, 15], *pc*=50, *ie*=2.5	10.33	**8.37**	8.62	8.38	7.9
uta92 I	Yes, [1000, 0, 1], *pc*=78.6, *ie*=3.6	119.08	3.37	3.62	3.36	3.14
ute92	No, [100, 0, 10], *pc*=6.1, *ie*=4.8	1.54	27.99	30.60	25.8	24.4
yor83 I	Yes, [100, .00001, 50], *pc*=39.6, *ie*=1.7	5.6	**39.53**	42.05	40.66	36.2

Table 4.9 Summary of the results obtained from using linear combinations of primitive vertex-selection heuristics compared with other approaches. The best constructive reported and the best cited results are taken from (Qu et al., 2009b)

The comparison of the LCS and the SCS demonstrates a significant superiority of the linear combination strategy. We also see that LCS can improve on the best results reported for constructive algorithms on four out of the twelve problem instances (bold font).

The best results from the LCS in Table 4.9 are obtained by conducting an exhaustive search from a large range of parameter settings. Therefore, it requires a significantly large execution time. This experiment rather demonstrates the possibility of finding better timetables within the search space of linear combinations of the primitive vertex-selection heuristics. Further research on the landscape of the parameter search space represents a promising future research direction.

4.6 Chapter Summary

We find the results of using combination strategies encouraging, given that they involve a one-pass construction without backtracking. The results suggest that linear combinations of primitive heuristics supersede their use sequentially as tie-breakers. Moreover, the tie-breaker effect could be alternatively achieved by setting one weight much larger than another. Weight settings in linear combinations also allow different heuristics to play a more equal role in the selection process, which has the potential to lead to more effective heuristics and is worthy of further investigation. We provided a set of experiments to justify the decisions to select heuristics and set weights to efficiently solve the exam timetabling problem. Suggestions about the effectiveness of the newly proposed constructive heuristics compared to the conventional heuristics are also justified by experiments. The vertex partitioning technique in the weighted graph model is shown to be a useful technique.

Chapters 3 and 4 focus on tailor-made approaches that exploit particular problem specific information. Chapter 6 will focus on raising the level of generality of search methodologies by developing a hyper-heuristic with the capability of solving different problems. To pave the way for Chapter 6, Chapter 5 presents a heuristic strategy which is suitable within a hyper-heuristic context for one of the investigated problems (i.e. the 3D-SPP).

Chapter 5 An Extended Best-Fit Strategy for the 3D-SPP

Chapters 3 and 4 provide our contributions in designing a weighted graph model and introducing new constructive heuristics which may be more precise in measuring the difficulty level of vertices at the cost of longer running time. We then propose two different strategies to combine heuristics to improve further the decision to select vertices and colours. This chapter will be mainly on improving a constructive approach for the 3D-SPP while in Chapter 6, both the 3D-SPP and the ETP will be investigated in applications of a hyper-heuristic. This chapter presents an extended version of the 'best-fit' strategy for three-dimensional strip packing problems. From that, we introduce several novel constructive heuristics based on human packing intuition. Some of them are designed to be suitable for the 3D-SPP with different sets of constraints (in this research, we consider orientation and stability constraints). The heuristics provide more, possibly better decision options concerning different problem solving situations. Within the extended strategy, we also propose an adjustment technique to move the box around its position and a procedure to group similar boxes together to increase the compactness of packing. The main aim of these extensions is to improve the effectiveness of the best-fit strategy, especially for the use within a hyper-heuristic context.

5.1 The Three-Dimensional Best-Fit Strategy (3BF)

As described in Section 2.2, the 3D-SPP concerns packing a number of rectangular boxes, specified by their width, length and height, into a three dimensional rectangular container with unlimited length. The objective is to find a non-overlapping packing of all the boxes into the container such that the length of the packing is minimised. Recently, Allen et al. (2011) developed the three-dimensional best-fit strategy by extending the best-fit strategy (2BF) for the 2D stock cutting problem (Burke et al., 2004b). The idea of the 3BF is to search dynamically from the list of unpacked boxes in order to find the most suitable boxes to fill the available *gaps*. The aim of the 3BF is to gradually pack boxes as compactly as possible into the deepest gaps inside the container. For reference purposes within this research, we choose the deepest-bottom-left corner of a box or a container as its *origin* and define the coordinate terminology, as shown in Figure 5.1. A gap is defined as an unobstructed space within the container originated from a *starting point*, i.e. its closest coordinate to the container origin. The packing of a selected box into a gap is understood as placing the origin of the box onto the starting point of the gap. The 3BF heuristic repeatedly makes the following two decisions until all boxes are accommodated:

- *Gap-finding*: finds a number of the deepest gaps inside the container.
- *Box-placement*: finds boxes that can fill the largest gaps. If there are ties, one of the tie-breaking heuristics will be used to further reduce the options. If no box can fit into the deepest gaps, those gaps are ignored and the gap-finding process will continue at the next deepest level. Note that, the ignored gaps will not be revisited.

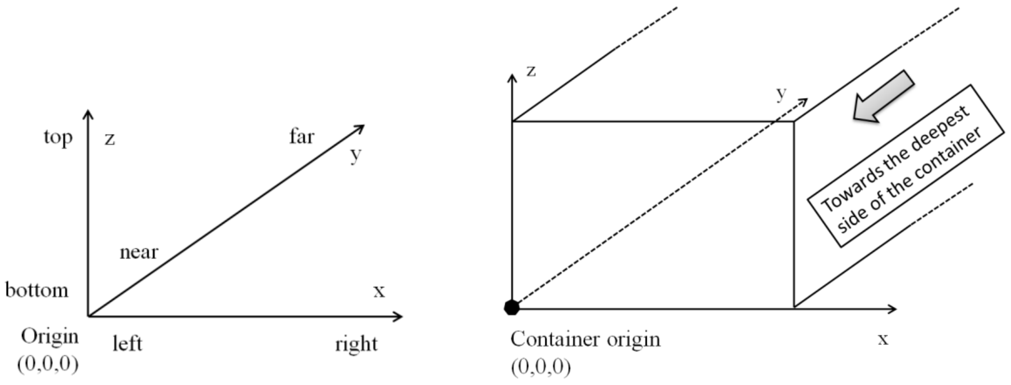


Figure 5.1 Coordinate terminology

An advantage of filling the deepest gaps is that when a box is considered to be placed into the container, we only need to check collisions with other boxes on the *x*- and *z*-axis. This is due to the boxes already placed not having the nearest *y* coordinate greater than the nearest *y* coordinate of the box being considered.

5.2 Extensions for the 3BF Strategy

Within the hyper-heuristic context, the main drawback of the 3BF is that the process of searching for boxes offers little diversification due to the largest gap filling criterion. We provide an alternative approach in this section to compensate for that drawback. In addition, we also present our approaches in finding the deepest gaps, breaking box selection ties and improving the compactness of the packing.

5.2.1 Gap-finding Process

In the first decision of gap-finding, different strategies may be used, ranging from finding all the deepest gaps to only the deepest-bottom-left gap. For the 3D-SPP without the stability constraint (3D-SPP-NS), these gap-finding methods generate significantly different packing outcomes. This is due to the fact that, at a decision point, we may place a box into the deepest gap near the top of the container without concerning the support of its bottom surface. From the implementation point of view, finding all the deepest gaps requires large computational cost. We devise the following fast gap-finding process which shows from experimental experience almost as good results as finding all the deepest gaps.

In this fast gap-finding process, a '*row(y, z)*' is defined as all coordinates with a fixed length (*y*) and a fixed height (*z*). As the sizes of a box are integers, only a limited number of coordinates may lie on a row. The process considers those coordinates which are not being contained by other boxes on the deepest-bottom *row(y, z)* as the candidate gaps. In addition, to reduce the number of candidate gaps, only coordinates on a row with *x* being 0 or equivalent to the *x*-coordinate of *adjacent boxes* already accommodated are considered. Adjacent boxes are those with the furthest $y \geq y$-coordinate, and the highest $z \geq z$-coordinate of the row being considered. Figure 5.2a presents an example of the deepest-bottom row and two candidate gaps (i.e. starting points marked as 1 and 2). If no box fits into any gap in that row, the gap-finding process will find gaps in the next deepest-bottom row (i.e. the row that contains two other candidate gaps with starting points 3 and 4 in Figure 5.2a). Otherwise, if a box can be placed into either starting point 1 or 2 in Figure 5.2a, the deepest-bottom row of the next step may remain the same (e.g. Figure 5.2b) or vary (e.g. Figure 5.2c).

5.2.2 Box-placement Process

In the box-placement decision, a combination of two heuristics is used for each gap: *gap-filling* and *tie-breaking* heuristics. The first one plays a major role in finding a list of boxes that can fill the largest gap and the second one uses a particular strategy to break any existing ties. There are also different ways to define the *largest gap* that boxes can fill. For example, a box that can fill the maximum gap-width is not necessarily the box that can fill the maximum gap-height. We consider three gap-filling heuristics *GF1*, *GF2*, and *GF3* in Table 5.1 with different definitions of the box that can fill the largest gap.

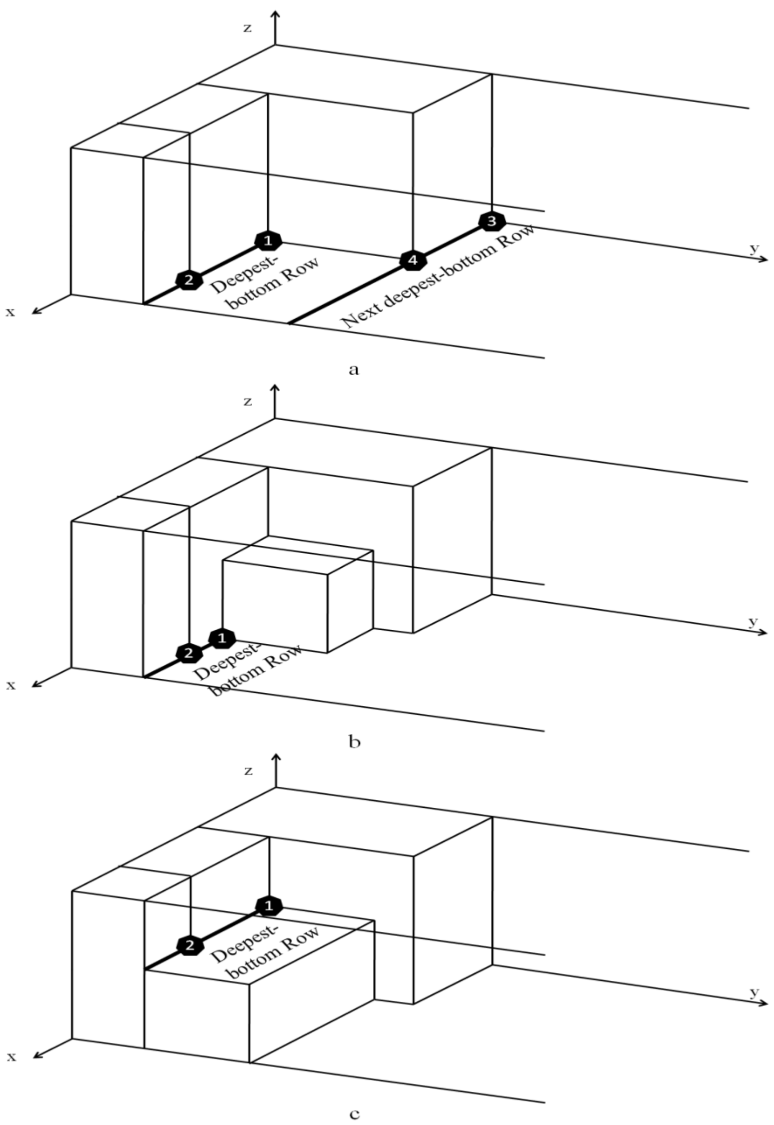


Figure 5.2 Examples of the deepest-bottom rows and starting points of candidate gaps

For the 3D-SPP-NS, 3BF has been shown to be a highly effective constructive heuristic (Allen et al., 2011). However, within the hyper-heuristic context, *GF1*, *GF2*, and *GF3* provide limited diversity in the box being selected, i.e. in many cases, there is only one box satisfying the criteria of these gap-filling heuristics. Therefore, we introduce three variants of these gap-filling heuristics in Table 5.1 to select boxes. The variant *$GF1_P$* for *GF1* selects boxes that cover at least *P* percent of the *xz*-covering area of the boxes found by *GF1*. Similar definitions are applied for *$GF2_P$* and *$GF3_P$*. A small-enough value of *P* will result in a higher number of candidate boxes, thus increasing the search space of packing solutions for hyper-heuristics. Note that when *P* is set to 100, these variants become their original gap-filling heuristics. All experiments in this research use the proposed variants with a parameter *P* instead of the pure gap-filling heuristics.

In this research, the gap-filling heuristics will be chosen intuitively based on problem characteristics. For the 3D-SPP-NS, it is sufficient to use only *$GF1_P$* to construct towards a compact packing. For the 3D-SPP with the stability constraint (3D-SPP-WS), boxes selected by *$GF2_P$* and *$GF3_P$* are also considered as they may have higher impacts on the overall packing quality. For instance, boxes selected by *$GF3_P$* generally provide greater support on the top surface for other boxes than boxes selected by *$GF1_P$*.

Gap-filling heuristic	**Description**
GF1 - Max xz-covering Area	Selects boxes that cover the maximum area on the cutting *xz*-surface containing the considering row and perpendicular to container's *y*-axis (see Figure 5.3a).
GF2 – Max Volume	Selects boxes that cover the maximum volume (see Figure 5.3b).
GF3 – Max xy-covering Area	Selects boxes that cover the maximum area on the cutting *xy*-surface containing the considering row and perpendicular to container’s *z*-axis (see Figure 5.3c).
$GF1_P$	Selects boxes that cover at least *P* percent of the *xz*-covering area of the boxes found by *GF1*.
$GF2_P$	Selects boxes that cover at least *P* percent of the volume of the boxes found by *GF2*.
$GF3_P$	Selects boxes that cover at least *P* percent of the *xy*-covering area of the boxes found by *GF3*.

Table 5.1 Three gap-filling heuristics and their variants in the box-placement procedure

There is another motivation for designing these variants for the gap-filling heuristics. Although the original heuristics are likely to produce a high quality packing, they often lead to wasted space. For example, when a box covers the most area on the *xz*-surface of the gap, it may leave the remaining space on the *x*-axis and the *z*-axis too small to contain any extra boxes. By considering other boxes or their rotations that cover slightly less area, the overall utilisation of the space may be improved. Figure 5.4 presents an example of such scenarios. The assignment of box 1 which covers the maximum *xz*-area results in a wasted space (grey area), while by selecting box 2, boxes 3 and 4 are able to fill the remaining space.

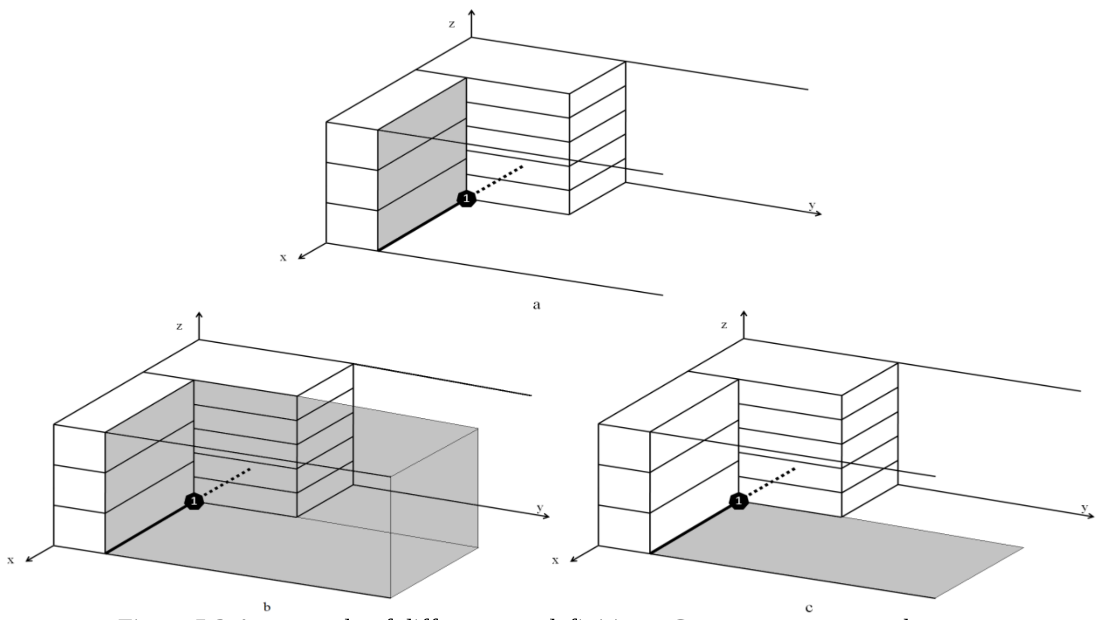


Figure 5.3 An example of different gap definitions. Grey areas represent the gaps

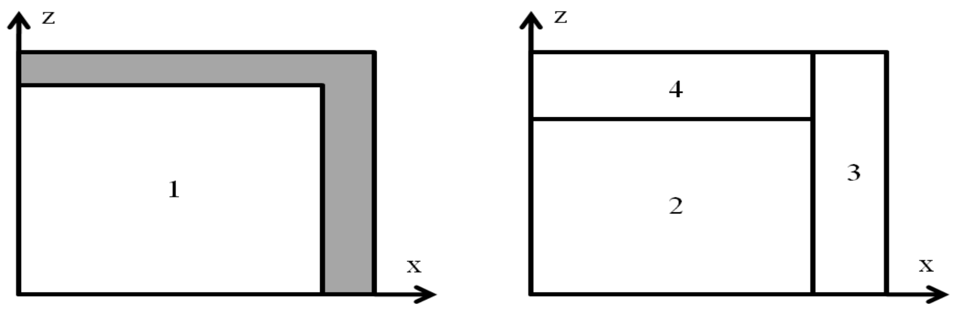


Figure 5.4 An example of reducing wasted space by assigning a box with a slightly smaller *xz*-covering area

The variants of the gap-filling heuristics with a large-enough *P* will consider more boxes while still preserving the 'best-fit' characteristic. After a variant of a gap-filling heuristic is applied, tie-breaking heuristics are employed to further improve the decision making to select the most suitable box for the chosen gaps. The tie-breaking heuristics listed in Table 5.2 are designed concerning different packing situations encountered in practice. Some of them will be applied for both the 3D-SPP-NS and the 3D-SPP-WS; some will be applied for only one problem class.

Tie-breaking heuristic	**Description**
TB1 - Maximum xz-covering Area	Prefers boxes that have the larger *xz*-covering area to fill as much gap on the deepest surface as possible.
TB2 - Maximum Volume	Prefers boxes with a larger volume.
*TB3 - *Maximum xy-covering Area*	Prefers boxes that have the larger *xy*-covering area to support other boxes.
TB4 - Maximum Contact Area	Prefers boxes covering larger area of surfaces which is in contact with other placed boxes.
*TB5 - *Minimum Rotations with Maximum min_length*	Prefers boxes with a smaller number of rotations. It is very likely there will be ties. In these cases, the minimum length of all rotations of each box is measured, denoted as *min_length*. In the tie list, we prefer the box that has a higher *min_length*. For example, some boxes may have large width and length, but small height, and can be rotated only on *z*-axis. Leaving these boxes until the late stage of packing may significantly increase the resulting container's length.
TB6 - Maximum Furthest_y Surrounding Contact	Calculates the contact length of edges surrounding the furthest-*y* surface. This prefers boxes that may form a vertical wall at the furthest-*y* surface and thus are less likely to increase the container length. An example is shown in Figure 5.5.
*TB7 - *Minimum Wasted Volume*	Prefers boxes that may generate the least wasted space to the overall packing. We calculate all wasted spaces adjacent to the top, bottom, left, and right surfaces of the box placement. See Figure 5.6 for an example of the wasted spaces on the right surface of different candidate placements. The implementation for this heuristic is simplified and presented in Appendix B.
*TB8 – *Maximum Support*	Prefers boxes that can support a larger number of unallocated boxes on their top surface.

