Selection Hyper-heuristics for Healthcare Scheduling

Monica Banerjea-Brodeur, BAA, MSc.

Thesis submitted to the University of Nottingham

for the degree of Doctor of Philosophy

June 2013

Abstract

A variety of approaches have been used to solve a variety of combinatorial optimisation problems. Many of those approaches are tailored to the particular problem being addressed. Recently, there has been a growing number of studies towards providing more general search methodologies than currently exist which are applicable to different problem domains without requiring any algorithmic modification. *Hyper-heuristics* represent a class of such general methodologies which are capable of automating the design of search process via generating new heuristics and/or mixing existing heuristics to solve hard computational problems. This study focuses on the design of *selection hyper-heuristics* which attempt to improve an initially created solution iteratively through heuristic selection and move acceptance processes and their application to the real-world healthcare scheduling problems, particularly, nurse rostering and surgery admission planning. One of the top previously proposed general hyper-heuristic methodology was an adaptive hyper-heuristic consisting of many parameters, although their values were either fixed or set during the search process, with a complicated design. This approach ranked the first at an international cross-domain heuristic search challenge among twenty other competitors for solving instances from six different problem domains, including maximum satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, travelling salesman, vehicle routing problems. The hyper-heuristics submitted to the competition along with the problem domain implementations can now be considered as the benchmark for hyper-heuristics. This thesis describes two new easy-to-implement selection hyper-heuristics and their variants based on iterated and greedy search strategies. A crucial feature of the proposed hyper-heuristics is that they necessitate setting of less number of parameters when compared to many of the existing approaches. This entails an easier and more efficient implementation, since less time and effort is required for parameter tuning. The empirical results show that our most efficient and effective hyper-heuristic which contains only a single parameter outperforms the top ranking algorithm from the challenge when evaluated across all six problem domains. Moreover, experiments using additional nurse rostering problems which are different than the ones used in the challenge and surgery scheduling problems show that the results found by the proposed hyper-heuristics are very competitive, yielding with the best known solutions in some cases.

Acknowledgements

I would like to thank my supervisors, Professor Edmund K. Burke and Dr. Ender Ozcan for their support and help during my PhD studies. I wish to thank my examiners Professor Patrick De Causmaecker and Professor Robert John for their valuable comments. I also wish to thank the members of the ASAP group and the School of Computer Science for their help and the UK Engineering and Physical Sciences Research Council (EPSRC) who funded this research under grant EP/D061511/1.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A variety of methods have been developed to solve a variety of real world combinatorial problems. The state-of-the-art methods for search and optimisation are frequently designed specific to the problem dealt with or even problem instances being addressed. In most of the cases, those algorithms are not applicable to the new problem domains, even if there might be many similarities. Additionally, a "minor" modification in the problem definition, such as a change in a constraint or inclusion of a new constraint could require an expert intervention, for example, to modify the problem-specific algorithm. It is both time consuming and costly to develop and maintain a solution method for a given problem or different problems. There is a growing interest in the development of low-cost general intelligent solution methodologies which are capable of automatically designing the search process, and so applicable not only to different unseen problem instances, but also to different problem domains. This study focuses on: (i) development of a general method to solve a range of combinatorial problems, and (ii) their application in healthcare. To this end, extremely effective methods are developed and their performances are investigated on six different problem domains using a benchmark. Moreover, the proposed approaches are applied to two healthcare real world problems: nurse rostering and surgery admission planning problems. A new software framework is designed for the surgery admission planning problems, while the same software framework as in the benchmark is used for nurse rostering. A comparison with other state-of-the-art and more general methods is done and the results show that the approaches developed outperform the current best known solution for 12 real-world nurse rostering and all available instances of the surgery admission planning problems as well as 8 instances of 5 different problem domains included in the benchmark.

1.1 Hyper-heuristics

Recently, there has been a growing number of studies towards providing more general search methodologies than the existing approaches which are applicable to different problem domains without requiring any algorithmic modification. For example, *hyper-heuristics* are capable of automating the design of search process via generating new heuristics and/or mixing existing heuristics to solve hard computational problems. This study will focus on hyper-heuristic methodologies. A more detailed outline of the research goals is defined in the following section. Over the last decade, there has been an increase in the development of more general approaches to solve combinatorial optimisation problems. This has been done successfully through the use of hyper-heuristics for a diverse range of problem domains, such as personnel scheduling and timetabling problem (Burke et al. 2003a, Cowling et al. 2000). A *hyper-heuristic* is an algorithm that operates on a search space of heuristics Burke et al (2003a). Figure 1.1 provides a general illustration of a hyper-heuristic framework. Hyper-heuristics can be represented as the algorithmic strategy that drives the selection or the generation of heuristics. These heuristics are applied to the different problems being solved. The hyper-heuristic will guide its search by the information provided by the low level heuristics. There exists a clear barrier between the hyper-heuristic and the problem domain implementations. There are two main types of hyper-heuristics in the literature; hyper-heuristic methodologies that *select* or *generate* low level heuristics (Burke et al. 2009).

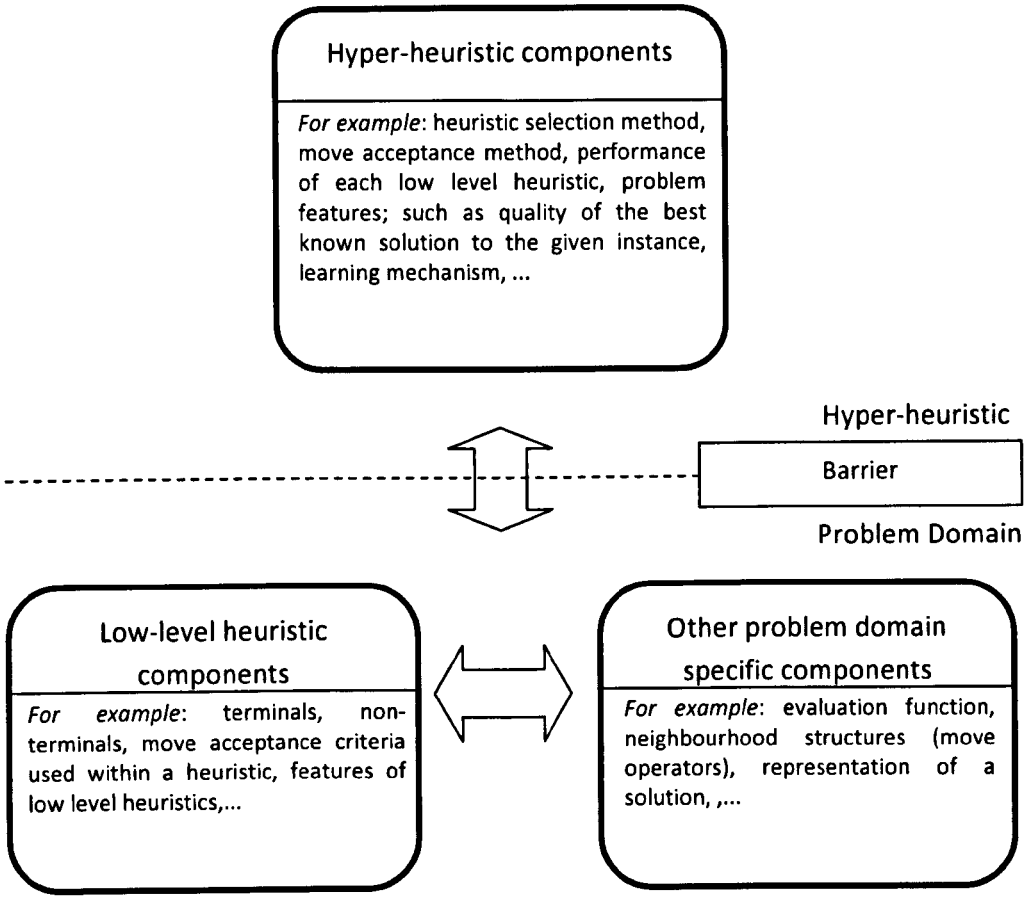


FIGURE 1.1: Layers and components of a hyper-heuristic framework

Hyper-heuristics have been classified in different ways. Earlier classifications such as Soubeiga (2003) include a distinction between a learning hyper-heuristics and a non-learning hyper-heuristic, where the learning or non-learning relates to the heuristic selection mechanism. The on-learning selection of heuristic uses a predetermined sequence of heuristics to apply. Further subcategories are defined within the learning category. Bai and Kendall (2005) and Ross and Marin-Blazquez (2005) distinguish between constructive hyper-heuristics and local search methodologies, where the low level heuristics applied are either constructive or local searches. Chakhlevitch and Cowling (2008) also establish a categorisation based on the selection of the heuristics. Burke et al. (2009e) classify hyper-heuristics in two main categories: selection hyper-heuristic and constructive hyper-heuristics, a

distinction is also made between an offline, online or no learning mechanism for the selection of the low level heuristics.

In this study, the focus is on *selection hyper-heuristics* which aim to improve an initial solution through heuristic selection and move acceptance processes, iteratively. Table 1.1 provides a description of a selection hyper-heuristic framework.

TABLE 1.1 A General Selection Hyper-heuristic Framework

Hyper-heuristic framework
1 generate initial candidate solution *s*
2 while (termination criteria not satisfied){
3 select a low level heuristic (or subset low level heuristics) *llh* from {LLH1, ...,LLH*n*}
4 generate a new solution (or solutions) *s'* by applying *llh* to *s*
5 accept/reject *s'* // decide based on acceptance method
6 if (*s'* is accepted) then *s* = *s'* }
7 return *s*;

Based on the classification of Burke et al. (2009b), four novel selection hyper-heuristics are presented produced as a result of this work. The proposed methods are capable of mixing and managing a given set of perturbative low level heuristics which process and return a complete solution at each step.

As mentioned previously, hyper-heuristics represent more general methodologies to solve combinatorial optimisation problems. The motivation for this research is to propose a more general methodology. The level of generality of a method can be evaluated against different criteria. For this study, the algorithms will be applied to unseen problem solving scenarios (instances) with different characteristic within the same problem domain. The second evaluation in regards to generality is to create a method that enables through its structure the solving of instances from different problem domains. Hence, the proposed hyper-heuristics are tested on different instances from seven problem domains: nurse rostering, flow shop, bin packing, travelling

salesman, Boolean satisfiability, vehicle routing and surgery scheduling. For each problem domain at least 10 different problem solving scenarios are considered and the performance of the proposed algorithms are analysed.

1.2 Personnel Scheduling

There have been many studies dedicated to the personnel scheduling problems over the years. Those problems can be categorised in the more general class of timetabling problems. Personnel scheduling problems arise at a variety of locations in the real world, such as crew scheduling on airplanes, scheduling staff for call centres, public transport, or even restaurants. Though all those problems are different, they are all considered as personnel scheduling problems. A general set of objectives and requirements can be attributed to the personnel scheduling problems. The objective is to assign personnel to different shifts or period of times in a day over a determined period of time. A general set of requirements can be categorised under three main headings; *coverage*, *work regulations* and *employee preferences* constraints. In this research, the focus is on the nurse rostering problems.

Providing better nurse rosters has been shown to improve the welfare of the nurses and the quality of the care provided to the patients. The western world is currently going through a phase of increase in healthcare costs, this trend will continue due to an ageing population. A shortage of nurses has been identified within many western countries. It is with this in mind that work done to improve nurse rosters and make the process efficient, i.e. automatically through algorithmic tools, is important. It has been argued that ensuring nurses are allowed fair schedules and accommodated in as many preferences as possible improves satisfaction and lowers fatigue. This will be reflected in the quality of care given to patients as well as an improvement in the nurse's life.

The nurse rostering problem consists of creating a roster for a ward in a hospital. A roster includes the schedule of each nurse within the scheduling period. The scheduling period is typically a month but it can also be another predetermined period. Each nurse's schedule must respect a set of restrictions and considerations such as contract obligations, preferred days off, preferred

work period as well as other considerations such as training days. The hospital will also have regulations and restrictions such as a nurse cannot work a night shift and the next day work an early morning shift, the roster must also ensure that the adequate number of nurses of each skill category is working when needed. The roster must also be fair as an example ensuring week-ends off are assigned equally between each nurse. It is easy to see that the nurse rostering problem is complex to solve and can never be realistically solved to optimality, hence the interest of the research done on this issue within the scientific community over the last 45 years.

As mentioned previously, there are three main sets of constraints that include all requirements for the nurse rostering problem. The objective is to assign a nurse to a shift or period of time, while respecting the three general sets of constraints. Due to the high number of constraints most models will separate these in two categories; soft constrains and hard constraints. The constraints included in these two categories will depend on the hospital's or the organisation's regulations. Soft constraints are requirements that can be violated. The respect of as many soft constraints as possible will contribute to determining the quality of the roster. Typically the more important soft constraints will be attributed a high cost or penalty when violated; this ensures a better quality of rosters. Hard constraints are defined as requirements that cannot be violated when building the roster. These two categories will not affect the three sets of constraints that will be defined. In fact, any constraint in any of the three sets could be considered a hard or soft constraint this will depend on the direction of the organisation. Briefly, the required number of nurses of each skill category needs to be scheduled at the required period of time, this encompasses essentially the coverage constraint. The number of required nurses is determined prior to establishing the roster and is not included in the nurse rostering problem definition. The work and contract regulations ensure that the number of hours worked by each nurse respects his/her contract as well as including regulations that apply to all staff. The third category will include all the nurse preferences. Table 1.2 will provide a few examples for each set of constraints, these are not exhaustive.

TABLE 1.2: Examples for each constraint type

Constraint Types	Description
Coverage	- Schedule the number of nurses required for each shift - Ensure required number of nurses in each skill: head nurse - Ensure proper skill pairing: student nurse-senior nurse
Work and Hospital	- Maximum number of hours of work - Minimum number of hours of work - Late shift must not be followed by early morning shift - Maximum of three consecutive night shifts - No solitary shifts - Two consecutive days off after specified number of hours worked
Preferences	- Specific day of the week off - Distribute week-ends off evenly between nurses - Preferences for night shift work

Many approaches and models have been used to solve and reflect the nurse rostering problem. Burke et al. (2004a) review the research done over the last 40 years. The paper raises some interesting points; specifically weaknesses and future direction within the research community in nurse rostering. Over the last decade, new approaches have been developed to use on the nurse rostering domain. As this research is concerned with providing more general methodologies to solve healthcare issues, these methods will be highlighted.

For this thesis, 43 real world problems will be used to evaluate a more general strategy that has been developed. The following chapter will also provide a more in-depth review of the problem domain and an algorithmic overview of methods used to solve nurse rostering problems.