Table 5.2 The list of tie-breaking heuristics. The novel ones proposed in this research are marked with (*). The remaining is originated from (Allen et al., 2011)

If ties exist after applying a gap-filling and a tie-breaking heuristic, we sort the list of qualified boxes in descending order of the largest width, then height and then length of the boxes. This is to encourage the early assignment of bigger boxes. After that, if there are still ties, the first encountered box in the ordered list is selected.

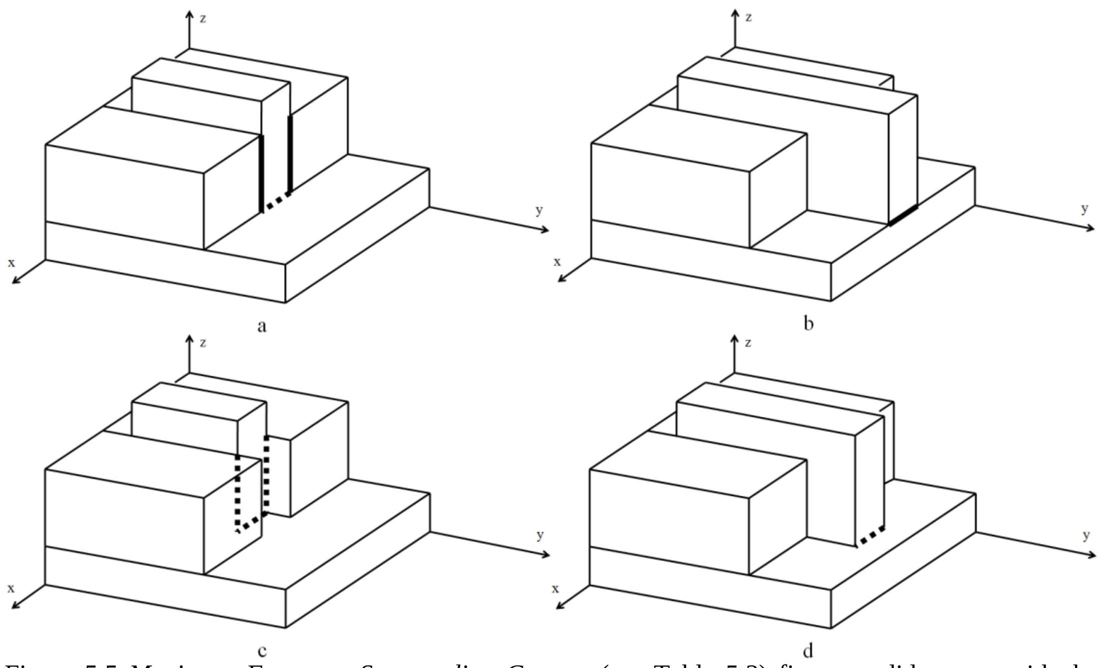


Figure 5.5 *Maximum Furtest_y Surrounding Contact* (see Table 5.2) fits a candidate gap with the preferred box in the order of: a, b, c, and then d. The solid bold lines represent the contact with other boxes with the same level of the furthest-*y*. The dotted lines represent the contact with other boxes with the furthest-*y* coordinate being greater than the furthest-*y* coordinate of the considering box. The heuristic prefers boxes that can form vertical walls with other boxes. Otherwise, it prefers boxes that have the furthest-*y* coordinate not exceeding the furthest-*y* coordinates of adjacent boxes

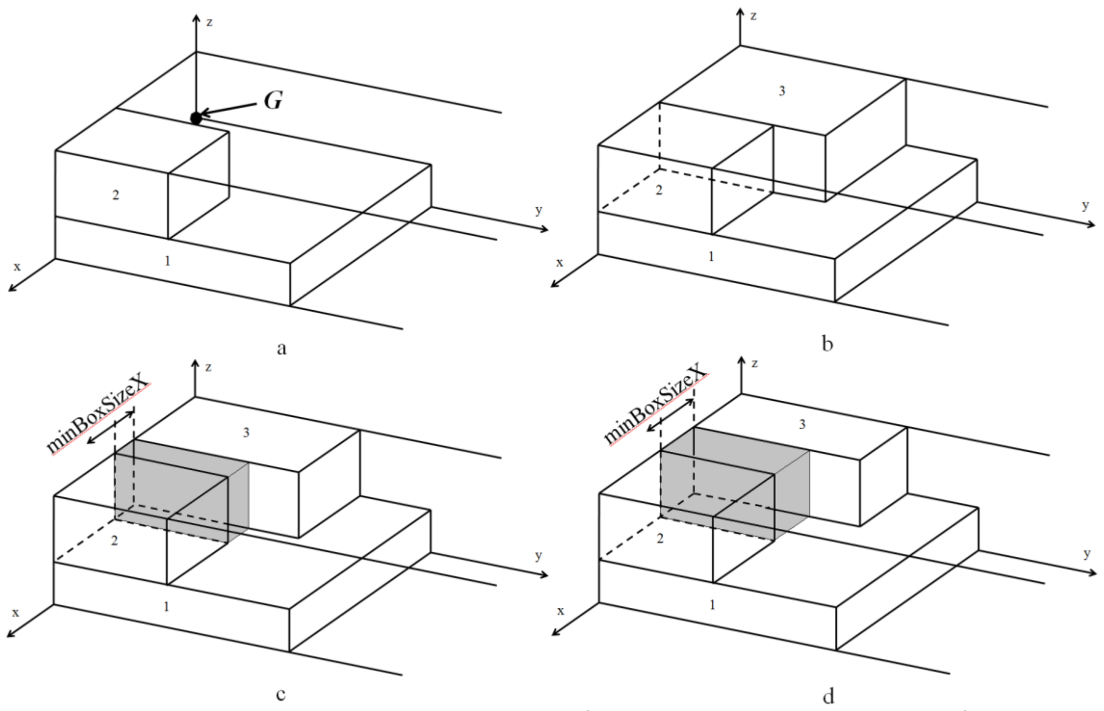


Figure 5.6 *Minimum Wasted Volume* (see Table 5.2) fits a candidate gap *G* with the preferred box in the order of: b, c, and then d. The grey areas represent wasted spaces. During the packing process, the information of the box types with the minimum width (*minBoxSizeX*) and minimum height (*minBoxSizeZ*) are dynamically updated

5.2.3 Assembled Box

We allow a number of boxes of the same type to be combined as a single box which is called an assembled box. An assembled box must be of rectangular shape, and is treated exactly the same as a single box. An example in Figure 5.7a demonstrates all possibilities to form an assembled box from a box type with quantity 4. Note that assembled boxes along the *y*-axis (e.g. in Figure 5.7b) are not allowed in this research. This follows the objective of the 3BF to heuristically pack the gaps deeply inside the container to receive as compact a packing as possible. A small amount of wasted volume towards the left, right, top, and bottom surfaces caused by one box may be accumulated to become a significant amount of wasted volume if that box is assembled on the *y*-axis.

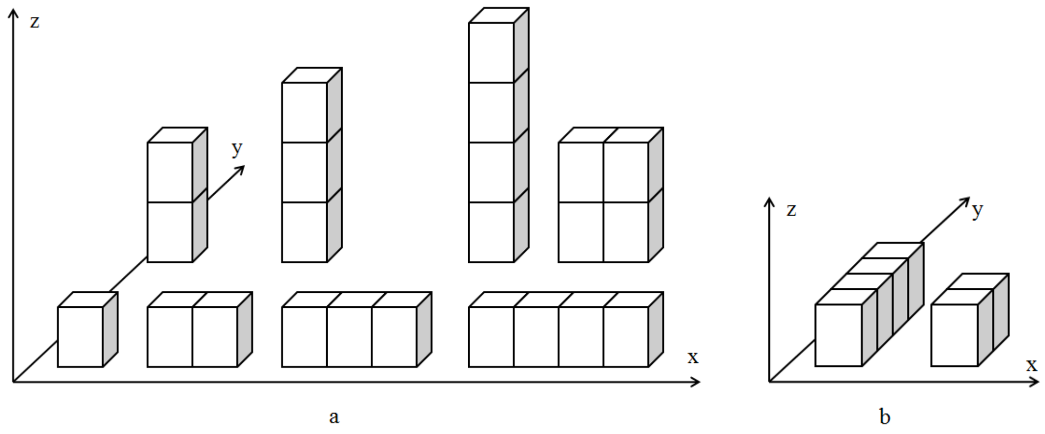


Figure 5.7 Examples of assembled boxes

5.2.4 A Technique to Adjust Box Positions

To further improve the compactness of packing, we introduce here a simple yet effective adjustment technique to slide a box after it is placed into a candidate gap. For the 3D-SPP-NS, the technique concerns sliding a box either left, right, up, or down, while this technique is only applied to slide a box left or right in the 3D-SPP-WS. The main process of this adjustment includes the following steps:

- Given a direction to slide the box, we calculate the colliding distances (see an example in Appendix B) from all points on the corresponding edge of the box's furthest-*y* surface (e.g. if the direction is to the right, the corresponding edge is the right edge).

- If the colliding distances from all those points are the same and smaller than the corresponding minimum box size (i.e. *minBoxSizeX* for sliding left and right, *minBoxSizeZ* for sliding up or down), the box is moved as much as possible towards the direction of that surface.

Figure 5.8a presents an example of placing box 3 into a position that results in wasted volume (dark area) on its right side. The adjustment is applied to slide box 3 to the right until it touches the container wall. Not only may this adjustment reduce wasted volume (dark areas in Figure 5.8), but it may also create larger gaps (grey area) for the next packing step. It is equivalent to increase the box-placement options for the neighbour gaps, thus increasing the chance of achieving a more compact packing.

If the stability constraint is applied, the calculated distances and the moves of the boxes are subject to the support on its bottom. In addition, the adjustment is made only if it does not reduce the supporting area connected to the top surface of the considered box. For the example presented in Figure 5.8, concerning the stability constraint, box 3 will not be moved because the top surface of box 3 together with box 2 forms a larger supporting area.

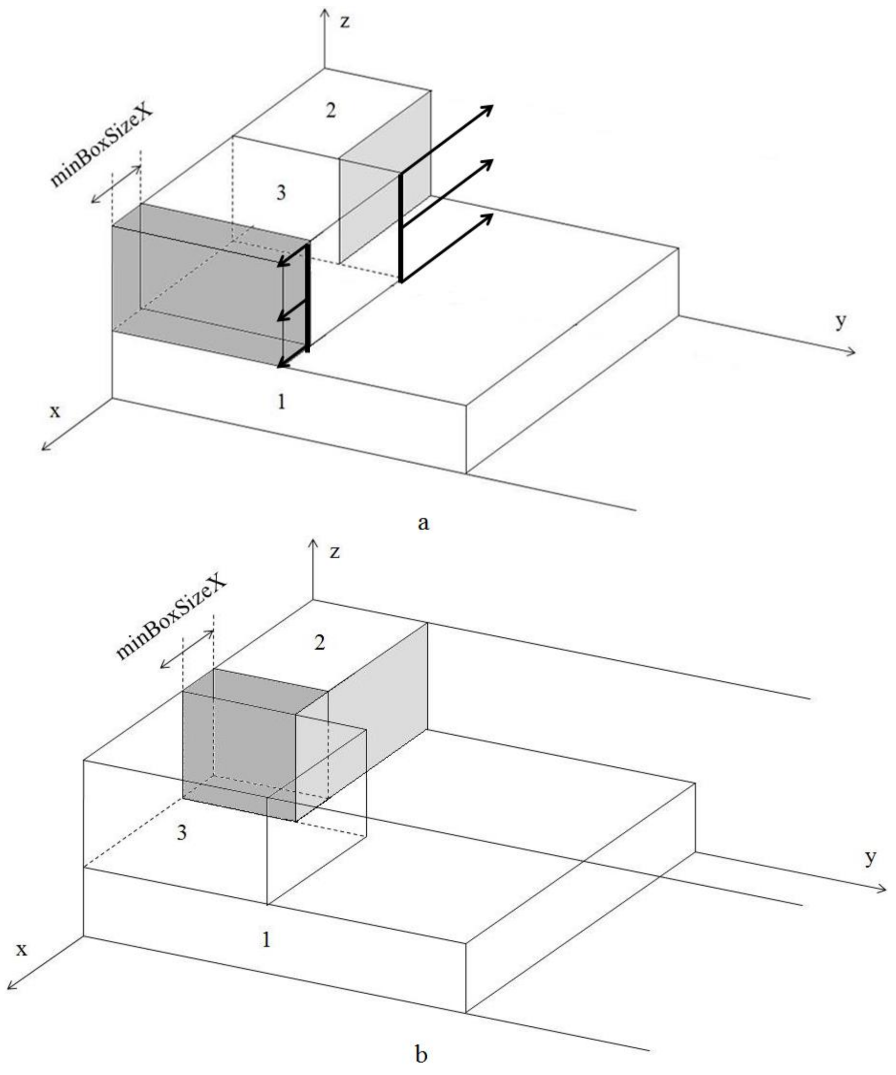


Figure 5.8 Adjusting the box position

5.2.5 Tower Processing

The nature of the 3BF is to repeatedly find a box to fill the deepest gaps. It is likely that a box that covers a small *xz*-area with large length *y* is placed at the end. Accommodating that box may result in a significant increase to the overall length of the container i.e. it forms a *tower*. The container length can sometimes be reduced simply by rotating the tower and assigning it to another, possibly not deepest position. A tower processing procedure as in (Burke et al., 2004b; Allen et al., 2011) is also implemented in this research. After all boxes are placed into the container, the box with the furthest y may be rotated and assigned to a different position to reduce the overall y. This process is repeated until no reduction is possible. Similar to the technique to adjust box positioning in Section 5.2.4, for the 3D-SPP-WS, changing a box’s position is subject to the supporting requirement in both old and new positions.

5.2.6 Pseudo-code for the Extended Strategy

The pseudo for our extended strategy is presented in Algorithm 5.1. For the input of a sequence of heuristics, each heuristic in the sequence is represented by a combination of a gap-filling heuristic and a tie-breaking heuristic. Our extended strategy first initialises the remaining number of boxes $N_{remaining}$ to be the total number of boxes N and finds the deepest bottom row R. Then, a while-loop repeatedly accommodates at each iteration a single box or an assembled box. The process at each iteration follows the following order. We apply the in-turn heuristic to find all qualified box placements on row R and store them in the list LP. If there is at least one qualified box placement, we will take the first qualified (assembled) box b in the list and get its assumed position p. We then apply the technique proposed in Section 5.2.4 to find a better position p' for the box (possibly not moving) and place it into that position. $N_{remaining}$ is deducted by the number of boxes in b. However, if there is no qualified box placement on row R, we continue by processing the next deepest bottom row and ignoring all unoccupied gaps in the current row R. The process stops after all boxes are accommodated. At the end, the tower processing is applied to further improve the volume utilisation.

Algorithm 5.1 The Extended Best-Fit Strategy

```
Input:  the number of boxes $N$
        a sequence of heuristics $H_1 \ldots H_n$

$N_{remaining} = N$
$i := 1$
find the deepest bottom row $R$

while ($N_{remaining} > 0$) do
    use heuristic $H_i$ to find the list of qualified box placements $LP$ on $R$

    if ($LP$.length > 0) then
        $b$ := get the first qualified (assembled) box from $LP$
        $p$ := get the box's positioning
        adjust the box to a new position $p'$
        place box $b$ into position $p'$
        update the number of remaining unassigned boxes $N_{remaining}$
        $i := i + 1$
    else
        find the next deepest bottom row $R'$
        $R := R'$
    end if
end while
apply tower processing procedure
```

5.3 Comparison between the new Strategy and the Original 3BF

We compare our extended 3BF with the previously implemented 3BF for the 3D-SPP-NS on the datasets SP-BR and SP-BR-XL. Four sequences of random heuristics are generated in which each element represents a heuristic for the 3D-SPP-NS. Each heuristic is applied in order to pack one or several boxes until all boxes are accommodated. The volume utilisations obtained from those sequences are compared with the results obtained from the 3BF using four different tie-breaking heuristics implemented in (Allen et al., 2011).

5.3.1 Experimental Settings

Experiments were conducted on a PC Pentium IV 3.4GHz with 2GB memory.

Heuristics

Each heuristic is represented by a box-placement strategy which combines one or more gap-filling heuristics (see Table 5.1) as the main `selection mechanism with one tie-breaking heuristic (see Table 5.2) to shorten the list of qualified boxes. Such combinations are denoted as:

[*Gap-filling heuristic*] [*Tie-breaking heuristic*].

Each heuristic can be one of the following combinations:

H1	-	[$GF1_P$] [*TB1*]
H2	-	[$GF1_P$] [*TB2*]
H3	-	[$GF1_P$] [*TB4*]
H4	-	[$GF1_P$] [*TB5*]
H5	-	[$GF1_P$] [*TB6*]
H6	-	[$GF1_P$] [*TB7*]

Fitness Measurement

Instead of measuring the required length for the container, we use the volume utilisation function (see Section 2.2.4) as the fitness evaluation for packing solutions.

Parameter Settings for the Gap-filling Heuristics

For all experiments, we set the parameter *P* (defined in Section 5.2.2) for the gap-filling heuristics $GF1_P$, $GF2_P$, and $GF3_P$ as in Table 5.3. As weakly heterogeneous test cases have less box types, *P* is set to smaller values to increase the number of considering options at each decision point. These values for *P* are also selected empirically by performing trial runs and will be used throughout all experiments on the 3D-SPP in this thesis. We aim at having more box options for each decision step while still keeping the characteristic of the 'best-fit' strategy.

SP-BR(-XL)	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**
P	65	65	75	75	85	85	85	90	90	90

Table 5.3 Parameter settings for the gap-filling heuristics in different test cases

5.3.2 Experimental Results

For each test case of the SP-BR and SP-BR-XL datasets, the result is for the first 10 instances (similar to the compared approach). We report the mean volume utilisations obtained by our approaches from 3 different runs (MVU). The standard deviations (StDev) are also presented.

Tables 5.4 and 5.5 show the comparison results on the datasets SP-BR and SP-BR-XL, respectively. The average results are comparable on both SP-BR and SP-BR-XL dataset; however, for the weakly heterogeneous test cases (i.e. 1 and 2) we currently set *P* only at 65%. If the running time is strictly limited, a higher value of P will result in significant improvement in volume utilisation due to the selections of larger boxes for the gaps. However, adjusting *P* is not the subject of this research. We aim at using a fixed set of *P* to test different problem situations.