Meta-heuristics and heuristics comprise the vast majority of the research done over the last decade on nurse rostering. A few examples of methods that were successfully applied to instances of the nurse rostering problem that are evaluated in this study include tabu search algorithms (Burke et al.,2009b), genetic evolution algorithms (Burke et al., 2004a), a hybrid method proposed by Valouxis and Housos (2000). Their algorithm starts by finding an initial solution using a integer linear program and improve the solution by using two heuristics. Case-based reasoning is also a strategy that has had success in

solving different problems within nurse rostering instances that are explored in this study (Beddoe et al., 2009, Beddoe and Petrovic ,2007). More general methods such as hyper-heuristics were also applied successfully to a subset of the nurse rostering problems evaluated in this thesis (Burke et al., 1999a).

1.3 Surgery Scheduling

Another research interest explored in the thesis is the planning of surgeries within a hospital, this entails scheduling surgery teams and assigning patients a day and time of surgery considering various requirements and allocating physical resources such as equipment and operating theatre. Surgeries are the highest cost drivers within hospitals. Therefore the importance of properly managed surgeries at the human and physical resources level is of high importance. Again patient quality of care is heavily involved in proper surgery planning as patients who require emergency surgeries or quick surgeries need to be treated within the prescribed time. Other considerations such as eliminating surgery cancellations are also important factors that contribute to the patient quality of care.

This research will explore the surgery admission planning problem. The problem can be defined as assigning known surgeries a day over a determined period of time. This is a medium to long term planning problem. Once the day on which the surgery will be done has been determined, the second task involves determining the time of day and assigning the operating theatre for the surgery. This is a short to medium term planning problem. The final schedule must ensure that surgeons are not booked for more than one surgery at a time and that all equipment and operating theatres are available. Surgeries also need to be assigned while respecting the delay prescribed by the specialist i.e. each surgery is given a maximum delay by which it needs to be done, this will reflect the urgency of the intervention.

The general set of problems that can be defined within the surgery scheduling domain has not been as extensively studied as the nurse rostering problem. It encompasses a wide range of problems and there is as yet no current consensus as to the exact definition of the problem. Some models consider the

availability of nursing and associate members of staff, bed recovery time required, outpatients and inpatients surgeries when scheduling surgeries. The planning horizon may also differ, from one day of surgeries in a hospital, a week, a month or even a year. The surgery scheduling domain includes many activities or tasks from the planning of the surgery i.e. the initial schedule, the variation in the surgery duration, the equipment required will determine the operating theatre and the day of operation. The last stage is the recovery time required for the patients following the operation, often with complex interventions the time required can be greater than the duration of the surgery. This recovery time may involve a few days spent in the hospital. For this research, as mentioned previously, the surgery admission planning problem will be explored. Many different sets of requirements and objectives can be included in the surgery admission planning problem. For the purpose of this research, three main objectives will be defined. Following Riise and Burke (2011), the surgery admission planning model will be defined using three main objectives. These include the maximum permitted delay between the referral date of the patient to the scheduled surgery date, the maximum overtime of surgeons tolerated and assigning children an early time of surgery on the scheduled day. Table 1.3 provides a general description of the surgery admission planning problem.

TABLE 1.3: Description of surgery admission planning problems

Constraint Types/Objectives	Description
Patient Waiting Time	- Maximum delay from referral date - Preferred window of time for operation - Previous cancellations
Surgeon	- Surgeon overtime - Surgeon availability - Surgeon can only perform one operation at a time - Surgical team availability
Time of Day	- Schedule children early - Schedule elderly/disabled patients later - Schedule patients with health problems early

Several studies have been done within the field of surgery scheduling. As mentioned previously, this area of healthcare planning has not been as extensively studied as the nurse rostering problem. There are however, important contributions within this field. As there is currently no general consensus within the research community on an exact definition of the problem, each problem proposed varies considerably and the objectives emphasised are different. Hans et al. (2008) propose to maximise the utilisation of the operating theatre whilst minimising overtime. Denton et al. (2007) focus on assigning a time of surgery for one day and one operating theatre, where the focus is to account for the uncertainty of the duration of the surgeries. Santibanez et al. (2007) create different models where different constraints are prioritised such as operating room equipment, operating theatre capacity, bed recovery availability, staff availability and surgery duration. These three publications provide a glimpse of the wide array of problem definitions and objectives included in the field of surgery scheduling.

The four hyper-heuristics developed for this research are also applied to ten real world different problem solving scenarios of the surgery admission planning problem.

1.4 Research Objectives

This research has two key objectives. Designing more general methods and applying these to real world problems. Specifically, two important healthcare scheduling problems will be investigated: the nurse rostering and the surgery admission planning problems. A variety of instances collected across the world are used for the experiments; 43 well-known benchmark nurse rostering problems and 10 benchmark surgery admission planning problems. The recent developments in nurse rostering allowed a variety of problems across the world to be represented based on the same format. The format is so general that problem instances with totally different characteristics can be represented under the same description. This is the motivation to come up with a general methodology that is successful over those instances as well as the importance of healthcare issues.

Designing an effective and general method applicable to different problems as well as problems with different characteristics is a challenging task, considering there is almost no theoretical or mathematical guidance to do so. Consequently, a software package, HyFlex is used to analyse the behaviour of different low level heuristics implemented for different domains in order to come up with a viable design. The secondary objective of this research is to assess the effectiveness of the designed approach across different problem domains in relation to different choices of hyper-heuristic components.

1.5 Contributions

A number of original research contributions follow from this research:

- An efficient and effective general selection hyper-heuristic is designed which outperforms the best known selection hyper-heuristic on the CHeSC[1] 2011 benchmark. Another hyper-heuristic based on a different framework combining different heuristic selection and move acceptance components ranks at the same level as the third CHeSC 2011 competitor. All the proposed hyper-heuristics are easy-to-implement approaches with a few parameters to set. The parameters are a probability when selecting low level heuristics, the number of iterations before an evaluation is done and an improvement threshold.

- The analyses of the CHeSC 2011 low level heuristics is performed for the first time in the literature. The analyses show that the ruin and recreate heuristics are basically mutational heuristics and so a different strategy for annotating heuristics is required in HyFlex.
- The hyper-heuristic methods are modified slightly and redesigned as anytime algorithms (which runs as long as the solution at hand improves) rather than time contract algorithms (which terminate after a given time limit) as implemented for the CHeSC 2011 benchmark, since there is more than ten minutes of nominal time (as imposed in the competition) for nurse rostering and surgery scheduling. One of the proposed hyper-heuristics produced the best results for the nurse rostering instances of, *BCV1.8.1,*

[1] CHeSC 2011: Cross-domain Heuristic Search Challenge.

BCV.1.8.2, BCV2.46.1, ERRVH-A and *QMC-A* , while the other one produced the best results for *BCV1.8.3, BCV1.8.4, BCV6.13.1, CHILD-A2, ERMGH-A, ERMGH-B, ERRVH-B.*

- A new software tool is implemented for the development of hyper-heuristics for solving the surgery admission planning problem, including domain specific low level heuristics. The same anytime hyper-heuristics used for nurse rostering is applied to a set of surgery planning problem instances. One of the selection hyper-heuristic beats the best known solutions on the instances of *WOT4, WOT5, WOT6* and *WOT8* while the other also produced the best results for *WOT1, WOT2, WOT3, WOT4, WOT5, WOT6, WOT7, WOT9* and *WOT10.*

1.6 Dissemination

As a result of this study, three journal papers are produced which are under review or in writing:

- Banerjea-Brodeur Monica, Edmund K. Burke and Ender Özcan. An Efficient and Effective Hyper-heuristic for Cross-domain Heuristic Search. INFORMS Journal of Computing.
- Banerjea-Brodeur Monica, Edmund K. Burke and Ender Özcan. Hyper-heuristics for Real World Nurse Rostering. Naval Research Logistics.
- Banerjea-Brodeur Monica, Edmund K. Burke and Ender Özcan. Hyper-heuristics for Real World Surgery Admission Planning Problems. Applied Soft Computing.

1.7 Thesis Structure

Chapter 2 provides an overview of the research that has been done in the field on nurse rostering and surgery scheduling. The papers are grouped by subject and methodology. This chapter also presents the benchmark instances for those healthcare problems.

Chapter 3 overviews intelligent and general search methodologies applicable to different problems, with an emphasis on hyper-heuristics which explore the space of heuristics in search and optimisation. The details of a software tool,

namely HyFlex which was used at a hyper-heuristic competition is presented, focusing on the framework and problem domain implementations. The problem instances used in each domain and the design of low level heuristics are all described.

Chapter 4 provides empirical analyses of the behaviour and performance of all low level heuristics using the public problem domains of the CHeSC 2011 benchmark to form a basis for the design of effective and efficient selection hyper-heuristics.

Chapter 5 provides the specifics of the proposed both anytime and time contract selection hyper-heuristic methodologies build on the analyses provided in the previous chapter.

In chapter 6, the experimental results of applying the proposed hyper-heuristics to forty three nurse rostering benchmark instances are discussed. The average and best-of-runs performances of the anytime hyper-heuristics are compared to each other as well as to the previously known approaches whenever available.

Chapter 7 focuses on the application of the time contract hyper-heuristics to the following problem domains; Boolean maximum satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, vehicle routing and travelling salesman problems. The experimental design and results are presented in detail. Firstly, performance of the proposed hyper-heuristics is compared to the performance of hyper-heuristics that joined the mock competition organised before CHeSC. Again, the average and best-of-runs performances of the time contract hyper-heuristics are compared to each other as well as to the CHeSC competitors.

Chapter 8 explains the domain specific low level heuristics created for the surgery admission planning problem under a selection hyper-heuristic framework. This chapter also reviews the results obtained by the anytime hyper-heuristics on the surgery admission planning domain. The experimental design and results obtained are discussed. The performances of the proposed hyper-heuristics are compared on this problem domain.

Chapter 9 summarises the results of the thesis and concludes the thesis. The future work is outlined in this chapter.

Chapter 2

Healthcare Scheduling

The algorithms developed in this study will focus on solving real world healthcare problems and showing that it is possible to increase the level of generality of the solution method. Taking this into consideration, the literature review will emphasise work done on real world healthcare problems in surgery scheduling and nurse rostering and on more general methods developed[2].

Personnel scheduling has been studied extensively through the years. Ernst et al's (2004) survey on personnel scheduling includes up to 700 publications and the study is not exhaustive. Of these problems, many publications are specifically on the nurse rostering problem. As mentioned previously, the field of surgery scheduling has not been studied as extensively as the nurse rostering problem. This is partly due to the fact that it is a large and complex problem that includes many sub-problems with conflicting objectives reflecting the interest of various stakeholders and there is currently no consensus within the research community on the definition of the problem. Surgery scheduling is of increasing interest as surgeries are the largest cost drivers of a hospital and operating rooms are the costliest resources in a hospital.

Nurse rostering publications are grouped by methodologies and models. Most surgery scheduling problems researched are different i.e. they address different phase and/or sub-problem of this domain. The publications will therefore be categorised by the objectives the model is emphasising. The chapter will be divided in the following sections. Section 2.1 will include a review of research done in nurse rostering. Section 2.2 will review the algorithms proposed for the international nurse rostering competition. Section 2.3 will focus on the research within the surgery scheduling domain. Section 2.4 and section 2.5 will

[2] A more extensive literature review on studies done on nurse rostering and surgery scheduling are included in Appendix B.

explain the benchmark instances used for the nurse rostering problem domain and the surgery admission problem domain, respectively.

2.1 Nurse Rostering

As discussed in Chapter 1, the real world nurse rostering problem is complex to solve. Good quality nurse rosters entail better quality of care for patients, nurse retention and better quality of life for nurses. Nurse rostering problems have held the interest of the scientific community for over 45 years.

Burke et al. (2004a) review the research done over the last 45 years on nurse rostering problems. The authors analyse the feasibility of applying models and solutions developed over the last 45 years in current hospital environments as well as highlight some interesting models, parameters and algorithms/solutions which could be further developed or be added in other models to provide solutions to real nurse rostering problems. The paper also concludes on the weaknesses of the research on nurse rostering to solve nurse rostering problems in the current context. The authors propose means to overcome these weaknesses in order to apply research to real life nurse rostering problems. By reviewing and analysing the different approaches used over the last 45 years in regards to nurse rostering the authors demonstrate the complexity of the nurse rostering problem.

The authors conclude that very few studies can be applied to the current nurse rostering problem and suggest the means that need to be explored to overcome the current weaknesses, these should be used as a benchmark to evaluate future research in the field. Specifically it is necessary to evaluate the practicality of implementing the models and algorithm in real world environments considering the representation of the problem, the accessibility and implementation of the software, solution or model i.e. comprehension, user-friendly, ease of implementation, flexibility of modelling.

The following publications have been grouped by the general methodology Section 2.1.1 encompasses research done using single point search methods. Section 2.1.2 reviews a few publications on multiple point search methods and

section 2.1.3 focuses on a few select hyper-heuristic and cased-based reasoning approaches.

2.1.1 Single Point Search Algorithms

Although published in 1976, Smith (1976) developed an interactive software that has been used in a large St Louis hospital to create nurse schedules while allowing the scheduler to manage the trade-off between various requirements. The software allowed flexibility and enabled the decision maker to ensure more important constraints were satisfied. The algorithm developed reflects many realities encountered in nurse scheduling such as determining the number of full time nurses to employ and vacations days. The software was enhanced in Smith and Wiggins (1977) by storing individual preferences and work requirements. The model considers skill categories of nurses, part-time workers and floating nurses.

Burke et al. (1999a, 2001a, 2004b, 2004c, 2006a, 2009b) describe the problem of nurse rostering in Belgian hospitals, the software developed to replace the manual creation of the schedules and enhancements provided in further studies. All the problems modelled in these publications are used in this thesis. Although Burke et al. (2001a, 2001b, 2004b) employ multiple point search algorithms, these publications are included in this section because they deal with the same nurse rostering problems. The Belgian hospital nurse rostering problem is unique; the nurses' preferred schedules are flexible, they are not cyclical and the work period is not divided in shifts. The level of flexibility, work requirements and nurse preferences make the problem difficult to solve optimally in adequate computational time. The software and subsequent algorithms tested and/or included in the initial software include tabu search algorithm with diversification heuristics, a steepest descent search algorithm, greedy search algorithm with varying depth of neighbourhood searches, tabu search with hybridisations techniques based on the respect of certain types of constraints, memetic, scatter search and evolutionary algorithms. All these algorithms have been successful in providing good solutions often the best known solution for the problems in Belgian hospitals. The scatter search algorithm obtaining the best known solution and optimal results for many of

these instances. The software developed in these studies was implemented in Belgian hospitals due to the flexibility of the model and approach; enabling the scheduler to prioritise constraints and determine staffing requirement levels.