Test case	**3BF (Allen et al., 2011)**		**Extended 3BF**		
	VU(%)	Time(s)	MVU(%)	StDev(%)	Time(s)
SP-BR01	88.7	0.3	87.4	0.58	0.3
SP-BR02	89.0	0.4	88.2	0.29	0.3
SP-BR03	87.8	0.5	88.1	0.48	0.3
SP-BR04	87.8	0.6	87.5	0.11	0.4
SP-BR05	87.7	0.7	88.7	0.47	0.4
SP-BR06	87.6	0.9	88.2	0.33	0.5
SP-BR07	87.4	1.1	87.9	0.34	0.5
SP-BR08	86.8	1.3	86.8	0.24	0.6
SP-BR09	86.5	1.6	87.3	0.40	0.s6
SP-BR10	86.3	2.0	86.4	0.49	0.7
Average	87.6		87.6		

Table 5.4 Results on the SP-BR dataset from four randomly generated sequences for the 3D-SPP-NS

Test case	**3BF (Allen et al., 2011)**		**Extended 3BF**		
	VU(%)	Time(s)	MVU(%)	StDev(%)	Time(s)
SP-BR01-XL	92.2	8.6	91.0	0.39	3.4
SP-BR02-XL	92.2	10.2	90.6	0.37	3.9
SP-BR03-XL	91.8	12.5	91.8	0.04	4.2
SP-BR04-XL	92.0	14.9	92.1	0.26	4.6
SP-BR05-XL	92.4	16.8	93.4	0.20	4.6
SP-BR06-XL	92.5	19.4	93.6	0.14	5.1
SP-BR07-XL	92.4	22.9	93.2	0.11	5.2
SP-BR08-XL	92.6	26.2	94.2	0.18	5.5
SP-BR09-XL	92.1	30.1	94.1	0.07	5.7
SP-BR10-XL	92.5	35.3	94.0	0.12	6.3
Average	92.3		92.8		

Table 5.5 Results on the SP-BR-XL dataset from four randomly generated sequences for the 3D-SPP-NS

5.4 Chapter Summary

We presented in this chapter an extended version of the 3BF. The use of gap-filling heuristics with parameter *P* can offer more options to select boxes at every decision. Four new tie-breaking heuristics in Table 5.2 are proposed based on the common intuition of human packing in different situations. By using suitable combinations of gap-filling and tie-breaking heuristics, larger areas in the solution space can be explored compared to the original 3BF. A particular set of settings for parameter *P*, which will be used throughout all experiments in this research, has shown comparable results with those obtained by the original 3BF. The next chapter will investigate the application of a hyper-heuristic which uses this extended 3BF within one of the experimented domains – the 3D-SPP.

Chapter 6 An Estimation of Distribution Algorithm-based Hyper-heuristic (EDA-HH)

We investigate a hyper-heuristic method based on estimation of distribution algorithms (EDAs) to solve different classes of the exam timetabling problems and the 3D strip packing problems. The challenge is to develop automated methods which choose and hybridise suitable heuristics based on problem solving situations from a given set of low-level heuristics. The EDA-based hyper-heuristic works on the search space of constructive heuristics and is not restricted to solve one problem but is applicable for a number of optimisation problems. We demonstrate a higher level of generality by applying the hyper-heuristic on four different problem variants i.e. the ETP, the ETP with only the student-conflict hard constraint, the 3D-SPP without a stability constraint and the 3D-SPP with a stability constraint. For the 3D-SPP, this hyper-heuristic works on the extended best-fit strategy presented in Chapter 5. The hyper-heuristic can produce comparable results to those obtained using other approaches reported in the literature. Some of the results are even better than the best reported ones. We also present the probability distributions of the low-level heuristics which are hybridised by EDA-HH in solving the investigated problems. These probability distributions naturally represent a technique to identify effective or ineffective heuristics during and after the search. This capability of our hyper-heuristic may help facilitate more intelligent hyper-heuristics in future research.

6.1 A Review on Estimation of Distribution Algorithms

Evolutionary algorithms represent a class of population-based approaches which include two essential procedures: selecting high-quality solutions (parents) and applying variation operators to these solutions to obtain offspring with close characteristics to their parents. By repeatedly applying these procedures coupling with updating the population in each iteration, the solutions obtained from the evolutionary algorithms gradually evolve to have better fitness. One of the main requirements for success of an evolutionary algorithm is therefore to have an efficient variation operator that captures the features which distinguish high-quality solutions from the inferior ones. In many problem domains, the interaction between variables (genes) has a positive impact on the quality of solutions such as groups of variables being close together (*building blocks*) or linkage between variables. Despite being successful in many applications, traditional genetic operators (crossover and mutation) in genetic algorithms usually cause the disruption of the groups or linkage between variables having positive effects in solution fitness due to random nature. One of the directions to avoid these disruptions is to change the variation process.

6.1.1 Introduction on EDAs

Estimation of Distribution Algorithms (Baluja, 1994; Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2002; Pelikan et al., 2002) were proposed as an alternative for GAs within evolutionary algorithms. EDAs replace genetic operators with explicit modelling of gene distribution in promising individuals and sample new offspring from that distribution. EDAs have attracted increasing research interest recently and been successfully applied to numerous problems including amino-acid alphabet reduction for protein structure prediction (Bacardit et al., 2007), economic dispatch (Chen and p. Chen, 2007), forest management (Ducheyne et al., 2004), portfolio management (Lipinski, 2007), environmental monitoring network design (Kollat et al., 2008), multi-objective knapsack (Shah and Reed, 2010). In all these publications, EDAs were shown to either achieve better performance or be capable of solving problems of the largest size compared to other techniques. We outline in Algorithm 6.1 the pseudo-code of a basic EDA.

Algorithm 6.1 EDA pseudo-code

```
$t$ := 0;
generate initial population $P(0)$
evaluate $P(0)$
while (not done) do
    select a set of promising solutions $I(t)$ from $P(t)$
    build probabilistic model $M(t)$ from $I(t)$
    sample $M(t)$ to generate new candidate solutions $O(t)$
    evaluate $O(t)$
    incorporate $O(t)$ and $P(t)$ into $P(t + 1)$
    $t := t + 1$
end while
```

EDAs contain all basic procedures of an evolutionary algorithm. Starting from generating solutions for the initial population, EDAs then repeatedly select promising parent solutions (e.g. using truncation selection or tournament selection techniques) and sampling offspring candidate solutions based on the probability distribution of variables in the parent solutions obtained from a probabilistic model, until some stopping criteria is met (usually after a predefined number of iterations or when a solution of sufficient quality is found). The step of incorporating the offspring candidate solutions with the population solutions of the current iteration will select a certain number of solutions for the new population based on some criteria (e.g. fittest solutions). Many EDAs usually replace the whole population by its offspring. As mentioned, the step that differentiates EDAs from other evolutionary algorithms is the construction of a probabilistic model and the process of sampling new candidate solutions based on the model.

6.1.2 A Classification of EDAs

EDAs can be categorised into three categories based on the solution representation of the problem, i.e. discreet variables, permutation and real-valued variables. Candidate solutions in EDAs are usually of fixed length. However, variables can either have a finite cardinality (discreet variables) or receive a real value that covers an infinite domain. Candidate solutions can also be represented by a permutation over a given set of elements, e.g. the travelling salesman problem and the quadratic assignment problem. This research concerns only EDAs for discreet variables. A brief survey on the other types of EDAs can be found in (Pelikan et al., 2002; Hauschild and Pelikan, 2011). For EDAs with discreet variables, EDAs can be divided into three groups: Univariate, Bivariate and Multivariate based on the level of interactions among variables.

Univariate EDAs assume no interaction among variables. The joint probability distribution of a solution which will be used afterward in the sampling process is simply the product of univariate marginal probabilities of all variables in that solution. The algorithms in this category have simple model building and sampling procedures and can solve problems where variables are independent. However, for problems with variable interactions, they tend to produce poor results. Different variants in this category include: Population-based Incremental Learning (PBIL)

(Baluja, 1994), Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein and Paaß, 1996) and Compact Genetic Algorithm (cGA) (Harik et al., 1997).

For bivariate EDAs, pair-wise interactions among variables in the solutions are considered. Therefore, the probabilistic models of the algorithms in this category contain factors involving conditional probability of two interacting variables. New solutions are also sampled in a certain ordering of variables to address the conditional probabilities. These algorithms outperform univariate EDAs in problems with pair-wise variable interactions; however, they tend to fail when multiple interactions among variables exists in the problem. Mutual Information Maximisation for Input Clustering (MIMIC) (De Bonet et al., 1997), Combining Optimiser with Mutual Information Tree (COMIT) (Baluja and Davies, 1997) and Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan and Mühlenbein, 1999) all use bivariate models to estimate probability distribution.

Multivariate EDAs use probabilistic models capable of capturing multivariate interactions between variables. However, constructing such models becomes particularly computational expensive with the increasing order of interactions. Searching through all possible models is usually infeasible. Algorithms using multivariate models of probability distribution include: Extended Compact Genetic Algorithm (ECGA) (Harik, 1999), Bayesian Network Algorithm (EBNA) (Etxeberria and Larrañaga, 1999), Factorised Distribution Algorithm (LFDA) (Mühlenbein and Mahnig, 1999), Bayesian Optimisation Algorithm (BOA), (Pelikan et al., 2000), Hierarchical Bayesian Optimisation Algorithm (hBOA) (Pelikan, 2005), Markovianity-based Optimisation Algorithm (MOA) (Shakya and Santana, 2008), Affinity Propagation EDA (AffEDA) (Santana et al., 2010).

The algorithms listed in this section tackle problems with a discreet variable representation. More general EDAs can obviously solve a broader class of problems. However, there is a trade-off that affects the choice of EDAs between the expressiveness of a probabilistic model and the computational complexity in the model building and the sampling processes.

6.2 Motivations on an EDA-based Hyper-heuristic

The focus of this chapter is on a constructive hyper-heuristic with the high-level search technique employing the idea of EDAs. It aims at selecting the most appropriate heuristics subject to problem solving situations. An objective of hyper-heuristic research is that they should be methods of higher generality than most meta-heuristics, i.e. they should be applicable for different problems or different classes of a problem. A key issue in hyper-heuristic research is to explore to what extent hyper-heuristics can reach in terms of generality. This chapter addresses that issue and contributes by providing the experimental results of applying a constructive hyper-heuristic to solve four different problems, i.e. the ETP, the ETP with only the student-conflict hard constraint known as the graph colouring problem (GCP), the 3D-SPP-NS and the 3D-SPP-WS.

Following the hyper-heuristic framework in Figure 2.1, the high-level search remains the same for the encountered problems while the sets of low-level heuristics and the evaluation functions (or fitness measurement for heuristic-choice solutions) are distinctively chosen.

Assuming that an individual is a sequence of genes, where each gene represents one low-level heuristic, the motivations of applying a high-level search mechanism based on EDAs are as follows.

- First, sampling new individuals based on the estimation of gene distribution of promising individuals is equivalent to generating new individuals with added knowledge. Thus, results are very likely to outperform an arbitrarily random sampling process. The comparison with solutions obtained from random selections of low-level heuristics has been widely used to justify many hyper-heuristic approaches.
- Second, as a constructive hyper-heuristic, the solution on the high-level search is represented by a number of heuristics being applied consecutively. By using an EDA as the high-level search technique, we can put the focus on exploring the building block of high-quality heuristic-choice solutions. In this case, a building block is understood to be a group of low-level heuristics close together within a particular stage in the solution construction. Obtaining probability distributions of heuristics in these building blocks supports the generation of new individuals with similar characteristics.
- Third, the estimation of gene distribution naturally provides insights into the importance of genes during the evolutionary process. This helps not only analyse the effectiveness of low-level heuristics, but it also enables the design of more intelligent search mechanisms in future work that control the search intensification and diversification based on the removal and addition of low-level heuristics.

- Last but not least, to the best of our knowledge, this is the first attempt in applying an EDA as the high-level search technique within a hyper-heuristic context.

Note that the following terms are used interchangeably within the context of our EDA-HH approach. An *individual* is equivalent to a *sequence* or a *heuristic-choice solution*. A *gene* is equivalent to a *low-level heuristic*.

6.3 Heuristic-Choice Solution Representation, Fitness Measure, and Construction Stages

The hyper-heuristic approach in this research follows many other approaches in the literature where a heuristic-choice solution is represented by a sequence of low-level heuristics. Each element in a sequence can be a single low-level heuristic or a number of low-level heuristics depending on the employed solution construction. For example, in the exam timetabling problem, the constructive decisions at each step concern the selection of one exam and the selection of one timeslot, i.e. one step requires a pair of low-level heuristics to select an exam and to select the timeslot for that exam. However, in the problems encountered in this chapter, we focus on investigating sequences in which each element represents only a single low-level heuristic for one type of constructive decision (see Table 6.1). If each step needs more than one constructive decision, the rest will be suggested by fixed strategies.

Problem	**Investigating heuristics**
ETP	Heuristics to select an exam
GCP	Heuristics to select a vertex
3D-SPP-NS	Heuristics to select a box, i.e. a compound heuristic (A gap-filling heuristic and a tie-breaking heuristic)
3D-SPP-WS	

Table 6.1 Heuristic types for the encountering problems

After all pairs of low-level heuristics in a sequence are consecutively applied, we obtain a solution. In this research, the fitness of a heuristic-choice solution in the high-level search is simply set as the evaluation of the solution obtained in the low level. The evaluation of the solutions in the low level will be provided for each problem.

For the ETP and GCP, the length of a sequence is equal to the number of exams and vertices, respectively. We divide a sequence into a number of stages N_{stage} where each stage consists of a fixed number of elements L_{block} (with the exception of the last stage). For the 3D-SPP-NS and 3D-SPP-WS, the length of a sequence varies as each low-level heuristic may assign either one box or a number of boxes (an assembled box) to a gap. Despite this varying length, each heuristic will be associated with one packing stage as follows. For each problem instance, an *approximate container's length* is divided into N_{stage} intervals of equal length. The approximate container's length is set as the minimum container's length obtained from a number of randomly generated solutions in an initialisation procedure in our hyper-heuristic. A box belongs to stage i if its origin is placed within the i^{th} interval. If a solution has some boxes placed beyond the approximate container's length, these boxes will belong to the last stage. As the extended 3BF repeatedly fills the deepest gaps, boxes will be packed in a consecutive order of stages (i.e. 1, 2, 3, ..., N_{stage}).

6.4 The High-level Search Technique

A high-level search technique based on EDAs will evolve the probability distribution towards those that are likely to sample fitter heuristic-choice solutions to subsequently produce high-quality packing. This technique employs a probabilistic model based on univariate EDAs, i.e. it assumes no dependency among variables. However, we consider each variable here as a construction stage of the problems instead of a single step of using a low-level heuristic to assign one exam or accommodate one box. The rationale behind the use of construction stage is due to an assumption that particularly differentiated decisions from genes in steps close together should be avoided. As a stochastic approach, the EDA high-level search technique will not rapidly reduce the variation of heuristic choices within a stage while it can enforce the assumption. Considering each variable as a construction stage also solves the problem of sequences with different lengths in the 3D-SPP due to the use of assembled boxes.

A univariate EDA is employed instead of more advanced models due to its simplicity in implementation. This is also because of the uncertainty in the dependency between low-level heuristics in different stages. Nevertheless, an investigation on more complex EDAs represents one direction in our future work.

The hyper-heuristic keeps track of a two-dimensional probability matrix p_{ij}, $0 \leq i < N_{stage}$; $0 \leq j < H$, where H represents the number of choices of low-level heuristics that can be assigned to an element of a sequence. The description of the high-level search technique consists of the steps outlined in Algorithm 6.1.

The calculation of p_{ij} uses the Laplace correction (additions of 1 and H on the numerator and denominator, respectively) to avoid situations where a low-level heuristic will disappear permanently after a particular generation. The stopping criterion here is to simply limit the running time or the number of generations. The fittest sequence obtained during this evolutionary process will represent the best-found packing solution. For the 3D-SPP-NS and the 3D-SPP-WS problems, the approximate container length discussed in Section 6.3 is obtained after step 1 in Algorithm 6.1.

Algorithm 6.1. The High-Level Search Technique

1. Set the generation index $t = 0$, generate a population D_t with N random sequences.
2. Evaluate the population.
3. Select $N_{select} \leq N$ sequences from D_t using tournament selection with size S.
4. In generation t, the probability of the j^{th} low-level heuristic appeared in the i^{th} construction stage of the selected sequences from step 2 is estimated as:

$$p_{ij} = \frac{\left(\sum_{k=1}^{N_{select}} \delta_k (X_i = j)\right) + 1}{\left(\sum_{k=1}^{N_{select}} C_{ik}\right) + H}$$

   where $\delta_k (X_i = j)$ represents the number of times the j^{th} low-level heuristic appeared in the i^{th} construction stage of the k^{th} selected sequence. C_{ik} represents the number of low-level heuristics needed for the i^{th} construction stage of the k^{th} selected sequence.

5. Generate the next generation D_{t+1} by sampling off-springs in which the probability of genes in the i^{th} construction stage receive value j is p_{ij}. Set $t = t+1$.
6. If stopping criterion is not satisfied, go to step 2. Otherwise, stop the process and output the best solution.

The aim of this hyper-heuristic is not to solve a specific problem. By online learning on only the probability of low-level heuristics previously used at different stages of solution construction, the high-level search relies only on non-domain-specific information, and thus can be generalised on different problems.

6.5 Performance of the EDA-HH

We demonstrate in this section the generality of our EDA-HH by applying it to four different problems. The settings in EDA-HH for each problem include only a set of low-level heuristics and a fitness measurement for sequences. The algorithm was implemented in Java using JDK 1.6 and experiments were conducted on a PC Pentium IV 3.4GHz with 2GB memory.

6.5.1 EDA-HH for Exam Timetabling and Graph Colouring

The EDA-HH approach is tested on the 13 Toronto benchmark instances (see Section 2.1.6) in both the exam timetabling and the graph colouring context.

Low-level Heuristics

The set of low-level heuristics includes five constructive heuristics listed in Table 6.2.

Low-level heuristic	**Description**
H1	Largest Degree (LD) - Exam in conflict with the largest number of exams is considered to be more likely to cause conflict if deferred until later.
H2	Largest Weighted Degree (LWD) - Exam with the maximum total number of students in conflict are more likely to cause high penalty.
H3	Saturation Degree (SD) - Exam with the least number of valid timeslots should be scheduled earlier since it may not have any timeslots available at a later stage.
H4	Largest Enrolment (LE) - Exam with largest enrolment should be selected first since its high number of students may cause high penalty if scheduled at a later time.
H5	Largest Coloured Degree (LCD) - Exam with the largest number of conflicts with those already scheduled would be difficult to schedule since it would have less choice of valid timeslots.

Table 6.2 Exam-selection low-level heuristics

In Chapter 3, we propose new heuristics sharing the same idea of traditional graph colouring heuristics, e.g. max-bad-colours and max-bad-intersect-edges. However, in this chapter, we only target heuristics which is fast instead of those that require a certain number of trials to find a good parameter settings (i.e. bad proximity colour parameter *pc* or bad intersect edge parameter *ie*).

Apart from those in Table 6.2, we also include $H1_2$, $H1_3$, $H2_2$, $H2_3$, $H3_2$, $H3_3$, $H4_2$, $H4_3$, $H5_2$, $H5_3$ into the set of low-level heuristics. $H1_2$, $H1_3$ use the same ordering strategy as H1, but take the second and the third vertex, respectively in the ordering list instead of the first one. The additions are the same for H2, H3, H4 and H5. A larger set of low-level heuristics provides a larger search space of heuristic combinations and improves the decision diversity.

For the exam timetabling problem, the set of low-level heuristics consists of all 15 exam-selection heuristics. For the graph colouring problem, the set includes only 9 heuristics (H1, $H1_2$, $H1_3$, H3, $H3_2$, $H3_3$, H5, $H5_2$, $H5_3$). Those using the ordering strategies of H2 and H4 are excluded as there is no penalty from the soft constraint in the graph colouring problem.

<u>Fitness Measurement</u>

For the exam timetabling problem with the student-conflict hard constraint and the exam-spread soft constraint, the strategy to choose a timeslot for a selected exam prefers a timeslot that causes no conflict and the least penalty. If there are ties, the tie-breaking strategy will choose the one that reduces the least number of valid timeslots for those unassigned neighbours of the considered exam. The evaluation of a feasible timetable in the Toronto dataset is given in Section 2.1. If a feasible timetable cannot be found (i.e. the timeslot assignment for a selected exam causes conflict at some point during solution construction), the fitness for that heuristic-choice solution s will be calculated as $f(s) = M + (L - i)$, where M is a big-enough value, i.e. greater than any possible evaluation of a feasible solution; L is the length of the sequence while i represents the first position in the sequence where infeasibility occurs.