The problem proposed by Valouxis and Housos (2000) is included in the 43 real world benchmark problems used in the thesis. The authors solve the nurse rostering problem by a combination of integer linear programming (ILP) and heuristics. The authors model the problem as a partial ILP while considering constraints relating to requirements for nurses on each shift, a minimum rest period between shifts, a minimum number of Sundays of rest. An initial solution is found by solving the ILP and this solution is improved using two heuristics. The aim is to improve for each nurse the shift patterns of the roster. This is done by firstly changing shifts between two nurses if the solution value is improved and creating partial rosters by removing shifts at the end of each day and reconstructing the roster. The methods were tested on data from a real hospital for wards of 10 to 30 nurses. As the authors suggest greater flexibility to add constraints and objectives would improve the benefits of the program.

Ikegami and Niwa (2003) model the nurse rostering problem in Japan. These instances are used in the thesis. In Japan nurses have rapid shift rotations i.e. nurses work the same shift for a short period of time. The authors define scenarios which include 2 and 3 shifts per day. The instances are sub-divided and one nurse schedule is solved at a time, using a tabu search for the 2 shifts and a branch and bound algorithm for the 3 shifts per days scenarios. The computational time required to solve these instances needs to be decreased, as pointed out by the authors.

Burke et al. (2008a) model and solve the problems obtained by industrial collaborator ORTEC. The methodology finds an initial feasible solution that satisfies the coverage and work constraints. The initial solution is found by evaluating each shift and determining the difficulty of attributing the shift to a nurse. The more difficult shifts are assigned first in the roster. An improvement phase is removing costly schedules from one nurse and attributing it to another. Only improving moves are accepted. The results

obtained are similar to those found by HARMONY, a commercial software that uses a genetic algorithm to solve the problem. The approach developed by Burke et al. (2008a) allows quickly to view if a feasible solution is possible. Burke and al. (2009f) apply a combination of integer programming and a variable neighbourhood search algorithms to solve the same problem. An initial feasible solution is found for the relaxed version of the problem using integer programming. A variable neighbourhood search is applied to satisfy the remaining constraints.

Burke et al. (2009g) represent a nurse scheduling problem in a Dutch hospital as a multi-objective model, where each objective is a soft constraint. The authors introduce the idea of building shift patterns which are allocated to each nurse by a squeaky wheel optimisation. The criteria of allocation is based on the respect of coverage requirements. The algorithm adds an element of randomness by evaluating the shift pattern's fitness against a random number and eliminates a small number of shifts to create partial schedules randomly. The nurses that do not have a shift pattern anymore are ordered based on fitness and are scheduled to ensure coverage requirements and feasibility.

Li et al. (2009) focus on an two phase local search algorithm similar to Lourenco et al. (2003) to solve a nurse rostering problem in a UK hospital. This problem was previously studied by (Aickelin and Dowsland, 2000 and Aickelin and White, 2004). The first step of this approach is to find a solution i.e. a schedule pattern for each nurse. The schedules are evaluated according to the preferences of the nurses and the restrictions of the hospital in terms of coverage i.e. a certain number of nurses of specific qualifications are required. Each nurse's schedule is evaluated and the ones that do not satisfy these restrictions are eliminated, some nurses do not have an assigned schedule anymore. Then a second elimination step is done randomly where a small percentage of schedules are eliminated, similarly to Burke et al. (2009g). The last phase of the heuristic is to assign the nurses to a schedule while taking into account the coverage, qualifications requirements and the nurse's preferences.

Brucker et al. (2010) present and use real world nurse rostering data, which is used in the thesis. The problems are decomposed into subsets which will be

solved separately. Sequences of shifts are created for each nurse, the best sequences will then be attributed to each nurse and each schedule combine to create the final roster. If roster constraints are not met, the roster is modified by swapping nurse schedules. A further evaluation is done and more changes are made to the roster to ensure respect of constraints or obtain better solutions.

2.1.2 Multiple Point Search Algorithms

Multiple point search algorithms, such as genetic algorithms have been explored to solve the nurse rostering problem.

Aickelin and Dowsland (2000) present results of a variant on a genetic algorithm that was used to solve a real nurse rostering problem in a UK hospital. The authors find an initial solution provided by a genetic algorithm. The next step is to make swaps between schedules to improve the roster. In Aickelin and White (2004) a genetic algorithm is used to determine the nurses that will work, an added element which the authors call a decoder assigns shifts to the nurses. The authors' main objective is to present a method to compare algorithms or improvements to algorithms.

Aickelin et al. (2009) combined a memetic algorithm and an ant-miner algorithm to solve the same nurse rostering problem. The authors transform the mathematical model previously created by Aickelin and Dowsland (2000) to reflect the nurse rostering problem into an acyclical graph where nodes represent the nurse and the rule used to schedule the nurse and the edges will be used to evaluate schedule patterns.

2.1.3 Hyper-heuristics and Case-based reasoning: real world problems

In Burke et al. (2003a) apply a hyper-heuristic to one nurse rostering and one course timetabling problem. The authors propose a set of simple heuristics and define a criterion which will be used to evaluate improvements of the solution. Heuristic selection is based on its performance. The heuristics that do not improve the solution are placed in a tabu list. Only improving solutions are accepted.

Bai et al (2010) present a simulated annealing hyper-heuristic approach to solve a real world nurse rostering problem. This method starts with an initial often unfeasible solution of shift patterns for each nurse. The shift patterns are evaluated by attributing a ranking based on a probability of the pattern being chosen above another pattern (stochastic ranking). A hyper-heuristic is introduced that will select between a set of low level heuristics to find feasible shift patterns. The moves made by the heuristic are evaluated against a criterion (temperature) that is updated at each iteration. The evaluation is used by the selection process to monitor the performance of each heuristic.

Smet et al. (2012) present a general model, 6 new real world benchmark problems and hyper-heuristic experiments. The model includes all requirements and objectives relating to the nurse rostering problem. The authors evaluate different combination of hyper-heuristics components such as heuristic selection and move acceptance. The authors found that the move acceptance component is more important to solution quality than the heuristic selection for the 6 data sets tested.

Cased-based reasoning methods have been applied to the QMC nurse rostering problem that is used in this thesis. Petrovic et al. (2003), Beddoe and Petrovic (2007), Beddoe et al. (2009) use the same methodology which consists of creating a case-base where previous constraints violations and the repair used by a scheduler are stored in the case. When identifying a constraint violation in the roster creation process the cases in the case-base that resemble the current violation are retrieved and evaluated according to their similarity to the current violation. The repair done to the current violation will reflect the repair stored in the case. The repair is done through swaps in the schedule and the use of tabu lists.

2.2 Nurse Rostering Competition

The first international nurse rostering competition 2010, Haspeslagh et al. (2010), provided various algorithms to solve benchmark instances of the nurse rostering problem. The objective of the competition was to encourage the

creation of new solution methods for the problem from researchers of diverse backgrounds and to provide problems that contain more real world information than many of the nurse rostering problems researched. Training or early instances of the problem were provided to the competitors at the opening of the competition and other instances were provided 2 weeks before the end date. Other instances were hidden and the competing algorithms were applied to these after the close of the competition. Three types of tracks or times were selected i.e. sprint where the time limit is of 10 seconds, middle distance the maximum time limit is 10 minutes and long distance where the time allocated is 10 hours. These reflect different aspects of the problem. The algorithms that were submitted to the competition and that made the final list will be reviewed in this section.

Nonobe (2010) proposes to model the problem as a constraint optimisation problem where a weight is assigned to each constraint and the goal is to minimise the total cost. The author assigns a possible set of shifts to each nurse and a new binary variable for which a value of 1 is given if a nurse is working on a specific day a specific shift (from its possible set of shifts). A tabu search type algorithm is used to solve the problems, the candidate solution is modified by interchanging the value of a variable for another. The same model and algorithm was applied to all three tracks. The algorithm came in third place for the sprint, second place for the middle distance and third place for the long distance.

Zhipeng Lu and Jin-Kao Hao (2010) provide a local search based algorithm with multiple restarts. Two moves are considered, the first is to assign a shift in a day to a nurse and the second is to swap two shifts between two nurses on a specific day. The search strategies include an in-depth search, a diversifying mechanism and an intermediate search. For the in-depth or intensive search all moves are explored but a tabu list is kept to ensure the recently visited solutions are not re-evaluated. The diversification mechanism ensures that the algorithm does not get stuck in local optimum. All moves concerning possible swaps between two nurses are evaluated against a subdivision of the total cost function. Only a move that can improve a sub-cost is accepted. The intermediate search consists of using the same diversification mechanism and

allowing the best move that has not been recently made. Further diversification is ensured by alternating the search process between two neighbourhoods. A fixed list of good solutions is kept and if the current search process does not provide good results a re-start of the search procedure is done using a good initial solution.

Burke and Curtois (2010) re-model the nurse rostering competition instances to their previously defined rostering model problems where the problem requirements are defined within acceptable patterns, workload, skill requirements and pairing as well as other conditional requirements. The authors applied a variable depth search algorithm to the instances in the sprint category. The algorithm constructs an initial solution that is improved by improving the nurses schedule individually, if at the end of the consecutive moves the roster is worst a return to the pre-move roster is made. This is proven to be very successful. A branch and price algorithm is used for the middle distance and long distance categories, where the same variable depth search algorithm is applied to the pricing problem.

Valouxis et al. (2012) propose a decomposition of the problem instances, these sub-problems are solved using integer programming. The first step is to create a workload schedule for each nurse, this implies creating a schedule where the nurse is assigned a work day or a rest day. The first step is divided in sub-problems to represent 7 consecutive work days. Each sub-problem is solved using integer programming. The accumulation of these sub-problems will cover the whole scheduling period. Once a solution is found for this first step a heuristic with three types of moves is applied to improve the solution. The moves include separating the complete schedule at one or two points and then recreating the complete schedule with these pieces. The last operator consists of evaluating potential swaps between each individual schedule to all the other schedules, only improving moves are accepted. An integer programming model is also used to solve the second step of the problem i.e. to assign a shift to the work days for each nurse, while ensuring required coverage for each shift and no nurse works more than one shift per day. This method was the winner for all three tracks in the competition.

Bilgin et al. (2010) present a hybrid approach to solve the nurse rostering problems defined in the competition. The algorithm uses as a first step a hyper-heuristic approach followed by a greedy shuffle heuristic. The hyper-heuristic is used for 80% of the available computational time; the remainder is given to the greedy shuffle heuristic. Starting from an initial feasible solution the hyper-heuristic selects different low-level heuristics. These evaluate swaps between two nurses. The candidate solution is accepted based on a simulation annealing principle, where a solution is accepted if it is better than the current solution or due to a random number calculation. The best solution found by the hyper-heuristic will after 80% of the total time limit be given as a starting solution to a greedy shuffle heuristic. The greedy shuffle heuristic evaluates swaps between partial schedules and accepts only non-worsening solutions. After a determined number of iterations without any improvements to the candidate solution a perturbation is made. This method obtained 6 best results in the sprint track and five best results for the long-distance track.

Rizzato et al. (2010) define the nurse rostering problem as a multilevel assignment problem, where the roster is divided in two parts and the assignment of a nurse to a day is done in a first constructive phase. The second phase consists of improving the roster through different swaps i.e. possible recombination of shifts between two days, swapping shifts between nurses on one day and possible perturbation to the solution to find new solutions. The heuristic algorithm based on multi-assignment problems found good all-around results for the long distance track of the competition.

The following publications use the benchmark instances provided by the first international nurse rostering competition. Messelis and De Causmaecker (2011) use the instances in the middle track distance category and two algorithms submitted to the competition to build an algorithm portfolio where the goal is to predict the results that would be obtained by an algorithm on an instance of a problem without having run the algorithm on this instance beforehand. The prediction model is based on the learning achieved by using the two algorithms on training instances and identifying the important features within the problem instances that determine the efficiency of the algorithm. This means eliminating the features that do not affect solution quality. The

authors show that a good algorithm portfolio is better than the use of only one algorithm.

Bilgin et al. (2012) evaluate different selection hyper-heuristic on instances of patient scheduling problem and instances of the nurse rostering problem. The instances of the nurse rostering problem are the benchmark instances provided for the first international rostering competition within the sprint and middle distance categories. The authors evaluate a selection mechanism and the move acceptance criteria for both problems. The selection mechanism includes a simple random choice, a choice function which will evaluate the performance of each low-level heuristic individually, their combination and a dynamic strategy that evaluates the best low level heuristics subsets during different phases of the algorithm. The move acceptance criteria include only accepting improving solution, accepting non-worsening solutions, accepting worsening solutions with a probability (simulated annealing) and moves that are better than a threshold (great deluge). The authors found that the best move acceptance criterion is the great deluge for the patient scheduling problem, there was no statistically significant better selection mechanism for this problem. No selection mechanism or move acceptance is better for the nurse rostering problem. However, the authors note that the acceptance of only improving candidate solutions provides the poorest results for both problems.

The nurse rostering competition has provided benchmark instances of the nurse rostering problem as well as spurring different approaches to solve these instances.

2.3 Surgery Scheduling

As discussed in Chapter 1, the problems included in surgery scheduling are complex and vary in different hospitals, clinic, wards and countries. As previously mentioned, this area of healthcare has not been studied as extensively as the nurse rostering problem, it is however an area of increasing interest. This section is an overview of the research published within the vast field of surgery scheduling. Most papers address a different issue within the field and each method is different, however the models proposed often

emphasise one or more goals. The surgery scheduling problem contains like most combinatorial problems many conflicting requirements. A summary of the most widely defined objectives/requirements is provided in Table 2.1.

TABLE 2.1 Examples of Objectives and/or Requirements in Surgery Scheduling

Objectives and/or Requirement	Description
Personnel and Other Resources Requirements	- Availability of surgeons - Availability of nurses - Availability of OR (operating rooms) - Availability of specialist equipment - Availability of other members of staff - Availability of post-recovery facilities
Personnel Objectives	- Minimise surgeon overtime - Distribute surgeon workload evenly - Respect hospital and/or ward work regulations - Minimise post anaesthesia nurses
Resource Objectives	- Maximise surgeries to OR - Minimise wasted time in OR - Minimise overtime usage of OR - Maximise post-surgery facilities - Minimise bed shortages - Minimise duration of surgeries
Patient Objectives	- Minimise patient waiting time to schedule surgery - Schedule children early on day of surgery - Maximise number of patients having surgeries

As can be seen from Table 2.1 many requirements or objectives could have been placed in a different group or are very closely related. For example, minimising overtime usage of the operating room influences the overtime of surgeons. Another example could be that minimising bed shortages is related to maximising post-surgery facilities.