For the graph colouring problem (i.e. the exam timetabling problem with only the student-conflict hard constraint), the evaluation can be simply the number of timeslots to obtain a feasible solution. When an exam is selected, we identify the minimum number of remaining valid timeslots for its neighbours, *minNeighbourValidTimeslot*. The timeslot-selection strategy will select the one with the highest *minNeighbourValidTimeslot*. The tie-breaker in this case is the same as the one for the exam timetabling problem described above. If there is no valid timeslot for an exam, the total number of timeslots is increased by one. However, experimental observations show that a significant number of sequences produce the same fitness, i.e. timetables using the same number of timeslots. Using the above evaluation function provides little distinction on the quality of solutions. Therefore, we employ here the evaluation function for graph colouring proposed by (Culberson, 1992). It concerns not only the number of colours, *k*, but also the *colouring sum* in a given colouring π:

$$f(\pi) = |V|k + \sum_{v \in V} \pi(v)$$

where V is the set of all vertices and $\pi(v)$ is the colour assigned to vertex v. This evaluation function prefers solutions having bigger size colour classes, thus it reduces smaller size colour classes and the overall number of colours used. A c*olour class* is understood to be a set of all vertices with the same colour.

Experimental Setup

The parameters for the high-level search were set to be the same for both testing problems. Some parameter values (including population size, number of generations, tournament size, and number of heuristics in a stage) were selected empirically by performing several trial runs. We then carry out extensive experiments on six sets of parameters presented in Table 6.3.

Parameters	**EDA-HH-TOUR6**	**EDA-HH-TOUR9**	**EDA-HH-TOUR12**
Population N – No. of Generations G	(100 – 20000), (1000 – 2000)		
Selection amount N_{select}	20% of N		
Tournament size S	6% of N	9% of N	12% of N
Number of heuristics in a stage L_{block}	10		
Maximum running time	24 hours.		

Table 6.3 High-level search parameter values

Results on the Exam Timetabling Problem

For each instance, EDA-HH is executed 10 times. In all runs using any set of parameters, the hyper-heuristic could find feasible solutions. Table 6.4 shows the best and average soft constraint penalty cost with the average running time of each setting on the Toronto uncapacitated exam timetabling instances.

Table 6.4 shows a clear preference to use a larger population size in EDA-HH (i.e. N=1000). In EDA-HH (1000-2000), the tournament selection TOUR9 could find a greater number of best results than both the tournament selections TOUR6 and TOUR12.

We also compare our hyper-heuristic with other hyper-heuristics tested on the same benchmark dataset in the literature. The main objective of a hyper-heuristic algorithm is to raise the level of generality over all instances rather than finding the best solution for some instances. Thus, we evaluate all hyper-heuristic approaches by calculating their average percentage differences to the best results reported in the literature. The algorithms producing the best results for the 13 exam timetabling problem instances are listed in Table 6.5 and the descriptions of the algorithms can be found in Chapter 2.1.

Table 6.6 presents the soft constraint penalty costs for the EDA-HH and other hyper-heuristic approaches. As described in Section 2.1.5, these constructive hyper-heuristic approaches work on the search space of heuristics. They either search for good sequences of low-level heuristics and apply them sequentially, or select good combinations of ordering criteria to measure the difficulty of unassigned exams. Table 6.7 shows the average percentage differences of the hyper-heuristic approaches to the best results reported in the literature. The approaches in Tables 6.6 and 6.7 include:

(1) The tabu-search developed by Burke et al. (2007)
(2) The variable neighbourhood search with two neighbourhoods (Qu and Burke, 2005)
(3) The automated heuristic construction using heuristic hybridisation (Qu et al., 2009a)
(4) Four different high-level search techniques based on local search (Qu and Burke, 2009)
(5) The fuzzy logic system on a pair of ordering criteria by Asmuni et al. (2005)
(6) The fuzzy logic system with tuning (Asmuni et al., 2009)
(7) The extended fuzzy logic system on three ordering criteria (Asmuni et al., 2007)
(8) The evolutionary algorithm on variable-length sequences (Pillay and Banzhaf, 2007)
(9) The approach that combines heuristics as tie-breakers (Pillay and Banzhaf, 2009)
(10) The genetic programming to evolve functions to order exams (Pillay, 2009)
(11) The linear combination approach in Chapter 4.

From Tables 6.6 and 6.7, the EDA-HH has produced promising results over all instances compared to the best results reported in the literature for each of the problems. It also demonstrates high generality over all instances compared to other hyper-heuristic approaches. We obtained the lowest average percentage differences to the best results cited in the literature.

<u>Results on the Graph Colouring Problem</u>

For each Toronto benchmark instance, EDA-HH is executed 10 times. Table 6.8 shows the best and average number of required colours with the average running time for each setting on the Toronto instances. We also compare our EDA-HH with the results obtained by using a hyper-heuristic that randomly selects heuristics (RS-HH).

Instance	EDA-HH (100-20000)						EDA-HH (1000-2000)						Approx. Average Running Time (hours)
	TOUR6		TOUR9		TOUR12		TOUR6		TOUR9		TOUR12		
	Best	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	
car91 I	5.13	5.17	5.14	5.17	5.13	5.16	4.98	5.00	**4.95**	4.99	4.96	5.00	10.05
car92 I	4.36	4.38	4.33	4.36	4.29	4.33	4.12	4.16	4.16	4.17	**4.09**	4.16	6
ear83 I	36.41	36.65	35.93	36.37	35.91	36.30	35.11	35.59	34.99	35.38	**34.97**	35.56	1.03
hec92 I	11.59	11.79	11.75	11.85	11.73	11.85	11.25	11.34	**11.11**	11.32	11.25	11.37	0.33
kfu93 I	14.93	15.25	14.81	15.12	14.75	15.11	**14.09**	14.49	14.19	14.50	14.15	14.32	1.93
lse91	10.97	11.12	10.91	11.05	10.89	11.06	10.77	10.89	10.77	10.90	**10.71**	10.87	1.64
pur93	4.77	4.78	4.76	4.77	4.76	4.78	**4.71**	4.73	**4.71**	4.74	4.72	4.73	24
rye92	9.86	10.03	9.87	10.03	9.9	10.02	9.23	9.31	**9.2**	9.33	9.25	9.32	2.95
sta83 I	**157.64**	157.95	157.82	157.96	157.81	157.97	157.75	157.92	157.81	157.92	157.76	157.91	0.58
tre92	8.5	8.56	8.49	8.53	8.51	8.54	8.28	8.31	**8.27**	8.34	8.29	8.33	1.91
uta92 I	3.43	3.45	3.43	3.45	3.43	3.44	3.35	3.37	**3.33**	3.36	3.34	3.36	8.27
ute92	26.77	27.06	26.91	27.19	27.05	27.21	**26.18**	26.79	26.68	26.85	26.68	26.82	0.61
yor83 I	40.23	41.11	40.45	41.26	40.81	41.26	38.25	38.79	**37.88**	38.58	38.31	38.91	1.02

Table 6.4 Experimental results on the Toronto exam timetabling benchmark. Bold values represent the best results obtained from our EDA-HH

Instance	**Caramia et al. (2001)**	**Yang and Petrovic (2005)**	**Burke and Bykov (2008)**
car91 I	6.6	**4.5**	4.58
car92 I	6.0	3.93	**3.81**
ear83 I	**29.3**	33.7	32.65
hec92 I	**9.2**	10.83	10.06
kfu93 I	13.8	13.82	**12.81**
lse91	**9.6**	10.35	9.86
pur93	**3.7**	-	4.32
rye92	**6.8**	8.53	7.93
sta83 I	158.2	158.35	**157.03**
tre92	9.4	7.92	**7.72**
uta92 I	3.5	**3.14**	3.16
ute92	**24.4**	25.39	24.79
yor83 I	36.2	36.35	**34.78**

Table 6.5 Best results reported in the literature on the Toronto exam timetabling benchmark. Bold font indicates best results. '-' represents the corresponding instance is not tested by the method

Instance	**Best reported**	**EDA-HH**	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**	**(10)**	**(11)**
car91 I	4.5	**4.95**	5.36	5.4	5.11	5.3	5.29	5.29	5.19	-	4.97	-	5.03
car92 I	3.81	**4.09**	4.53	4.7	4.32	4.7	4.56	4.54	4.32	-	4.28	-	4.22
ear83 I	29.3	**34.97**	37.92	37.29	35.56	35.54	37.02	37.02	36.16	36.74	36.86	37.39	36.06
hec92 I	9.2	**11.11**	12.25	12.23	11.62	12.23	11.78	11.78	11.6	11.55	11.85	11.43	11.71
kfu93 I	12.81	**14.09**	15.2	15.11	15.18	15.09	15.81	15.8	15.03	14.22	14.62	-	16.02
lse91	9.6	**10.71**	11.33	12.71	11.32	12.71	12.09	12.09	11.35	10.90	11.14	-	11.15
pur93	3.7	**4.71**	-	-	-	-	-	-	-	-	4.73	-	-
rye92	6.8	**9.2**	-	-	-	-	10.35	10.38	9.75	9.35	9.65	-	9.42
sta83 I	157.03	**157.64**	158.19	158.8	158.88	159.2	160.42	160.42	158.64	158.22	158.33	158.38	158.86
tre92	7.72	**8.27**	8.75	8.67	8.52	8.67	8.67	8.67	8.47	8.48	8.48	-	8.37
uta92 I	3.14	3.33	3.88	3.54	**3.21**	3.32	3.57	3.57	3.52	-	3.4	-	3.37
ute92	24.44	**26.18**	28.01	29.68	28.0	30.32	27.78	28.07	27.55	26.65	28.88	27.31	27.99
yor83 I	34.78	**37.88**	41.37	43	40.71	40.24	40.66	39.80	39.25	41.57	40.74	39.96	39.53

Table 6.6 Soft constraint penalty costs for hyper-heuristic approaches on the Toronto exam timetabling benchmark. Bold font indicates best results from hyper-heuristic approaches. '-' represents the corresponding instance is not tested by the method

Instance	**EDA-HH**	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**	**(10)**	**(11)**
car91 I	10.00	19.11	20.00	13.56	17.78	17.56	17.56	15.33	-	10.44	-	11.78
car92 I	7.35	18.90	23.36	13.39	23.36	19.69	19.16	13.39	-	12.34	-	10.76
ear83 I	19.35	29.42	27.27	21.37	21.30	26.35	26.35	23.41	25.39	25.80	27.61	23.07
hec92 I	20.76	33.15	32.93	26.30	32.93	28.04	28.04	26.09	25.54	28.80	24.24	27.28
kfu93 I	9.99	18.66	17.95	18.50	17.80	23.42	23.34	17.33	11.01	14.13	-	25.06
lse91	11.56	18.02	32.40	17.92	32.40	25.94	25.94	18.23	13.54	16.04	-	16.15
pur93	27.30	-	-	-	-	-	-	-	-	27.84	-	-
rye92	35.29	-	-	-	-	52.21	52.65	43.38	37.50	41.91	-	38.53
sta83 I	0.39	0.74	1.13	1.18	1.38	2.16	2.16	1.03	0.76	0.83	0.86	1.17
tre92	7.12	13.34	12.31	10.36	12.31	12.31	12.31	9.72	9.84	9.84	-	8.42
uta92 I	6.05	23.57	12.74	2.23	5.73	13.69	13.69	12.10	-	8.28	-	7.32
ute92	7.12	14.61	21.44	14.57	24.06	13.67	14.85	12.73	9.04	18.17	11.74	14.53
yor83 I	8.91	18.95	23.63	17.05	15.70	16.91	14.43	12.85	19.52	17.14	14.89	13.66
Average	**13.17**	18.95	20.47	14.22	18.61	20.99	20.87	17.13	16.91	17.81	15.87	16.48

Table 6.7 Percentage differences between hyper-heuristic approaches and the best reported results in the literature on the Toronto exam timetabling benchmark. '-' represents the corresponding instance is not tested by the method

Instance	**EDA-HH (100-20000)**						**EDA-HH (1000-2000)**						**Approx. Average Running Time (hours)**	**RS-HH ($2*10^6$ evaluations with time limit = 24hrs)**	**Max Clique (Battiti, 2001)**
	TOUR6		**TOUR9**		**TOUR12**		**TOUR6**		**TOUR9**		**TOUR12**				
	Best	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**			
car91 I	29	29	29	29	29	29	**28**	28	**28**	28	**28**	28	12.31	29	23
car92 I	**27**	27.5	**27**	27.2	**27**	27.2	**27**	27	**27**	27	**27**	27	7.59	28	24
ear83 I	**22**	22	**22**	22	**22**	22	**22**	22	**22**	22	**22**	22	1.15	22	21
hec92 I	**17**	17	**17**	17	**17**	17	**17**	17	**17**	17	**17**	17	0.25	17	17
kfu93 I	**19**	19	**19**	19	**19**	19	**19**	19	**19**	19	**19**	19	4.45	19	19
lse91	**17**	17	**17**	17	**17**	17	**17**	17	**17**	17	**17**	17	3.09	17	17
pur93	**32**	32	**32**	32	**32**	32	**32**	32	**32**	32	**32**	32	24	33	29
rye92	**21**	21	**21**	21	**21**	21	**21**	21	**21**	21	**21**	21	5.38	21	21
sta83 I	**13**	13	**13**	13	**13**	13	**13**	13	**13**	13	**13**	13	0.51	13	13
tre92	**20**	20	**20**	20	**20**	20	**20**	20	**20**	20	**20**	20	1.86	20	20
uta92 I	**29**	29.2	**29**	29.3	**29**	29.1	**29**	29	**29**	29	**29**	29.1	10.08	30	26
ute92	**10**	10	**10**	10	**10**	10	**10**	10	**10**	10	**10**	10	0.77	10	10
yor83 I	**18**	18	**18**	18	**18**	18	**18**	18	**18**	18	**18**	18	1.13	19	18

Table 6.8 Experimental results on the Toronto graph colouring benchmark. Underline instances represent easy instances; bold instances represent hard instances. Bold values represent the best results obtained by our EDA-HH

It is well known that the size of the maximum *clique* of a graph can be used as the lower bound to find the chromatic number of that graph. A clique in a graph is a subset of vertices where every two vertices are connected. The maximum clique found by a reactive local search technique on this benchmark (Battiti and Protasi, 2001) is also listed in Table 6.8 for comparison purposes.

For eight instances underlined in the 'Instance' column in Table 6.8, the optimal colourings can be found after generating only a few sequences. Although the best found result for *ear83 I* is greater than the maximum clique, we carry out a complete search on the solution search space and know that it is the optimal colouring. For the remaining five hard instances, we observe the same superiority of evolving sequences on a larger population as for the exam timetabling problem. Moreover, EDA-HH is always at least as good as the hyper-heuristic that randomly selects heuristics. This demonstrates the effectiveness of the learning process from the estimation of distribution algorithm in the high-level search.

Instance	**EDA-HH**	**(1)**	**(2)**	**(3)**
car91 I	28	28	28	30
car92 I	**27**	28	28	29
ear83 I	22	22	22	22
hec92 I	17	17	17	17
kfu93 I	19	19	19	19
lse91	17	17	17	17
pur93	**32**	35	36	-
rye92	21	21	21	-
sta83 I	13	13	13	13
tre92	20	20	20	20
uta92 I	**29**	32	30	31
ute92	10	10	10	10
yor83 I	**18**	19	19	19

Table 6.9 The minimum number of colours found by constructive approaches on the Toronto graph colouring benchmark. Bold values indicate new best solutions while optimal results are underlined. '-' represents the corresponding instance is not tested by the method

Table 6.9 shows the EDA-HH performance in comparison with other constructive approaches in the literature including:

(1) The constructive approach by Carter et al. (1996)
(2) The sequential construction method by Caramia et al. (2001)
(3) The automated heuristic construction using heuristic hybridisation (Qu et al., 2009a)

EDA-HH obtained better results than the best reported results in the literature in four hard instances (i.e. *car92 I*, *pur93*, *uta92 I*, and *yor83 I*).

6.5.2 EDA-HH for the 3D-Strip Packing

We also show the generality of our hyper-heuristic by evaluating it on the 3D strip packing domain. EDA-HH was tested on the SP-BR and SP-BR-XL datasets introduced by (Bischoff and Ratcliff, 1995) (see Section 2.2.4) for both the 3D-SPP-NS and the 3D-SPP-WS.

Low-level Heuristics

A low-level heuristic for the 3D-SPP is represented by a box-placement strategy which combines one or more gap-filling heuristics (see Table 5.1) as the main selection mechanism with one tie-breaking heuristic (see Table 5.2) to shorten the list of qualified boxes. Such combinations are denoted as:

[*Gap-filling heuristic*] [*Tie-breaking heuristic*].

For the 3D-SPP-NS, the set of low-level heuristics includes the following combinations:

$H1_{NS}$	-	[$GF1_P$] [*TB1*]
$H2_{NS}$	-	[$GF1_P$] [*TB2*]
$H3_{NS}$	-	[$GF1_P$] [*TB4*]
$H4_{NS}$	-	[$GF1_P$] [*TB5*]
$H5_{NS}$	-	[$GF1_P$] [*TB6*]
$H6_{NS}$	-	[$GF1_P$] [*TB7*]

For the 3D-SPP-WS, the set of low-level heuristics includes:

$H1_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB1*]
$H2_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB2*]
$H3_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB3*]
$H4_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB5*]
$H5_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB6*]
$H6_S$	-	[$GF1_P$ + $GF2_P$ + $GF3_P$] [*TB8*]

Note that a compound gap-filling heuristic [$GF1_P$ + $GF2_P$ + $GF3_P$], used in the 3D-SPP-WS, will consider all boxes that satisfy at least one criteria of $GF1_P$, $GF2_P$, or $GF3_P$.

Fitness Measurement

Instead of measuring the required length for the container, we use the volume utilisation function (see Equation 2.1) as the fitness evaluation for the heuristic solutions.

Experimental Setup

The parameters were set to be the same for both the 3D-SPP-NS and the 3D-SPP-WS. Some parameter values (including population size, selection amount, tournament size, grid, and number of stages) were selected empirically by performing several trial runs. We then carry out extensive experiments on the two sets of parameters presented in Table 6.10.

Parameters	**Set 1**	**Set 2**
Population size N	20	400
Stopping criteria	160 seconds	50 generations
Selection amount N_{select}	40% of N	
Tournament size S	30% of N	
Number of stages N_{stage}	5	
Grid (see an example in Appendix B)	5×5	

Table 6.10 Parameter settings

The settings of the parameter P (defined in Section 5.2.2) for the gap-filling heuristics $GF1_P$, $GF2_P$, and $GF3_P$ are consistent with those set in Section 5.3.1.

Results on the 3D Strip Packing Problems

For each test case, the result is for the first 10 instances (similar to comparison approaches). We report the mean volume utilisations obtained by our approaches from 3 different runs (MVU). The standard deviations (StDev) are also presented.

EDA-HH is experimented on the parameter set 1 to compare with the 3BF+TS (Allen et al., 2011) and the SPBBL-CC (Bortfeldt and Mack, 2007). The results in Tables 6.11 and 6.12 demonstrate clear improvements on most of the test cases, i.e. increases of 1.9% and 1.2% from the best reported average volume utilisations for SP-BR and SP-BR-XL datasets, respectively.