Cardoen et al. (2010) review publications within the field of operating room planning and scheduling. As the problem vary in the problem that is addresses and the approach, a few characteristics are included to divide the more recent research done in this field. Six characteristics are proposed, these relate to the type of patient (elective and non-elective), the type of decision, performance measures used to identify the objectives/goals of the study, the algorithmic

tools used to solve the problems, the uncertainty consideration, ease of implementation of the method or model. The authors note that very few studies integrate a realistic flow of incoming patients which include elective and non-elective patients and other realistically stochastic problems such as surgery durations. An integration of all resources used within scheduling of operation rooms should also be privileged.

The research outlined will be grouped by the objectives or main requirements emphasised in the models proposed. If there is more than one objective defined in the model, the publication will be included in only one category.

2.3.1 Patient Objectives

Riise and Burke (2011) study a real world surgery admission planning problem. The authors define the problem as scheduling a day for each elective surgery over a planning period and assigning a time of day and operating theatre. A schedule for each surgeon also needs to be maintained as the surgeon can only perform one operation at a time. Three objectives are addressed: minimise waiting time (between referral date and date of scheduled surgery) and surgeon overtime and assign children at the earliest possible moment in the day. The objective function consists of the aggregation of these three objectives. The authors created test data that reflects the current scenario in Norwegian hospitals. The algorithm used by Riise and Burke to solve this issue is a combination of a steepest variable neighbourhood descent and an iterated local search where an operator is selected and used at each iteration of the algorithm until no improvements can be made to the candidate solution using the operator. The candidate solution can then be considered the local optima. The local optimum is used for the following iteration with a new move operator. Two move operators are defined. Two lists of surgeries are maintained one containing the surgeries that have been scheduled and the other the surgeries that have not yet been scheduled, these are referred to as un-served. The operators will either switch between two surgeries from either list or move one surgery from the assigned list of surgeries to any other day and room schedule or the un-served schedule. Using an initial solution to reflect the reality of hospital scheduling, the un-served surgeries are scheduled as

soon as possible. The priority is evaluated based on the referral date, the deadline for the surgery and the surgery duration.

Santibanez et al. (2007) propose a mathematical model to schedule surgeries in OR (operating rooms). The method proposed builds a complete mixed integer programming model that considers constraints relating to the number of OR, speciality equipment required by units, OR capacity, bed capacity for post-surgery usage, staff availability and surgery duration. The planning period is four weeks and the model aggregates information from 8 hospitals. Different scenarios are analysed, modelled and solved. These models vary depending on the objective function and constraints.

Guinet and Chaabane (2003) focus on assigning patients to operating rooms (OR) over a one to two week planning period. The model aims to minimise patient waiting time and overtime use of the OR. The model takes into account surgery duration, surgeon work requirements and availability, equipment needed for surgery and recovery room time. The problem is solved by defining a graph in which a set of nodes represent surgeries that need to be scheduled and a second set of nodes representing resources are linked to obtain the schedule of the OR. Each surgery must be assigned to resources while respecting the constraints.

Dinh-Nguyen and Klinkert (2008) model a surgery scheduling problem into a job shop scheduling problem. The problem involves scheduling surgeries for in and out patients. The objective is to minimise the total time taken for all surgeries to be completed considering the restrictions on hospital resources and the maximum time interval in which an operation can be scheduled. The duration time is static. The schedule also needs to accommodate as many surgeon's and their team's preferences. The planning horizon can vary from one day to one week. The model proposed is adaptable it can be used to include overtime when necessary, for hospitals and independent clinics. The model also includes emergency cases and adding elective cases to fill the surgery schedule on a given day. The model was solved using exact solvers, the authors conclude this is time consuming and can only be used when solving small problems.

2.3.2 Personnel Objectives

Gendreau et al. (2006) propose a general model for the scheduling of physicians in emergency rooms (ER). From information provided by six hospitals in Montreal, Canada the authors determine a general set of constraints that can reflect the scheduling of physicians in ER. The authors find that the problematic is similar to the nurse scheduling problem. The authors divide the constraints in four major categories. Constraints related to the availability of physicians and the requirements in ER for a given period. The second category concerns workload, they are limitations on the total of hours physicians can work in a given period and a maximum number of types of shifts that can be worked. The third category concerns fairness of the schedule, the distribution of the workload has to be distributed amongst physician considering different factors such as seniority. The last category is concerned with the welfare of the physicians. Four solution methods are proposed: a heuristic with partial branch and bound algorithm, a column generation approach, a tabu search algorithm and constraint programming.

Jebali et al. (2006) present a two-step approach. The first step consists of assigning surgeries to operating rooms (OR) for one day and the second step concerns the sequencing of the surgeries assigned to the OR. The model considers constraints relating to surgeons availability, work regulations, post recovery rooms availability, specialist equipment as well as preparation and cleaning time of OR. Both steps are solved using a mixed integer program.

Blake et al. (2002) proposes an Integer Linear Programming model to solve the block surgery allocation problem at Mount Sinai Hospital in Toronto. The problem consists of assigning operating room (OR) time to a block of surgeons. The model takes into account constraints relating to specialized equipment and personnel restrictions. The model also tries to assign to each unit the number of hours targeted and also to maximise utilisation of the OR.

Hsu et al. (2003) look at the problem of scheduling surgeries that have been assigned to a day. The goal is to minimise the number of post anaesthesia care unit nurses and to establish the optimal sequence of surgeries to ensure that the total duration time of all combined operations is minimised. The authors model the problem as a job shop scheduling problem where one stage is used to minimise the total duration time of the surgeries and the second step stage consists of minimising the number of nurses while respecting a maximum operational duration time for the day. A greedy algorithm finds an initial schedule and it is improved by a tabu search algorithm. The algorithm was used to solve real data from a university hospital; the results obtained are close to the optimal solutions.

2.3.3 Resource Objectives

Hans et al. (2008) aim assign a list of known surgeries to an operating room while minimising overtime for one day. The model and the solution methods were developed using data from Erasmus Medical Centre in the Netherlands. A slack period is added to the planned duration of each surgery, this slack is based on statistical information provided by the hospital. An initial solution is found by assigning to each surgery the first available place or to prioritise the surgeries by decreasing the order of duration and placing each surgery from the list at the first available place. Another possibility is to assign in the list of priorities a probability to each surgery this will determine the order of allocating surgeries to the schedule. Once the initial solution is found improvements are made based on two possibilities i.e. moving one surgery to another operating room or swapping two surgeries between two days.

Denton et al. (2007) create a stochastic model to account for the uncertainty of the duration of a surgery. The model is used to schedule surgeries for one day for one operating room. The model aims to decide the sequence of the surgeries and the scheduled start time of each surgery. Different heuristics are applied using various sequencing logic. The authors found that contrary to scheduling practices to schedule the longest surgeries at the beginning of a day did not provide the best results.

Belien and Demeulemeester (2007) present several models to create a master surgery. The objective is to minimise the expected bed shortage. The models consider the number of patients and the length of stay of patients (this includes surgery duration and post-surgery recovery time) as being stochastic. Different methodologies have been used to solve this problem. The models that have been formulated with a linear or quadratic objective function are solved with integer programming solvers. These models minimise the mean or the variance of the expected bed shortages. The model that considers the stochastic/undetermined nature of expected bed shortages is non-linear and is solved using simulated annealing.