Test case	**3BF+TS**	**SPBBL-CC4**	**EDA-HH**	
	VU(%)	VU(%)	MVU(%)	StDev(%)
SP-BR01	90.0	87.3	**91.5**	0.11
SP-BR02	89.6	88.6	**91.4**	0.15
SP-BR03	89.0	89.4	**91.1**	0.17
SP-BR04	88.8	90.1	**91.3**	0.14
SP-BR05	88.5	89.3	**90.9**	0.05
SP-BR06	88.6	89.7	**90.8**	0.08
SP-BR07	88.7	89.2	**90.6**	0.17
SP-BR08	88.3	87.9	**89.6**	0.10
SP-BR09	87.9	87.3	**89.6**	0.11
SP-BR10	87.9	87.6	**89.3**	0.14
Average	88.7	88.6	**90.6**	

Table 6.11 Results of EDA-HH on the SP-BR dataset for the 3D-SPP-NS using the parameter set 1 (i.e. running time limit is 160 seconds)

Test case	**3BF+TS**	**SPBBL-CC4**	**EDA-HH**	
	VU(%)	VU(%)	MVU(%)	StDev(%)
SP-BR01-XL	**92.4**	86.9	91.9	0.14
SP-BR02-XL	**92.4**	88.3	91.8	0.15
SP-BR03-XL	91.9	89.8	**93.0**	0.08
SP-BR04-XL	92.1	90.2	**92.8**	0.14
SP-BR05-XL	92.5	89.9	**94.1**	0.07
SP-BR06-XL	92.6	91.5	**94.0**	0.06
SP-BR07-XL	92.6	91.0	**94.1**	0.11
SP-BR08-XL	92.8	90.8	**94.6**	0.11
SP-BR09-XL	92.3	90.9	**94.9**	0.06
SP-BR10-XL	92.7	90.4	**94.6**	0.06
Average	92.4	90.0	**93.6**	

Table 6.12 Results of EDA-HH on the SP-BR-XL dataset for the 3D-SPP-NS using the parameter set 1 (i.e. running time limit is 160 seconds)

Test case	**3BF+TS**	**SPBBL-CC4**	**EDA-HH**	
	VU(%)	VU(%)	MVU(%)	StDev(%)
SP-BR01	90.0	87.3	**93.1**	0.14
SP-BR02	89.6	88.6	**93.2**	0.08
SP-BR03	89.0	89.4	**93.5**	0.06
SP-BR04	88.8	90.1	**93.5**	0.06
SP-BR05	88.5	89.3	**92.9**	0.08
SP-BR06	88.6	89.7	**92.8**	0.07
SP-BR07	88.7	89.2	**92.3**	0.02
SP-BR08	88.3	87.9	**91.9**	0.06
SP-BR09	87.9	87.3	**91.3**	0.06
SP-BR10	87.9	87.6	**91.4**	0.02
Average	88.7	88.6	**92.6**	

Table 6.13 Results of EDA-HH on the SP-BR dataset for the 3D-SPP-NS using the parameter set 2

In order to understand further the capability of our approach, we allow the EDA-HH to run for a longer time, i.e. using the parameter set 2. Running times vary from approximately 2 hours for each instance in the SP-BR01 to approximately 5 hours for each instance in the SP-BR10. Apparently, the volume utilisations obtained on the SP-BR dataset for the 3D-SPP-NS are significantly improved as shown in Table 6.13. The aim here is, however, not to compare with the results obtained from other approaches but only to demonstrate the capability of our hyper-heuristic if the running time is not restricted.

We also show the generality of our hyper-heuristic by carrying out a similar experiment on the SP-BR dataset for the 3D-SPP-WS. Table 6.14 presents our results with the best results obtained from two different approaches TSACC-4P and GACC-4P in (Bortfeldt and Gehring, 1999). The hyper-heuristic can generate better volume utilisations in six test cases. Overall, the average volume utilisation of all 10 test cases is better than both approaches, i.e. an increase of 1.3% from TSACC-4P. The results shown in Tables 6.13 and 6.14 also demonstrate that given enough running time, our hyper-heuristic can solve both variants of the 3D-SPP to a competitive level.

Test case	**TSACC-4P**	**GACC-4P**	**EDA-HH**	
	VU(%)	VU(%)	MVU(%)	StDev(%)
SP-BR01	**92.3**	86.2	90.4	0.11
SP-BR02	**93.5**	84.3	91.2	0.15
SP-BR03	**92.3**	88.6	91.3	0.01
SP-BR04	**90.8**	89.0	90.4	0.13
SP-BR05	89.9	88.5	**90.0**	0.05
SP-BR06	89.2	88.1	**90.2**	0.29
SP-BR07	87.1	88.7	**88.9**	0.08
SP-BR08	83.9	87.8	**87.9**	0.11
SP-BR09	80.9	84.3	**86.8**	0.05
SP-BR10	79.1	82.1	**84.9**	0.03
Average	87.9	86.8	**89.2**	

Table 6.14 Results of EDA-HH on the SP-BR dataset for the 3D-SPP-WS using the parameter set 2

The Importance of Improvement Techniques

Experiments before this point always use the improvement techniques (i.e. box assembly, box position adjustment and tower processing). We demonstrate the importance of having these techniques in constructing packing solutions in Table 6.15. The columns show results of the following settings:

(1) EDA-HH without using any improvement techniques.
(2) EDA-HH with assembled boxes.
(3) EDA-HH with assembled boxes and box position adjustment.
(4) EDA-HH with all three improvement techniques.

Test case	**(1)**		**(2)**		**(3)**		**(4)**	
	MVU	StDev	MVU	StDev	MVU	StDev	MVU	StDev
SP-BR01	85.9	0.26	90.2	0.14	91.1	0.16	**91.5**	0.11
SP-BR02	86.5	0.12	89.9	0.15	91.0	0.18	**91.4**	0.15
SP-BR03	84.8	0.15	89.6	0.09	90.8	0.14	**91.1**	0.17
SP-BR04	85.6	0.17	89.6	0.14	90.4	0.09	**91.3**	0.14
SP-BR05	85.2	0.22	89.6	0.16	90.6	0.12	**90.9**	0.05
SP-BR06	84.8	0.12	89.8	0.18	90.1	0.11	**90.8**	0.08
SP-BR07	85.9	0.18	88.8	0.13	89.7	0.14	**90.6**	0.17
SP-BR08	85.0	0.13	88.2	0.11	89.3	0.09	**89.6**	0.10
SP-BR09	84.7	0.15	87.9	0.14	88.8	0.14	**89.6**	0.11
SP-BR10	85.3	0.21	87.1	0.17	88.5	0.16	**89.3**	0.14
Average	85.4		89.1		90.0		**90.6**	

Table 6.15 Results of EDA-HH using different sets of improvement techniques. Experiments are carried out using the parameter set 1 on the SP-BR dataset for the 3D-SPP-NS. The unit for volume utilisations and standard deviations is percent.

6.6 The Learning Capability of the EDA-HH

We present in this section the probability distributions of low-level heuristics. The observations show that certain low-level heuristics are often better than others on providing appropriate decisions in particular problem solving situations during the search. We also suggest a future hyper-heuristic research direction based on this capability of identifying the effective and ineffective heuristics.

6.6.1 Probability Distribution Observations on Exam Timetabling

Figure 6.1 shows the plots of the probability distribution of low-level heuristics from the run obtaining the best results of four sample instances: *hec92 I*, *sta83 I*, *ute92* and *yor83 I*. The probability of a heuristic (e.g. Saturation Degree – H1) is represented by the sum probability of its related heuristics (i.e. $H1_2$, $H1_3$). This probability distribution is recorded after the last generation of the evolutionary process. The probability at each stage on a curve represents the average probability of its last five stages. The remark we can make here is that saturation degree is an effective heuristic for the exam timetabling problem over a large period of solution construction; however, it is rarely used in the early stage. For instance *sta83 I*, the saturation degree is not particularly stronger than other low-level heuristics.

6.6.2 Probability Distribution Observations on Graph Colouring

Figure 6.2 presents the probability distribution of low-level heuristics from the run obtaining the best results of four hard instances: *car91 I*, *car92 I*, *uta92 I* and *yor83 I*. This probability distribution is recorded similarly as for the exam timetabling problem. The charts support the argument that different heuristics are suitable for different stages of the colouring process. Saturation degree has been shown to be among the most preferred exam-selection heuristics for the graph colouring variant.

However, applying saturation degree at the beginning of the colouring process is likely to produce a significant number of ties. Largest degree is most likely to be chosen at the very beginning of a colouring. Note that this observation applies to all other instances of the benchmark.

6.6.3 Probability Distribution Observations on 3D Strip Packing

Figures 6.3 and 6.4 present the average probability distributions of the first 10 instances for four sample test cases of the 3D-SPP-NS and the 3D-SPP-WS that EDA-HH obtained after the evolutionary process. These probability distributions show no general trend on the usage of low-level heuristics over all instances. That illustrates the capability of EDA-HH in adapting to different situations to produce competitive results. There are, however, some particular observations on the effectiveness of tie-breaking heuristics in different packing stages.

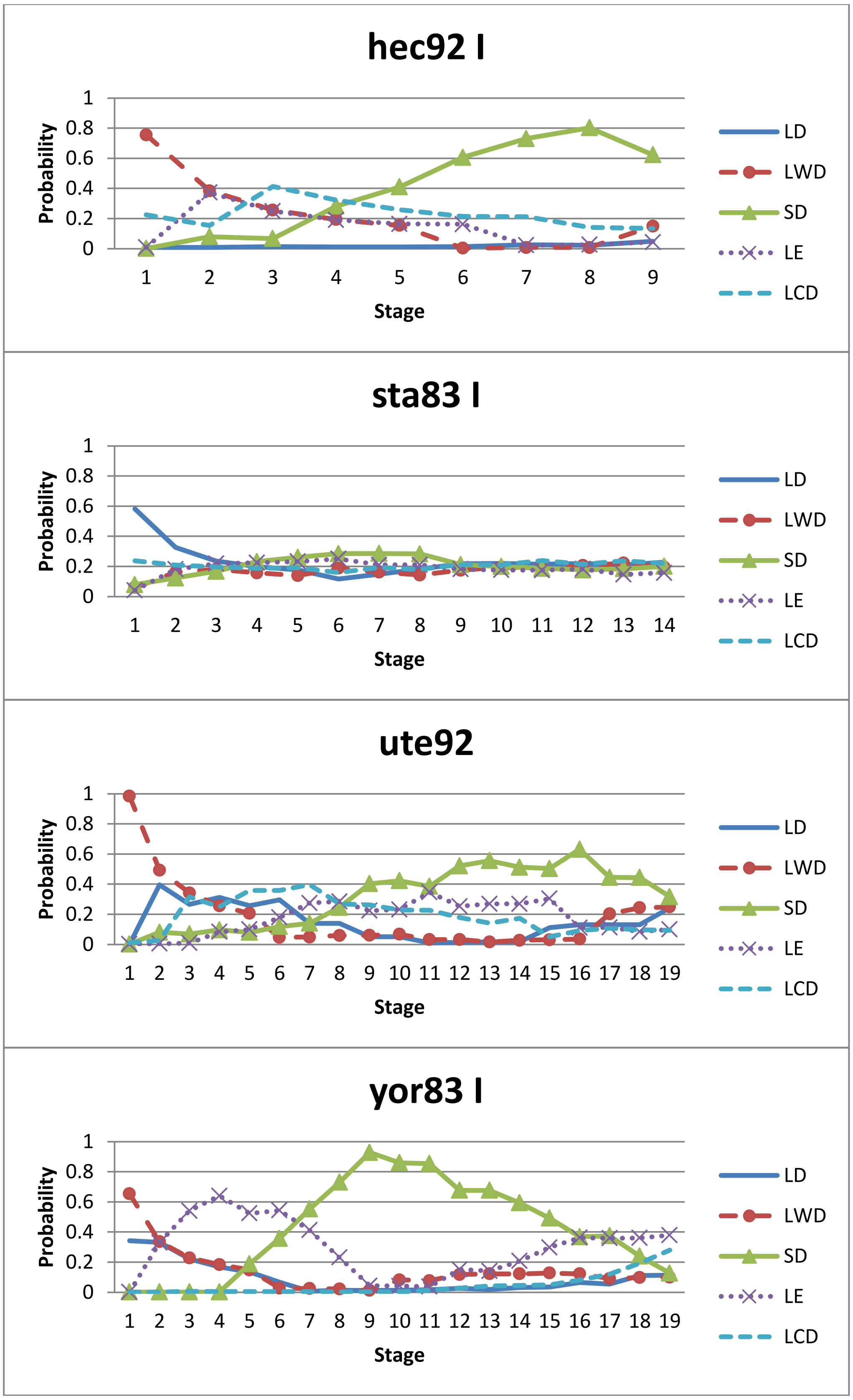


Figure 6.1 Plots of the final probability distribution of low-level heuristics obtained for hec92 I, sta83 I, ute92 and yor83 I

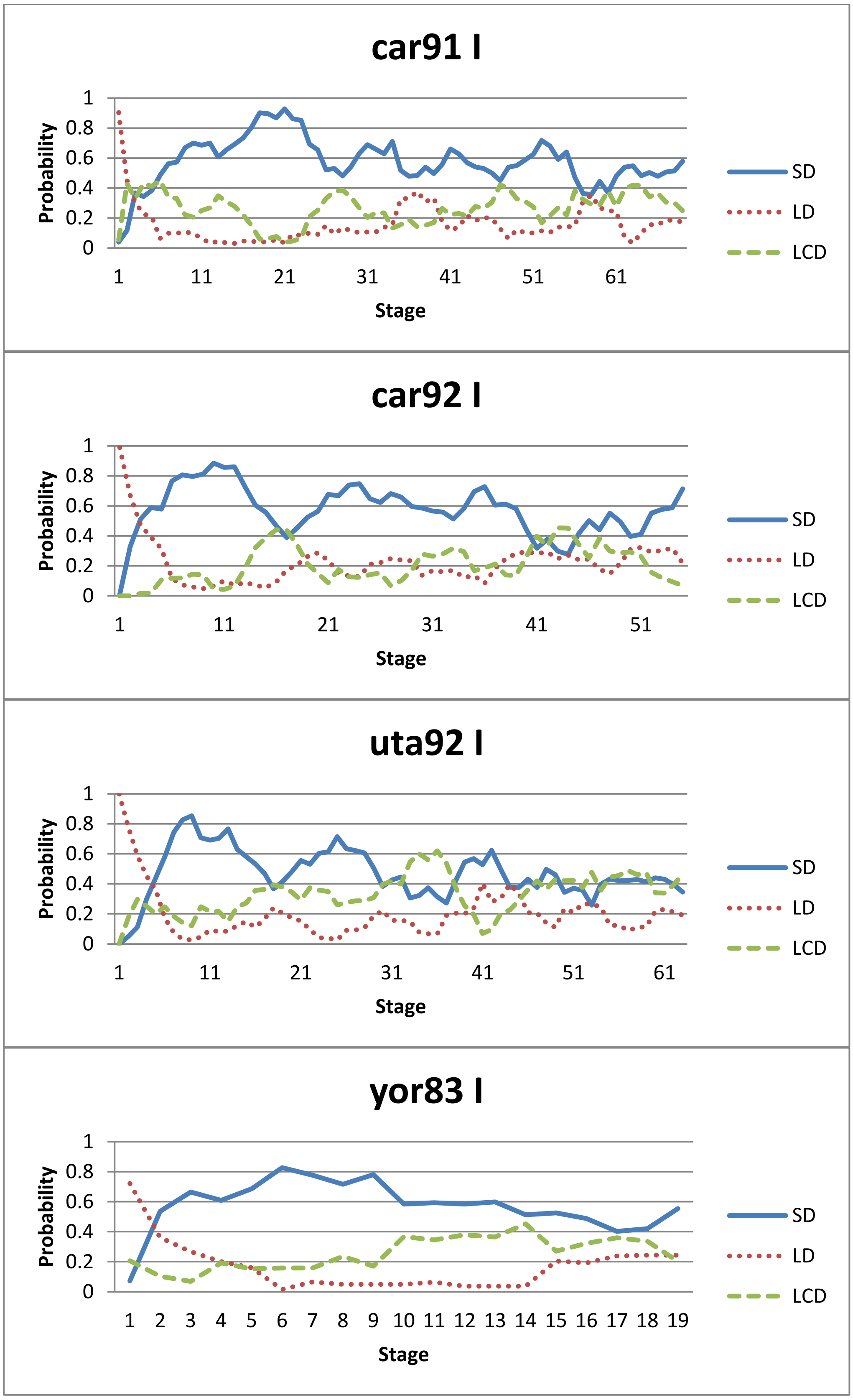


Figure 6.2 Plots of the probability distribution of heuristics at the end of the evolutionary process for the hardest graph colouring instances in the Toronto dataset

For the 3D-SPP-NS (see Figure 6.3):

- *Maximum Contact Area* and *Maximum xz-covering Area* are the two most frequently used heuristics, especially for weakly heterogeneous data.
- *Maximum Furthest_y Surrounding Contact* is least likely to be used.
- *Minimum Wasted Volume* is more likely to be used at the early and late stages of packing.
- *Minimum Rotations with Maximum min_length* is more effective with stronger heterogeneous data.

For the 3D-SPP-WS (see Figure 6.4):

- *Maximum Volume* and *Maximum xz-covering Area* are the two most frequently used heuristics at the early stage, especially for stronger heterogeneous data.
- Similar to the 3D-SPP-NS, *Minimum Rotations with Maximum min_length* is more effective with stronger heterogeneous data.

6.6.4 A Suggestion on Learning Capability of the EDA-HH

We further investigate the EDA-HH to understand its capability in learning. This could help facilitate more intelligent hyper-heuristics in future work to adaptively balance between the performance and computational time demands. The intensification and diversification of the searching process can be adjusted by including effective low-level heuristics or excluding ineffective low-level heuristics.

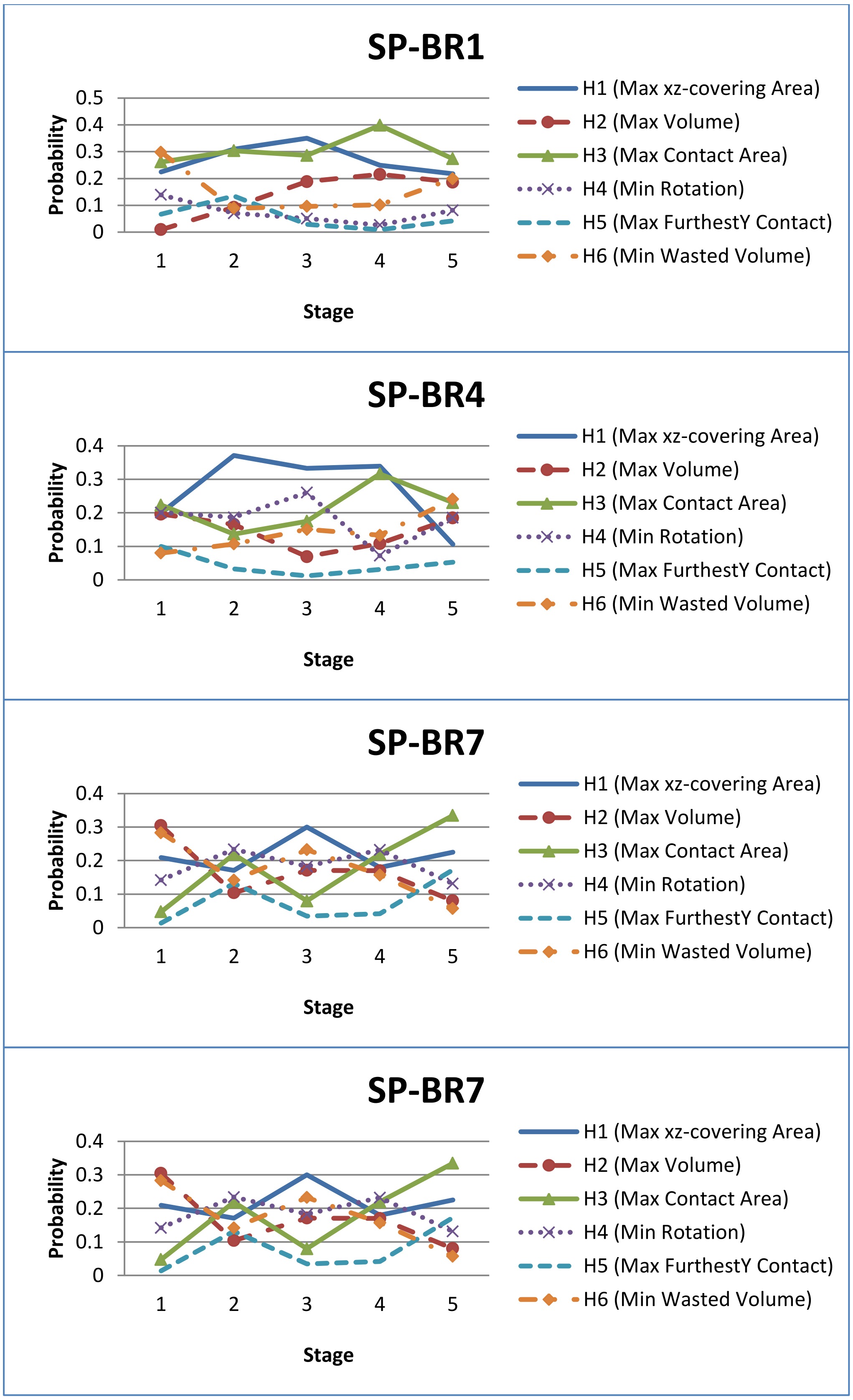


Figure 6.3 Average probability distributions of the first 10 instances for four test cases of the 3D-SPP-NS

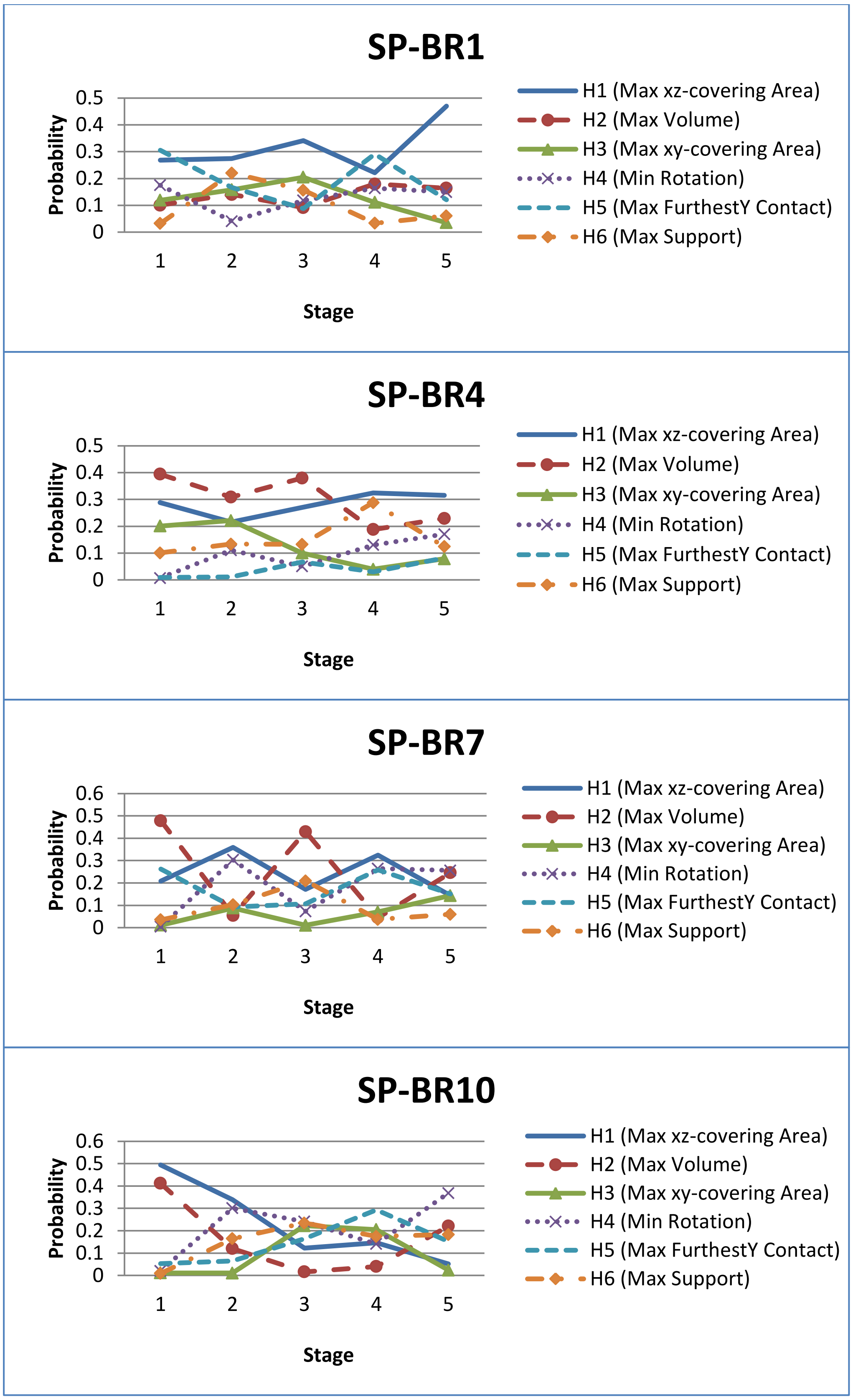
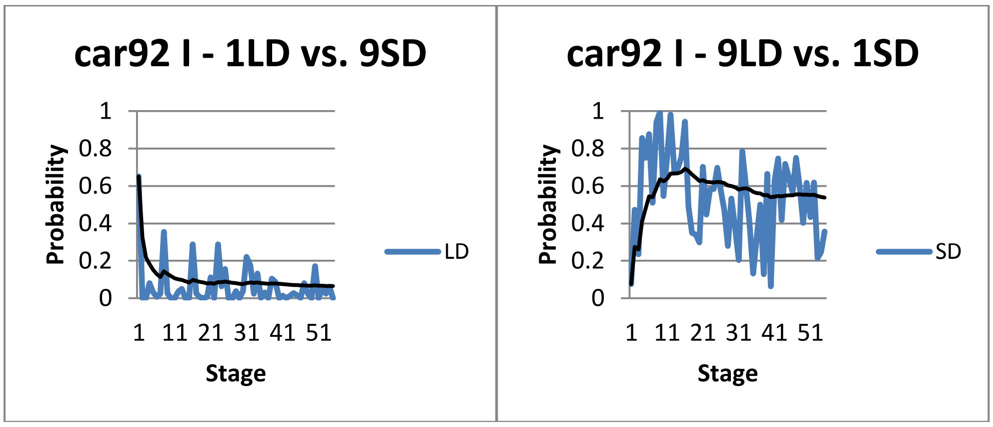


Figure 6.4 Average probability distributions of the first 10 instances for four test cases of the 3D-SPP-WS

Hyper-heuristics on a smaller set of effective low-level heuristics are more likely to achieve better results in a shorter computational time. However, with a larger set of low-level heuristics, hyper-heuristics are likely to explore better solutions eventually at the cost of longer computational time. The issue here lies in the selection of low-level heuristics to be added or removed. Our hyper-heuristic naturally represents a method that is capable of identifying the effectiveness of low-level heuristics by simply examining the obtained probability distribution during or after the evolutionary process. The following two experiments are carried out to examine the hyper-heuristic's ability to learn effective and ineffective heuristics after the evolutionary process. The results can possibly help to better solve other instances which share the same characteristics.

The experiments are conducted on the *car92 I* graph colouring instance with the parameter settings EDA-HH-TOUR12 (1000-2000). In the first experiment, the set of low-level heuristics includes the largest degree heuristic (H1) and nine other heuristics based on the saturation degree heuristic (H3, $H3_2$...$H3_9$). Figure 6.5a shows the probability distribution obtained at the end of the evolutionary process for the largest degree heuristic at each stage. Although the largest degree heuristic is placed into a set of many heuristics based on the saturation degree heuristic, EDA-HH still learned to use it frequently at the very beginning of the colouring process. Similarly, the second experiment is conducted on the pool of nine largest degree based heuristics (H1, $H1_2$...$H1_9$) and one saturation degree heuristic (H3). Even being put into a set of nine largest degree based heuristics, the saturation degree heuristic can still be picked regularly by EDA-HH at the appropriate stages of the colouring process. Figure 6.5b illustrates this phenomenon, especially from stages 4 to 16. In graph colouring, the decisions to select difficult vertices in that early part have a strong influence to the overall colouring.



a b

Figure 6.5 The learning capability of the EDA-HH on the selection of appropriate low-level heuristics at different stages

This promising observation motivates the design of more intelligent hyper-heuristics in our future work.

6.7 Chapter Summary

In this chapter, we developed a simple yet effective EDA-based hyper-heuristic and examined its high generality by applying it to four different problems. With little modifications in the high-level search involving almost no domain-specific information, the hyper-heuristic could provide many competitive results on four investigating problems. Given that there are many optimisation problems in practice which can be solved using constructive methods, this hyper-heuristic can be applied to many other problem domains and represents a cost-effective methodology.

One observation from this chapter is that the option to use constructive approaches in general and this hyper-heuristic in particular for a problem domain is dependent largely on the number of constraints that the problem carries. We see a greater level of success of using our hyper-heuristic on the 3D-SPP-NS and 3D-SPP-WS than on the ETP and the GCP. The difference is from the particular difficulty in designing a good neighbourhood structure for the 3D-SPP. With fewer constraints, effective neighbourhood structures for the GCP and ETP were found, so methodologies concerning local search techniques tend to achieve better results with the GCP and ETP.

In addition to the generality of the hyper-heuristic, we also show the superiority of packing solutions with the improvement techniques proposed in Chapter 5. Last but not least, we demonstrate the capability of our hyper-heuristic in identifying ‘good’ or ‘bad’ heuristics and suggest an approach for future work to better utilise that capability in more intelligent hyper-heuristics.

Chapter 7 Conclusions and Future Work

This chapter summarises the contributions and promising future research directions in this thesis with a focus on constructive approaches. Firstly, we introduce a novel weighted graph model for exam timetabling and based on that, we propose a number of novel constructive heuristics. Secondly, we develop different ways of selecting and combining several constructive heuristics (i.e. sequentially and linearly) at each decision step. The effectiveness of these approaches is demonstrated on the ETP. Thirdly, an extended strategy for the three-dimensional best fit algorithm is introduced as a set up for a hyper-heuristic context. Finally, a hyper-heuristic based on estimation of distribution algorithms is investigated. In particular, the hyper-heuristic demonstrated its high level of abstraction by producing competitive performance on four different problem variants. We also observe the use of probability distributions learned from the hyper-heuristic in identifying 'good' or 'bad' heuristics from a given set of low-level heuristics. These contributions are summarised in the subsequent sections.

7.1 Summary of the Heuristic-Combination Approaches

The first part of this research concerns different approaches to combine constructive graph colouring heuristics to solve the ETP. Apart from the conventional heuristics in the literature e.g. largest weighted degree, or saturation degree, we also examine some novel constructive heuristics based on information provided within the enhanced weighted graph model.

7.1.1 The Enhanced Weighted Graph Model for the ETP

We introduced the enhanced weighted graph model in Section 3.2 – a graph model for exam timetabling whose vertices and edges have several attributes that make it adaptable to a variety of exam timetabling scenarios. Apart from the experiments of the model in this thesis, there are also some notable model features that have not been used. The model can handle pre-coloured vertices, i.e. exams that must be assigned to certain timeslots. Furthermore, we can forbid the assignment of certain timeslots for a particular exam by setting an initial nonzero penalty for the relevant colour. Whilst each timeslot in the Toronto benchmark problems is simply a single number, for different exam timetabling scenarios, timeslots in our model can have attributes associated with general information such as start time, duration and/or finish time.

Some new vertex- and colour-selection heuristics arise naturally from this model, and our implementation allows the use and manipulation of various combinations of them along with or separately from the classical heuristics that have been used for decades. In addition, the model supports the change from a linear combination of heuristics to another linear combination during the colouring process by using a designated switching point.

We also introduce the vertex partitioning technique which is an initialisation step before the colouring starts. It separates vertices with at least a guaranteed non-conflict colour from the remaining vertices (i.e. the hardest subset). This technique has been shown to be useful for the majority of the Toronto benchmark instances. Its major advantage is that the exam-selection process after the hardest subset has been coloured can be based solely on soft constraint penalty, thus possibly leading to better solutions.

7.1.2 The Heuristic-Combination Approaches for the ETP

We investigate two different strategies in Chapter 4 to combine heuristics, i.e. linearly (LCS) and sequentially (SCS), within the enhanced weighted graph model. It is clearly the case that the LCS is the generalisation of the SCS. The tie-breaker effect in the SCS can be achieved by setting one weight to be much larger than the other in the LCS. The weights of the linear combinations define specific roles that each simple heuristic contributes to the process of ordering vertices.

The effectiveness of the newly proposed constructive heuristics from the enhanced weighted graph model compared to the conventional heuristics is justified by experiments on different linear combinations. We presented a specific selection of heuristics and weight settings which have shown competitive results compared to many other constructive approaches in the literature. The main idea is to combine, with different weights, the two newly proposed heuristics - maximum number of bad colours (heuristic 3) and maximum number of bad-intersect edges (heuristic 7). Whilst the the SCS results are on average 13% worse than the best constructive results reported in the literature for the Toronto dataset, the LCS can significantly outperform the SCS and improve on the best reported constructive results in four instances.

7.2 Summary of the EDA-based Hyper-heuristic

The second part of this research focuses on raising the level of generality for search methodologies by investigating the use of an estimation of distribution algorithm within a hyper-heuristic context to solve four different optimisation problems. The modifications made in the high-level search are limited and involve almost no domain-specific information. The results obtained by the EDA-HH on all four problems are competitive. In practice, many optimisation problems can be solved using constructive methods and such constructive processes typically can be divided into stages. Therefore, the EDA-HH is cost-effective and partly contributes to the direction of designing search methodologies of higher generality than other search systems based on meta-heuristics in the literature.

The EDA-HH also represents a mechanism to learn the effectiveness of a given set of heuristics by simply examining the probability distribution of heuristics learnt during/after the evolutionary process. Observations on the effectiveness of low-level heuristics for several datasets of the investigated problems have been presented in Chapter 6.

We also observe the relation in solution quality produced by our constructive hyper-heuristic and the number of constraints in the investigated problems. The level of success of using our hyper-heuristic on the 3D-SPP-NS and 3D-SPP-WS has been seen to be higher than on the ETP and the GCP. The difference is from the particular difficulty in designing a good neighbourhood structure for local search-based methods for the 3D-SPP. The conclusion is that for problems with a significantly high number of constraints, constructive methods represent a better alternative compared to iterative methods.

7.2.1 The Extended 3BF Strategy for the 3D-SPP

An extended strategy for the three-dimensional best-fit algorithm is introduced in Chapter 5 with the aim of improving the suitability when it is applied within a hyper-heuristic context. We proposed variants of gap-filling heuristics, new tie-breaking heuristics, an adjustment technique for box positioning, and a procedure to select assembled boxes.

The improved 3BF strategy represents an efficient approach to construct packing solutions. The results obtained show that it is as good as the original 3BF in terms of finding high quality solutions (see Section 5.3). In addition, the extended 3BF offers more diversification in finding good combinations of different heuristics. In a hyper-heuristic context, using suitable combinations of gap-filling and tie-breaking heuristics, larger areas in the solution space can be explored compared to the original 3BF. The four new tie-breaking heuristics are proposed based on the common ideas of human in packing to accommodate boxes in different situations. Techniques to increase the compactness of packing solutions (i.e. the use of assembled boxes, the technique to slide a box around its position, and the tower processing) have been shown to be effective.

7.2.2 The Results of the EDA-HH on the ETP

The EDA-HH is applied to the 13 instances of Toronto benchmark dataset on both the exam timetabling and the graph colouring domains in Section 6.5.1. For the exam timetabling domain, we compare our EDA-HH with other hyper-heuristics tested on the same benchmark. The EDA-HH generalises well over all exam timetabling instances compared to other constructive hyper-heuristic approaches. It outperforms others on 12 instances. We also reduce the gap between the results obtained from hyper-heuristic approaches and the best results from all approaches in the literature. Given that there is no local improvement and backtracking involved, we found our results encouraging. For the graph colouring domain (the ETP with only a hard constraint), the EDA-HH also shows promising results by obtaining newly best reported colourings in four hard instances. Note that the settings for the high-level search are the same for both investigated domains.

7.2.3 The Results of the EDA-HH on the 3D-SPP

The EDA-HH is applied to the SP-BR dataset for both the 3D-SPP-NS and the 3D-SPP-WS in Section 6.5.2. For the 3D-SPP-NS, given the same running time of 160 seconds, our EDA-HH can clearly improve on all test cases of the SP-BR dataset, i.e. an increase of 1.9% from the best reported average volume utilisation in the literature. The EDA-HH also generalises well for large instances of the SP-BR-XL dataset with an increase of 1.2% from the average volume utilisation obtained by other approaches. We also demonstrate the capability of our EDA-HH if the running time is not restricted. The results obtained in that case is very competitive compared the other approaches in the literature (see Section 6.5.2).

7.3 Future Work

The results obtained in this thesis encourage us to further develop more advanced and powerful algorithms in future research.

For the heuristic-combination approaches for the ETP on the enhanced weighted graph model there are several directions emerging:

- Further testing of the effectiveness of switching from one linear combination of heuristics to another during the colouring. One goal here would be to identify certain problem characteristics that would determine which weights to use.
- Reducing the sensitivity of the discrete-valued, threshold-based primitive heuristics by designing new continuous-valued analogues. For example, instead of counting an edge as either *bad* or not, according to whether its weight exceeds a threshold, count it as 1 towards the *badness degree* if it exceeds the threshold and if it does not, count the fraction of its weight over the threshold.
- Analysing the landscape of the search space of threshold parameters (*pc* and *ie*) in order to reduce the computational time in finding the best sets of parameter settings.

- We can design algorithms with a feedback loop that automatically adjust the parameters for the switching point, thresholds and weight vectors of linear combinations to suitable settings based on the algorithm's past performance. Also, if the number of primitive heuristics can be reduced, the dimension of the corresponding search space in the context of hyper-heuristics becomes more tractable.

- Adding a backtracking component to the algorithm is likely to reduce the total proximity penalty. A similar backtracking procedure as in (Carter et al., 1996) can be tested. Another approach is to check when a colour assignment for a selected vertex incurs a proximity penalty above some threshold, the algorithm un-colours or re-colours some other vertex or vertices in order to reduce the selected vertex's proximity penalty.

- Designing an improvement method that takes a given colouring produced by our algorithm and looks for vertices whose colours can be changed to decrease the total proximity penalty while maintaining feasibility.

- Adding a random factor into the vertex and colour selection processes and compare further the efficiency of the novel vertex-selection heuristics (e.g. max bad-intersect edges and max bad-colours) with the corresponding traditional heuristics (e.g. largest weighted degree and saturation degree).

For the EDA-based hyper-heuristic, the following directions are promising for future research:

- Integrating simple backtracking or local improvement into the evolutionary process to further improve the results.

- Examining a wider range of parameter settings including population size, different selection mechanism, parameters for selection mechanism, and stage settings, etc.

- Finding or designing other 3D strip packing datasets and applying EDA-HH on them.

- Designing and investigating more intelligent hyper-heuristics e.g. a hyper-heuristic that adjusts the intensification and diversification in the high-level search by removing or adding low-level heuristics.

- Implementing more complex estimation of distribution algorithms on the high-level search that take into account the dependency between stages.

Appendix A - Explanation and Derivation of the Expected Value *ev*

In this thesis, *ev* represents the expected value of the proximity weight associated with two different colours randomly chosen from a set of *x* colours. This mathematical calculation is the weighted average of all possible proximity weights arising from pairs of colours, taking into account the size and frequency of each possible proximity weight. The product (*ev*)*(avgIntersectionSize) is a measure of the contribution to the total proximity penalty that a randomly selected edge with randomly coloured endpoints makes. Both of these quantities are completely determined by the problem instance, and what we regard as a bad proximity penalty for assigning a given colour to a given vertex depends on the value of this product. In particular, our bad-proximity threshold is directly proportional to this product, where the multiplier *pc* is the constant of proportionality. Several of the results reported in this paper were obtained by experimenting with different values of *pc*.

NOTATION: Let *proxWt(c1,c2)* denote the proximity weight for two different colours, *c1* and *c2*. For the Toronto problems, *proxWt(c1, c2)* $2^{5-|c1-c2|}$ if $0 < |c1 - c2| <= 5$ and $= 0$ *if* $|c1 - c2| > 5$.

<u>Theorem</u>: Let *c1* and *c2* be two different colours chosen randomly from a set of *x* colours, and let *ev* be the expected value of *proxWt(c1, c2)*. Then

$$ev = \frac{62x - 114}{x(x-1)}$$

Proof: First observe that the total number of possible pairs of different colours = $\binom{x}{2} = \frac{x(x-1)}{2}$. Next we count the number of pairs of colours having each possible proximity weight.

There are *x-1* pairs of colours having *proxWt = 16*, namely {*1,2*}, *{2,3}, ..., {x-1, x}*.

There are *x-2* pairs of colours having *proxWt = 8*, namely *{1,3}, {2,4}, ..., {x-2, x}*.

There are *x-3* pairs of colours having *proxWt = 4*, namely *{1,4}, {2,5}, ..., {x-3, x}*.

There are *x-4* pairs of colours having *proxWt = 2*, namely *{1,5}, ..., {x-4, x}*.

There are *x-5* pairs of colours having *proxWt = 1*, namely *{1,6}, ..., {x-5, x}*.

All other pairs of colours have *proxWt = 0*.

ev is the weighted average of all these *proxWt* values. Thus,

$$ev = \frac{16(x-1)+8(x-2)+4(x-3)+2(x-4)+(x-5)}{\left(\frac{x(x-1)}{2}\right)} = \frac{62x-114}{x(x-1)}$$

Appendix B - A Simplified Implementation of the Minimum Wasted Volume Heuristic (TB7)

Most of the tie-breaking heuristics in Table 5.2 are trivial to implement and not the subject of this paper. Among the four novel tie-breaking heuristics (marked with * in Table 5.2), the *Minimum Wasted Volume* heuristic *TB7* requires the highest computational complexity. However, it deals with the situation as shown in Figure 5.4 more efficiently. In the context of hyper-heuristics, each of these tie-breaking heuristics will be called many times. Therefore, an efficient implementation is necessary.

We firstly define the *colliding distance* of a point *A* on a particular box's surface *S*. Suppose that we have a vector *V* with the initial point *A* and being perpendicular to *S*. The direction of the vector is towards the corresponding container's wall (e.g. if *S* is on the left surface of the box, the direction of *V* will be towards the container's left wall). The terminal point of *V* will be the first point that the vector collides with another box surface or the container wall. The colliding distance of point *A* on surface *S* is represented by the length of *V*. If the colliding distance is smaller than the corresponding minimum box size (i.e. if *S* is the left or right surface, the corresponding minimum box size will be *minBoxSizeX*; if *S* is the top or bottom surface, the corresponding minimum box size will be *minBoxSizeZ*), it will represent a wasted distance. Typically, the wasted volume towards a surface of a candidate box is calculated by checking the colliding distances of all points on that surface and adding up the wasted distances (e.g Figure B.1a). However, that requires significant running times to check each box. We simplify the implementation of this tie-breaking heuristic by defining an *n*×*n grid* on a box surface. For the considered surfaces of a candidate placement, we check and add up its wasted distances only from the intersecting points on the *n×n* grid. An example of a *3×3* grid on the right surface of a candidate placement is shown in Figure B.1b.

For the 3D-SPP-NS, the considered surfaces include the left, right, top, and bottom surfaces. This simplification uses a greedy method to suggest boxes that are likely to produce the minimal wasted volume for a candidate gap. It might not choose the same box as in the typical implementation. For example, the simplification might choose the box in Figure B.1d instead of the box with smaller wasted volume in Figure B.1b. However, in many situations, using a small grid still produces the same box selections (i.e. in this example, the box in Figure B.1c is correctly selected as having the minimum wasted volume) while the computational time is reduced significantly. For the 3D-SPP-WS with the stability constraint, this tie-breaking heuristic is not applied.

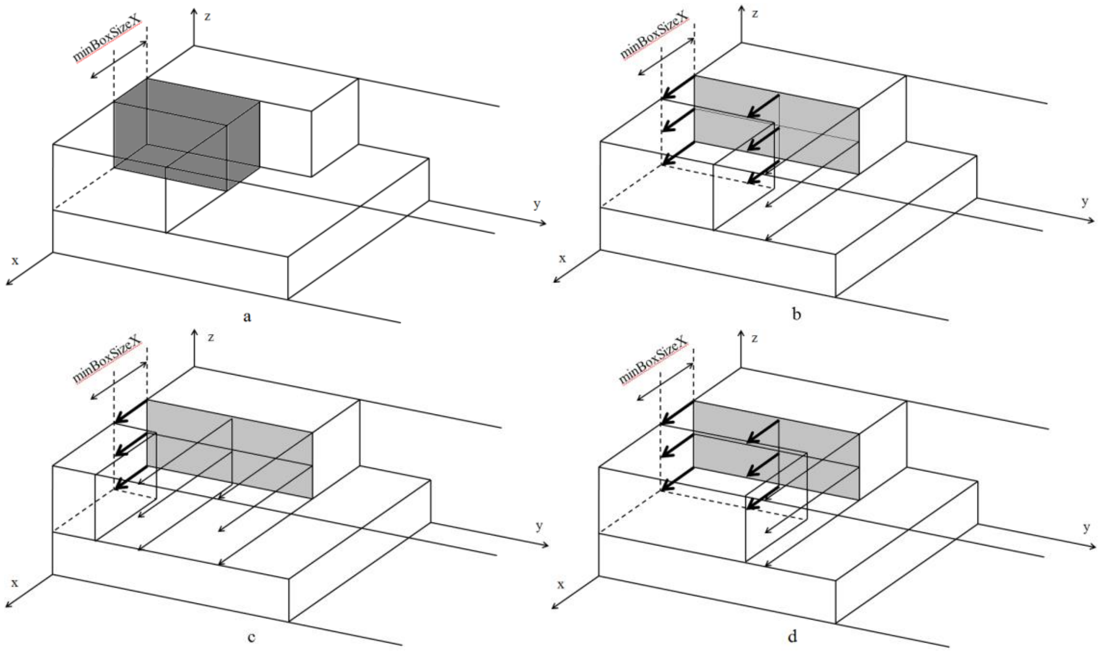


Figure B.1 A *3×3* grid to calculate the colliding distances of the right surface of a candidate box placement. If the sizes of a considering surface are not divisible by the size of the grid, the quotient will be truncated to integer value. The length of bold arrows indicates the wasted distance

Bibliography

AARTS, E. H. L. & KORST, J. 1989. *Simulated Annealing and Boltzmann Machines,* New York, John Wiley & Sons.

AARTS, E. H. L., KORST, J. & MICHIELS, W. 2005. Simulated annealing. *In:* BURKE, E. K. & KENDALL, G. (eds.) *Introductory Tutorials in Optimisation, Decision Support and Search Methodology.* Springer.

ABDUL-RAHMAN, S., BARGIELA, A., BURKE, E. K., ÖZCAN, E. & MCCOLLUM, B. 2009. Construction of examination timetables based on ordering heuristics. Proceedings of the 24th International Symposium on Computer and Information Sciences, 727-732.

ABDUL-RAHMAN, S., BARGIELA, A., BURKE, E. K., ÖZCAN, E. & MCCOLLUM, B. 2011a. Adaptive Linear Combination of Heuristic Orderings for Constructing Examination Timetable. *Accepted to European Journal of Operational Research.*

ABDUL-RAHMAN, S., BURKE, E. K., BARGIELA, A., MCCOLLUM, B. & ÖZCAN, E. 2011b. A constructive approach to examination timetabling based on adaptive decomposition and ordering. *Annals of Operational Reseach***,** 1-19.

AHMADI, S., BARRONE, P., CHENG, P., BURKE, E. K., COWLING, P. & MCCOLLUM, B. 2003. Pertubation based variable neighbourhood search in heuristic space for examination timetabling problem. Proceedings of Multidisciplinary International Scheduling: Theory and Applications (MISTA 2003), 155-171.

ALLEN, S. D., BURKE, E. K. & KENDALL, G. 2011. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research,* 209(3)**,** 219-227.

ASMUNI, H., BURKE, E. K. & GARIBALDI, J. M. 2005. Fuzzy multiple ordering criteria for examination timetabling. *In:* BURKE, E. K. & TRICK, M., eds. Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Springer, 334-353.

ASMUNI, H., BURKE, E. K., GARIBALDI, J. M. & MCCOLLUM, B. 2007. Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables. Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications Conference (MISTA 2007), 59-66.

ASMUNI, H., BURKE, E. K., GARIBALDI, J. M., MCCOLLUM, B. & PARKES, A. 2009. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research,* 36(4)**,** 981-1001.

BACARDIT, J., STOUT, M., HIRST, J. D., SASTRY, K., LLORÀ, X. & KRASNOGOR, N. 2007. Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. *Genetic and Evolutionary Computation Conference (GECCO-2007).*

BAKER, B. S., COFFMAN, E. G., JR. & RIVEST, R. L. 1980. Orthogonal packing in two dimensions. *SIAM Journal of Computing,* 9**,** 846-855.

BALUJA, S. 1994. Population-based incremental learning: A method for integrating genetic

search based function optimization and competitive learning. *Technical Report CMU-CS-94-163.* Pittsburgh, PA.

BALUJA, S. & DAVIES, S. 1997. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Proceedings of the International Conference on Machine Learning, 30-38.

BATTITI, R. & PROTASI, M. 2001. Reactive local search for the maximum clique problem. *Algorithmica,* 29(4)**,** 610-637.

BILGIN, B., ÖZCAN, E. & KORKMAZ, E. E. 2007. An Experimental Study on Hyper-heuristics and Exam Timetabling. *In:* BURKE, E. K. & RUDOVA, H., eds. Selected Papers from the 6th International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, 394-412.

BISCHOFF, E. E. & MARIOTT, M. D. 1990. A comparative evaluation of heuristics for container loading. *European Journal of Operational Research,* 44**,** 267-276.

BISCHOFF, E. E. & RATCLIFF, M. S. W. 1995. Issues in the development of approaches to container loading. *Omega,* 23**,** 377-390.

BORTFELDT, A. & GEHRING, H. 1999. Two metaheuristics for strip packing problems. *In:* DESPOTIS, D. K. & ZOPOUNIDIS, C., eds. Proceedings of the Fifth International Conference of the Decision Sciences Institute, Athens, 1153-1156.

BORTFELDT, A. & MACK, D. 2007. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research,* 183(3)**,** 1267-1279.

BRELAZ, D. 1979. New methods to color the vertices of a graph. *Communication of the ACM,* 22(4)**,** 251-256.

BREMERMANN, H. 1962. Optimisation through evolution and recombination. *In:* YOVITS, M., JACOBI, G. T. & GOLDSTINE, G. (eds.) *Self-Organising Systems.* Washington DC, Spartan Books.

BULLNHEIMER, B. 1998. An examination scheduling model to maximize students study time. *In:* BURKE, E. K. & CARTER, M. W. (eds.) *Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference. Lecture Notes in Computer Science.* Springer.

BURKE, E. K. & BYKOV, Y. 2008. A Late Acceptance Strategy in Hill-Climbing for Examination Timetabling Problems. Proceedings of Practice and Theory of Automated Timetabling (PATAT 2008).

BURKE, E. K., BYKOV, Y., NEWALL, J. P. & PETROVIC, S. 2004a. A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36(6)**,** 509-528.

BURKE, E. K., DROR, M., PETROVIC, S. & QU, R. 2005. Hybrid Graph Heuristics within a Hyper-heuristic Approach to Exam Timetabling Problems. *In:* GOLDEN, B. L., RAGHAVAN, S. & WASIL, E. A., eds. The Next Wave in Computing, Optimization, and Decision Technologies. Conference Volume of the 9th informs Computing Society Conference, Springer, 79-91.

BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E. & QU, R. 2012. Hyper-heuristics: A Survey of the State of the Art. *to appear in the Journal of the Operational Research Society*.

BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E. & WOODWARD, J. 2009. A Classification of Hyper-heuristics Approaches. *In:* GENDREAU, M. & POTVIN, J. Y. (eds.) *Handbook of Metaheuristics, International Series in Operations Research & Management Science.* Springer (in press).

BURKE, E. K., JACKSON, K., KINGSTON, J. H. & WEARE, R. P. 1997. Automated university timetabling: The state of the art. *The Computer Journal,* 40(9)**,** 565-571.

BURKE, E. K. & KENDALL, G. 2005. *Search Methodologies: Introductory Tutorial in Optimization and Decision Support Techniques,* Springer.

BURKE, E. K., KENDALL, G., MISIR, M. & ÖZCAN, E. 2010. Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*.

BURKE, E. K., KENDALL, G., NEWALL, J. P., HART, E., ROSS, P. & SCHULENBURG, S. 2003. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. *In:* GLOVER, F. & KOCHENBERGER, K. (eds.) *Handbook of Meta-Heuristics.* Kluwer Academic Publishers.

BURKE, E. K., KENDALL, G. & WHITWELL, G. 2004b. A new placement heuristic for the orthogonal stock cutting problem. *Journal of Operations Research,* 52(4)**,** 655-671.

BURKE, E. K., KINGSTON, J. H. & DE WERRA, D. 2004c. Applications to timetabling. *In:* GROSS, J. & YELLEN, J. (eds.) *The Handbook of Graph Theory*. Chapman Hall/CRC Press.

BURKE, E. K. & LANDA SILVA, J. D. 2004. The design of memetic algorithms for scheduling and timetabling problems. *In:* HART, E., KRASNOGOR, N. & SMITH, J. E. (eds.) *Recent Advances in Memetic Algorithms and Related Search Technologies. Studies in Fuzziness and Soft Computing.* Berlin, Heidelberg, NewYork, Springer.

BURKE, E. K., MCCOLLUM, B., MEISELS, A., PETROVIC, S. & QU, R. 2007. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research,* 176**,** 177-192.

BURKE, E. K. & NEWALL, J. P. 1999. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation,* 3(1)**,** 63-74.

BURKE, E. K. & NEWALL, J. P. 2003. Enhancing timetable solutions with local search methods. *In:* BURKE, E. K. & DE CAUSMAECKER, P. (eds.) *Practice and Theory of Automated Timetabling: Selected Papers from the 4th International Conference. Lecture Notes in Computer Science.* Springer.

BURKE, E. K. & NEWALL, J. P. 2004. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operational Reseach,* 129**,** 107-134.

BURKE, E. K., NEWALL, J. P. & WEARE, R. P. 1996. A memetic algorithm for university exam timetabling. *In:* BURKE, E. K. & ROSS, P., eds. Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference. Springer Lecture Notes in Computer Science, 241-250.

BURKE, E. K., NEWALL, J. P. & WEARE, R. P. 1998a. Initialization strategies and diversity in evolutionary timetabling. *Evolutionary Computation,* 6(1)**,** 81-103.

BURKE, E. K., NEWALL, J. P. & WEARE, R. P. 1998b. A simple heuristically guided search for the timetable problem. Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS98), 574-579.

BURKE, E. K. & PETROVIC, S. 2002. Recent research directions in automated timetabling. *European Journal of Operational Research,* 140(2)**,** 266-280.

BURKE, E. K., PETROVIC, S. & QU, R. 2006. Case based heuristic selection for timetabling problems. *Journal of Scheduling,* 9(2)**,** 115-132.

CARAMIA, M., DELLOLMO, P. & ITALIANO, G. F. 2001. New algorithms for examination timetabling. *In:* NAHER, S. & WAGNER, D., eds. Algorithm Engineering 4th International Workshop, Proceedings WAE 2000. Lecture Notes in Computer Science, 230-241.

CARAMIA, M., DELLOLMO, P. & ITALIANO, G. F. 2008. Novel local search-based approaches to university examination timetabling. *INFORMS Journal of Computing,* 20(1)**,** 86-99.

CARTER, M. W. 1986. A survey of practical applications of examination timetabling algorithms. *Operations Research,* 34(2)**,** 193-202.

CARTER, M. W. & JOHNSON, D. S. 2001. Extended clique initialisation in examination timetabling. *Journal of Operational Research Society (JORS),* 52**,** 538-544.

CARTER, M. W. & LAPORTE, G. 1996. Recent developments in practical examination timetabling. *In:* BURKE, E. K. & ROSS, P., eds. Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference. Lecture Notes in Computer Science, Springer, 3-21.

CARTER, M. W., LAPORTE, G. & LEE, S. Y. 1996. Examination timetabling: Algorithmic strategies and applications. *Journal of Operational Research Society (JORS),* 47(3)**,** 373-383.

CASEY, S. & THOMPSON, J. 2003. GRASPing the examination scheduling problem. In: Practice and Theory of Automated Timetabling. *In:* BURKE, E. K. & DE CAUSMAECKER, P. (eds.) *Lecture Notes in Computer Science.* Springer.

CHAZELLE, B. 1983. The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Transactions on Computers,* c32(8)**,** 697-707.

CHEN, C.-H. & P. CHEN, Y. 2007. Real-coded ECGA for economic dispatch. *Genetic and Evolutionary Computation Conference (GECCO-2007).*

CORNE, D. & ROSS, P. 1995. Comparing genetic algorithms, simulated annealing and stochastic hill climbing on timetabling problems. *In:* FOGARTY, T. E. (ed.) *Evolutionary Computing, Lecture Notes in Computer Science*

CORR, P. H., MCCOLLUM, B., MCGREEVY, M. A. J. & MCMULLAN, P. 2006. A New Neural Network Based Construction Heuristic for the Examination Timetabling Problem. *International Conference on Parallel Problem Solving From Nature (PPSN) 2006.* Reykjavik, Iceland.

COTE, P., WONG, T. & SABOURI, R. 2005. Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. *In:* BURKE, E. K. & TRICK, M. (eds.) *Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science.* Springer.

COWLING, P., KENDALL, G. & SOUBEIGA, E. 2000. A Hyper-heuristic Approach to Scheduling a Sales Summit. *In:* BURKE, E. K. & ERBEN, W., eds. Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference. Lecture Notes in Computer Science.

CROWSTON, W. B., GLOVER, F., THOMPSON, G. L. & TRAWICK, J. D. 1963. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA.* Carnegie Mellon University, Pittsburgh.

CSIRIK, J. & WOEGINGER, G. J. 1996. On-line packing and covering problems. Online algorithms, Springer Lecture Notes in Computer Science, 147-177.

CULBERSON, J. C. 1992. Iterated Greedy Graph Coloring and the Difficult Landscape. *Technical Report [1992-07].* Department Computer Science, University of Alberta, Canada.

DAWKINS, R. 1990. *The Selfish Gene*, Oxford University Press.

DE BONET, J. S., ISBELL, C. L. & VIOLA, P. 1997. MIMIC: Finding optima by estimating probability densities. *Advances in neural information processing systems (NIPS-97),* 9**,** 424-431.

DE JONG, K. 1975. *An analysis of the behaviour of a class of genetic adaptive systems. PhD Thesis.* University of Michigan.

DENZINGER, J., FUCHS, M. & FUCHS, M. 1996. High performance ATP systems by combining several AI methods. *Technical Report SEKI-Report SR-96-09.* University of Kaiserslautern.

DI GASPERO, L. 2002. Recolour, shake and kick: A recipe for the examination timetabling problem. *In:* BURKE, E. K. & CAUSMAECKER, D., eds. Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling, KaHo St.-Lieven, Gent, Belgium, 404-407.

DI GASPERO, L. & SCHAERF, A. 2001. Tabu search techniques for examination timetabling. *In:* BURKE, E. K. & ERBEN, W., eds. Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference. Springer Lecture Notes in Computer Science, 104-117.

DORNDORF, U. & PESCH, E. 1995. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research,* 22(1)**,** 25-40.

DOWSLAND, K. 1995. Simulated annealing. *In:* REEVES, C. R. (ed.) *Modern heuristics techniques for combinatorial problems.* McGraw Hill.

DU, J., KORKMAZ, E., ALHAJJ, R. & BARKER, K. 2004. Novel Clustering Approach that Employs Genetic Algorithm with New Representation Scheme and Multiple Objectives. Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery, Zaragoza, Spain.

DUCHEYNE, E., DE BAETS, B. & DE WULF, R. 2004. Probabilistic models for linkage learning in forest management. *In:* JIN, Y. (ed.) *Knowledge incorporation in evolutionary computation.* Springer.

DUECK, G. 1993. New optimization heuristics: the great deluge and the record-to-record travel. *Journal of Computational Physics,* 104**,** 86-92.

ERBEN, W. 2001. A grouping genetic algorithm for graph colouring and exam timetabling. *In:* BURKE, E. K. & ERBEN, W. (eds.) *Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference. Lecture Notes in Computer Science.* Springer.

ERSOY, E., ÖZCAN, E. & UYAR, Ş. 2007. Memetic Algorithms and Hyperhill-climbers. *In:* BAPTISTE, P., KENDALL, G., KORDON, A. M. & SOURD, F., eds. Proceedings of the 3rd Multidisciplinary International Conference On Scheduling: Theory and Applications, Paris, France, 159-166.

ETXEBERRIA, R. & LARRAÑAGA, P. 1999. Global optimization using Bayesian networks. *In:* OCHOA, A., SOTO, M. R. & SANTANA, R., eds. Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99), Editorial Academia, 151-173.

FANG, H., ROSS, P. & CORNE, D. 1993. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. *In:* FORREST, S., ed. Fifth International Conference on Genetic Algorithms, San Meteo, Morgan Kaufmann, 375-382.

FANG, H., ROSS, P. & CORNE, D. 1994. A promising hybrid ga/heuristic approach for open-shop scheduling problems. *In:* COHN, A., ed. Eleventh European Conference on Artificial Intelligence, John Wiley & Sons.

FISHER, H. & THOMPSON, G. L. 1961. Probabilistic learning combinations of local job-shop scheduling rules. *Factory Scheduling Conference.* Carnegie Institue of Technology.

FISHER, H. & THOMPSON, G. L. 1963. Probabilistic learning combinations of local job-shop scheduling rules. *In:* MUTH, J. F. & THOMPSON, G. L. (eds.) *Industrial Scheduling.* New Jersey, Prentice Hall, Inc.

FRASER, A. 1957. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science,* 10**,** 484-491.

GALINIER, P. & HAO, J. K. 1999. Hybrid Evolutionary Algorithms for Graph Colouring. *Journal of Combinatorial Optimization,* 3**,** 379-397.

GAREY, M. R. & JOHNSON, D. S. 1979. *Computers and intractability - a guide to NP-completeness,* San Francisco, W.H. Freeman and Company.

GARRIDO, P. & RIFF, M. C. 2007a. Collaboration between hyperheuristics to solve strip packing problems. Proceedings of the 12th International Fuzzy Systems Association World Congress, Springer, 698-707.

GARRIDO, P. & RIFF, M. C. 2007b. An evolutionary hyperheuristic to solve strip-packing problems. Proceedings of Intelligent Data Engineering and Automated Learning (IDEAL 2007), Springer, 406-415.

GEORGE, J. A. & ROBINSON, D. F. 1980. A heuristic for packing boxes into container. *Computers and Operations Research,* 7**,** 147-156.

GLOVER, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research,* 13**,** 533-549.

GLOVER, F. & KOCHENBERGER, K. 2003. *Handbook of Meta-heuristics,* Kluwer.

GLOVER, F. & LAGUNA, M. 1993. Tabu search. *In:* REEVES, C. R. (ed.) *Modern Heuristic Techniques for Combinatorial Problems.* Oxford, Scientific Publications.

GLOVER, F. & LAGUNA, M. 1997. *Tabu search,* Boston, Kluwer Academic Publishers.

GOLDBERG, D. E. 1989. *Genetic algorithms in search, optimisation, and machine learning,* Reading, MA, Addison-Wesley.

GOLDBERG, D. E., KORB, B. & DEB, K. 1990. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems,* 3(5)**,** 493-530.

HARIK, G. 1999. Linkage learning via probabilistic modeling in the ECGA. *IlliGAL Report No. 99010.* University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

HARIK, G. R., LOBO, F. G. & GOLDBERG, D. E. 1997. The compact genetic algorithm. *In:* 97006, I. R. N. (ed.). University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

HART, E. & ROSS, P. 1998. A heuristic combination method for solving job-shop scheduling problems. *In:* EIBEN, A. E., BACK, T., SCHOENAUER, M. & SCHWEFEL, H. P., eds. Parallel Problem Solving from Nature V, Lecture Notes in Computer Science, Springer-Verlag, 845-854.

HART, E., ROSS, P. & NELSON, J. A. D. 1998. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computing,* 6(1)**,** 61-80.

HAUSCHILD, M. & PELIKAN, M. 2011. An Introduction and Survey of Estimation of Distribution Algorithms. *In:* 2011004, M. R. N. (ed.). Missouri Estimation of Distribution Algorithms Laboratory.

HOLLAND, J. 1975. *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Harbor.

HOPPER, E. 2000. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods.* PhD Thesis, Cardiff University.

KARABULUT, K. & INCEOGLU, M. M. 2004. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. *ADVIS***,** 441-450.

KENDALL, G. & HUSSIN, N. M. 2005a. An Investigation of a tabu search based hyperheuristic for examination timetabling. *Selected papers from Multidisciplinary Scheduling; Theory and Applications.*

KENDALL, G. & HUSSIN, N. M. 2005b. A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. *In:* BURKE, E. K. & TRICK, M. (eds.) *Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science.* Springer.

KIRKPATRICK, S., GELATT, C. D. & VECCHI, M. P. 1983. Optimisation by simulated annealing. *Science,* 220**,** 671-680.

KOLLAT, J. B., REED, P. M. & KASPRZYK, J. R. 2008. A new epsilon-dominance hierarchical Bayesian optimization algorithm for large multi-objective monitoring network design problems. *Advances in Water Resources,* 31(5)**,** 828-845.

LARRAÑAGA, P. & LOZANO, J. A. 2002. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers.

LEAKE, D. B. 1996. *Case Based Reasoning: Experiences, Lessons, and Future Directions*, AAI Press/MIT Press.

LEWIS, R. 2008. A Survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum,* 30(1)**,** 167-190.

LIPINSKI, P. 2007. ECGA vs. BOA in discovering stock market trading experts. *Genetic and Evolutionary Computation Conference (GECCO-2007).*

LODI, A., MARTELLO, S. & MONACI, M. 2002. Two-dimensional packing problems: A Survey. *European Journal of Operational Research,* 141**,** 241-252.

MEHTA, N. K. 1981. The application of a graph coloring method to an examination scheduling problem. *Interfaces,* 11**,** 57-64.

MEHTA, N. K. 1982. A computer-based examination management system. *Journal of Educational Technology Systems,* 11**,** 185-198.

MERLOT, L. T. G., BOLAND, N., HUGHES, B. D. & STUCKEY, P. J. 2003. A hybrid algorithm for the examination timetabling problem. *In:* BURKE, E. K. & DE CAUSMAECKER, P. (eds.) *Practice and Theory of Automated Timetabling: Selected Papers from the 4th International Conference. Lecture Notes in Computer Science.* Springer.

MOSCATO, P. 1989. On evolution, search, optimisation, genetic algorithms and martial arts: towards memetic algorithms. *Caltech Concurrent Computation Program.* California Institute of Technology.

MOSCATO, P. 1999. Memetic algorithms: A short introduction. *In:* CORNE, D., DORIGO, M. & GLOVER, F. (eds.) *New Ideas in Optimisation.* McGraw Hill.

MÜHLENBEIN, H. & MAHNIG, T. 1999. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology,* 9.

MÜHLENBEIN, H. & PAAß, G. 1996. From recombination of genes to the estimation of distributions I. binary parameters. *In:* EIBEN, A. E., BÄCK, T., SHOENAUER, M. & SCHWEFEL, H. P., eds. Parallel Problem Solving from Nature (PPSN IV), 178-187.

MUMFORD-VALENZUELA, C., VICK, J. & WANG, P. Y. 2003. *Heuristics for large strip packing problems with guillotine patterns: An empirical study,* Kluwer Academic Publishers.

NORENKOV, I. & GOODMAN, E. 1997. Solving scheduling problems via evolutionary methods for rule sequence optimization. *2nd World Conference of soft Computing, WSC2.*

OCHOA, G., QU, R. & BURKE, E. K. 2009. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2009), Motreal, Canada, 341-348.

OSMAN, I. H. & KELLY, J. P. 1996. *Meta-heuristics: Theory and applications*, Kluwer Academic Publishers.

OSMAN, I. H. & LAPORTE, G. 1996. Metaheuristics: A bibliography. *Annals of Operational Reseach,* 63**,** 513-623.

ÖZCAN, E., BILGIN, B. & KORKMAZ, E. E. 2008. A Comprehensive Analysis of Hyper-heuristics. *Intelligent Data Analysis,* 12(1)**,** 3-23.

ÖZCAN, E., BYKOV, Y., BIRBEN, M. & BURKE, E. K. 2009. Examination timetabling using late acceptance hyper-heuristics. Proceedings of IEEE Congress on Evolutionary Computation (CEC 2009), 997-1004.

ÖZCAN, E., MISIR, M., OCHOA, G. & BURKE, E. K. 2010. A Reinforcement Learning – Great-Deluge Hyper-heuristic for Examination Timetabling. *International Journal of Applied Metaheuristic Computing,* 1(1)**,** 39-59.

PAPADIMITRIOU, C. H. & STEIGLITZ, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall.

PAQUETE, L. & STÜTZLE, T. 2002. Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. *In:* BURKE, E. K. & CAUSMAECKER, D., eds. Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling, KaHo St.-Lieven, Gent, Belgium, 413-420.

PELIKAN, M. 2005. *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*, Springer-Verlag.

PELIKAN, M., GOLDBERG, D. E. & CANTÚ-PAZ, E. 2000. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation,* 8(3)**,** 311-341.

PELIKAN, M., GOLDBERG, D. E. & LOBO, F. 2002. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications,* 21(1)**,** 5-20.

PELIKAN, M. & MÜHLENBEIN, H. 1999. The bivariate marginal distribution algorithm. *Advances in Soft Computing—Engineering Design and Manufacturing***,** 521-535.

PETROVIC, S. & BURKE, E. K. 2004. University timetabling. *In:* LEUNG, J. (ed.) *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press.

PILLAY, N. 2008. An analysis of representations for hyper-heuristics for the uncapacitated examination timetabling problem in a genetic programming system. Proceedings of the 2008 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, SAICSIT Conf. 2008, Wilderness, South Africa, 188-192.

PILLAY, N. 2009. Evolving Hyper-heuristics for the Uncapacitated Examination Timetabling Problem. Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 447-457.

PILLAY, N. & BANZHAF, W. 2007. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. *Progress in Artificial Intelligence, 13th Portuguese Conference on Artificial Intelligence, EPIA 2007 Proceedings, Springer, Lecture Notes in Computer Science,* 4874**,** 223-234.

PILLAY, N. & BANZHAF, W. 2009. A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem. *European Journal of Operational Research***,** 482-491.

PISINGER, D. 2002. Heuristics for the Container Loading Problem. *European Journal of Operational Research,* 141**,** 143-153.

QU, R. & BURKE, E. K. 2005. Hybrid Neighbourhood HyperHeuristics for Exam Timetabling Problems. Proceedings of The Sixth Metaheuristics International Conference (MIC 2005), Vienna, Austria.

QU, R. & BURKE, E. K. 2009. Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *Journal of Operational Research Society (JORS),* 60**,** 1273-1285.

QU, R., BURKE, E. K. & MCCOLLUM, B. 2009a. Adaptive Automated Construction of Hybrid Heuristics for Exam Timetabling and Graph Colouring Problems. *European Journal of Operational Research,* 198(2)**,** 392-404.

QU, R., BURKE, E. K., MCCOLLUM, B., MERLOT, L. T. G. & LEE, S. Y. 2009b. A Survey of Search Methodologies and Automated System Development for Examination Timetabling. *Journal of Scheduling,* 12(1)**,** 55-89.

REEVES, C. R. 1995. *Modern heuristics techniques for combinatorial problems,* McGraw Hill.

RESENDE, M. G. C. & RIBEIRO, C. C. 2003. Greedy randomised adaptive search procedures. *In:* GLOVER, F. & KOCHENBERHER, K. (eds.) *Handbook of Metaheuristics.* Kluwer Academic Publishers.

RESENDE, M. G. C. & RIBEIRO, C. C. 2005. GRASP with path-relinking: Recent advances and applications. *In:* IBARAKI, T., NONOBE, K. & YAGIURA, M. (eds.) *Metaheuristics: Progress as Real Problem Solvers.* Springer.

RIBEIRO, C. C. & HANSEN, P. 2001. *Essays and Surveys in Metaheuristics,* Kluwer Academic Publishers.

ROSS, P., HART, E. & CORNE, D. 1998. Some observations about GA-based exam timetabling. *In:* BURKE, E. K. & CARTER, M. W. (eds.) *Practice and Theory of Automated Timetabling II. Lecture Notes in Computer Science.* Springer.

ROSS, P. & MARÍN-BLÁZQUEZ, J. G. 2005. Constructive hyper-heuristics in class timetabling. *IEEE Congress on Evolutionary Computation, IEEE***,** 1493-1500.

ROSS, P., MARÍN-BLÁZQUEZ, J. G. & HART, E. 2004. Hyper-heuristics applied to class and exam timetabling problems: A new ga-based approach to hyper-heuristics. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'02, Morgan-Kauffman.

SANTANA, R., LARRAÑAGA, P. & LOZANO, J. A. 2010. Learning factorizations in estimation of distribution algorithms using affinity propagation. *Evolutionary Computation,* 18(4)**,** 515-546.

SASTRY, K., GOLDBERG, D. E. & KENDALL, G. 2006. Genetic algorithms. *In:* BURKE, E. K. & KENDALL, G. (eds.) *Introductory Tutorials in Optimisation, Decision Support and Search Methodology.* Springer.

SCHAERF, A. 1999. A survey of automated timetabling. *Artificial Intelligence Review,* 13(2)**,** 87-127.

SCHAERF, A. & DI GASPERO, L. 2001. Local search techniques for educational timetabling problems. Proceeding of the 6th International Symposium on Operational Research in Slovenia, 13-23.

SHAH, R. & REED, P. 2010. Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. *European Journal of Operations Research***,** In revision.

SHAKYA, S. & SANTANA, R. 2008. An EDA based on local markov property and gibbs sampling. Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO'08, New York, USA, 475-476.

STORER, R. H., WU, S. D. & VACCARI, R. 1992. New search spaces for sequencing problems with application to job shop scheduling. *Management Science,* 38(10)**,** 1495-1509.

STORER, R. H., WU, S. D. & VACCARI, R. 1995. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal of Computing,* 7(4)**,** 453-467.

TAILLARD, E. D., GAMBARDELLA, L. M., GENDREAU, M. & POTVIN, J. Y. 2001. Adaptive memory programming: A unified view of meta-heuristics. *European Journal of Operational Research,* 135(1)**,** 1-16.

TERASHIMA-MARÍN, H., ORTIZ-BAYLISS, J. C., ROSS, P. & VALENZUELA-RENDÓN, M. 2008. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. *In:* RYAN, C. & KEIJZER, M. (eds.) *Genetic and Evolutionary Computation Conference (GECCO 2008).* 571-578.

TERASHIMA-MARÍN, H., ROSS, P. & VALENZUELA-RENDÓN, M. 1999. Evolution of constraint satisfaction strategies in examination timetabling. *Genetic and Evolutionary Computation Conference, GECCO'99.*

TERASHIMA-MARÍN, H., ZARATE, C. J. F., ROSS, P. & VALENZUELA-RENDÓN, M. 2006. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. Proceedings of the 8th annual conference of Genetic and Evolutionary Computation (GECCO 2006), Newyork, USA, ACM Press, 591-598.

TERASHIMA-MARÍN, H., ZARATE, C. J. F., ROSS, P. & VALENZUELA-RENDÓN, M. 2007. Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. *In:* LISPON, H. (ed.) *Genetic and Evolutionary Computation Conference (GECCO 2007).*

THOMPSON, J. & DOWSLAND, K. 1996. Variants of simulated annealing for the examination timetabling problem. *Annals of Operational Reseach,* 63**,** 105-128.

THOMPSON, J. & DOWSLAND, K. 1998. A robust simulated annealing based examination timetabling system. *Computers and Operations Research,* 25**,** 637-648.

ÜLKER, Ö., ÖZCAN, E. & KORKMAZ, E. E. 2007. Linear linkage encoding in grouping problems: applications on graph coloring and timetabling.

Proceedings of the 6th international conference on Practice and theory of automated timetabling VI, Springer-Verlag Berlin, 347-363.

VOß, S. 2001. Meta-heuristics: The State of the Art. *Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers.* Springer-Verlag.

VOß, S., OSMAN, I. H. & ROUCAIROL, C. 1999. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers.

WÄSCHER, G., HAUSSNER, H. & SCHUMANN, H. 2006. An Improved Typology of Cutting and Packing Problems. *European Journal of Operational Research,* 183(3)**,** 1109-1130.

WELSH, D. J. A. & POWELL, M. B. 1967. The upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal,* 11**,** 41-47.

WHITE, G. M. & XIE, B. S. 2001. Examination timetables and tabu search with longer-term memory. *In:* BURKE, E. K. & ERBEN, W. (eds.) *Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference. Lecture Notes in Computer Science.* Springer.

WHITE, G. M. & XIE, B. S. 2004. Using tabu search with longer term memory and relaxation to create examination timetables. *European Journal of Operational Research,* 153(16)**,** 80-91.

YANG, Y. & PETROVIC, S. 2005. A Novel similarity measure for heuristic selection in examination timetabling. *In:* BURKE, E. K. & TRICK, M., eds. Practice and Theory of Automated Timetabling: Selected Papers from the 5th International Conference. Lecture Notes in Computer Science, 377-396.

ZHANG, D., KANG, Y. & DENG, A. 2006. A new heuristic recursive algorithm for the strip rectangular packing problem. *Computers and Operations Research,* 32**,** 2209-2217.