2.4 Real world Nurse Rostering Benchmark Data Sets

As mentioned previously, the primary goal of this research is to provide more general methodologies to solve real world healthcare problems, more specifically, nurse rostering and surgery scheduling problems.

The personnel scheduling data used for this research are 43 well-known real world benchmark instances of the nurse rostering problem. The 43 different problems provide a good basis for evaluating the robustness and efficiency of the four hyper-heuristics developed. These will be presented in chapter 4. The nurse rostering problems include well known benchmark instances. The instances include problems from the UK, Canada, the US, Greece, Japan, Belgium, Norway and the Netherlands. As defined previously, the nurse rostering problem consists of creating a schedule for each nurse in a ward over a pre-defined period of time. The nurse rostering data used for this research is included in the hyper-heuristic framework HyFlex v1.0. HyFlex will be described in depth in Chapter 3. For this subsection the focus is on the datasets.

Each nurse rostering instance is modelled to include all constraints related to the instance in the objective function. A cost is attributed to each constraint reflecting the importance of the constraint. A constraint that must be satisfied will be attributed a higher cost than a soft constraint that may be violated. The

quality of the roster will be reflected in the cost of the solution. The goal is to minimise the total cost of the objective function.

Each nurse rostering problem is unique and the constraints vary greatly for each problem. However, three broad categories of constraints can be defined: the coverage, the hospital and work regulations requirements and the nurse preferences. Each of the constraints included in the three categories are unique to each instance, and each problem has a set of these requirements. For example, the work and contract regulations differ for different hospitals but each instance has a set of these requirements. Each problem has different skill categories, skill pairing, although these are different the requirements may all be grouped under the general title of coverage constraints. The third general grouping involves all the requests/preferences determined by each individual nurse. The benchmark instances as previously mentioned have been assembled from different countries where work/hospital regulations vary greatly, coverage requirements are also unique to each problem. Therefore the objectives and the number of objectives are different for each problem. The problems have been collected from various publications, and industrial collaborators ORTEC and SINTEF. The length of the scheduling period, the shift types and the number of employees to schedule varies from problem to problem. More information on the instances can be found in Curtois et al. (2010).

The work and contract regulations constraints include restriction on the total number of hours scheduled; the minimum and maximum number of hours of work and also the preferred number of hours to be worked by each nurse. This can be very complex as some wards have part time and full time nurses. Another requirement that is placed under this category is to do with hospital regulations; as an example a nurse scheduled to work on a late night shift cannot be assigned to an early shift the next day. It is also preferred that nurses work a minimum number of consecutive days before having time off. Nurses may also have a maximum allowed number of night shifts that can be assigned in a planning period. The coverage category ensures there is the required number of nurses in each skill category when needed. This number will have been established prior to determining the roster. As an example, a head nurse

will be required at periods during the day; another example can be that student nurses need to be paired with more experienced nurses, etc.

The third category encompasses all nurses' preferences. It is found that this category includes the most requirements, although, these are often considered soft constraints i.e. these can be violated, it is still important to fulfil as many of these preferences as possible.

The 43 real world instances include small, medium and large problems. Each group of instances will be summarised briefly (Curtois et al. 2010). The BCV set of problems are taken from hospitals in Belgium. The BCV problems schedule 20 to 100 nurses per ward. The planning period is between two days to a few months. Different skill categories of personnel are considered. The roster is modelled to consider a period of one week at a time. The coverage constraints and ensuring nurses are assigned only one shift per day are the only hard constraints. The soft constraints include all requirements relating to hospital and work contract regulations and nurse preferences.

For the GPost instances there are 8 employees to schedule, 4 full time nurses and 4 part time nurses. Two shifts must be filled, a day and a night shift. The scheduling period is for 28 days. There is a need of 3 nurses for each day shift and one nurse for each night shift. The hard constraints relate to work contracts and hospital regulations. The soft constraints concern undesirable schedules; as an example, scheduling non-consecutive days off. The SINTEF problem creates a schedule for 24 nurses, the planning period is 3 weeks and the schedule is built by shifts. The ORTEC problems contain 16 nurses with 4 shift types and a planning period of 31 days.

The QMC instances include 19 employees to schedule over 28 days and there are 3 shift types. The Ikegami2d problems are the 2 shift problems, the planning period is 30 days and there are 28 nurses to schedule. The Ikegami3d instances are for 3 shifts for 25 nurses to schedule over a period of 30 days. For the Ikegami problems skill categories are considered.

The Millar problem instances consist of a roster for 14 days with 2 shift types and 8 employees, over or under coverage is not permitted. The Azaiez

problem has 13 nurses, 2 shift types and the planning period is of 28 days, coverage requirements include skill level. The Valouxis instance consists of scheduling 16 nurses over 28 days with 3 shift types. For the WHPP problem, 30 nurses must be scheduled over a 2 week period for 3 shift types.

The LLR problem creates schedules for 27 nurses for a planning period of 7 days and there are 3 shift types. The Musa problem schedules 11 nurses over 14 days with one shift type and the minimum and the preferred required number of nurses is considered. The Ozkarahan problem consists of 14 nurses to schedule over a seven day planning period with 2 different types of shifts.

For the MER problem, 54 nurses and hospital staff must be scheduled over a 48 days planning period. There are 12 shift types, and the coverage requirements are specified by the period of the day. The CHILD problems schedule 41 nurses over a six week planning period and there are 5 shift types. The coverage requirement is also specified by the period of the day. The ERRVH problems need to schedule 51 nurses and hospital staff over a six weeks period with 8 different shift types. For this problem the coverage requirement are also specified by period of the day. For the ERMGH, the coverage requirement is also specified by the period of the day, and there are 41 nurses to schedule over a six weeks period with 4 different shift types.

Table 2.2 outlines the best known solution or optimal solution (BKN) for each instance. The optimal solution is marked in bold. The relevant sources and methods employed which have achieved the associated BKN for each instance is also presented when it is known.

TABLE 2.2 Nurse Rostering Benchmark Instances

Instances	BKN	Method(s) and Reference(s)
BCV1.8.1	252	Scatter Search (Burke et al. 2009b), VNS (Burke et al. 2008a)
BCV1.8.2	853	
BCV1.8.3	232	
BCV1.8.4	291	
BCV2.46.1	1572	Scatter Search (Burke et al. 2009b), VNS (Burke et al. 2008a)
BCV3.46.1	3280	
BCV3.46.2	**894**	Scatter Search (Burke et al. 2009b)
BCV4.13.1	**10**	Scatter Search (Burke et al. 2009b)
BCV4.13.2	**10**	
BCV5.4.1	**48**	Memetic Algorithm (Burke et al. 2001a), Tabu Search (Burke et al. 1999a, 1999b), Constraint Programming (Métivier et al. 2009), Scatter Search (Burke et al. 2009b)
BCV6.13.1	768	Scatter Search (Burke et al. 2009b)
BCV6.13.2	392	
BCV7.10.1	**381**	Scatter Search (Burke et al. 2009b), Hybrid IP and VNS (Burke et al. 2009f)
BCV.8.13.1	**148**	Memetic Algorithm (Burke et al. 2001a), Scatter Search (Burke et al. 2009b), Tabu Search (Burke et al. 1999a), Decomposition and Greedy LS (Brucker et al. 2010)
BCV8.13.2	**148**	
BCVA12.1	1294	
BCVA12.2	1953	
ORTEC01	**270**	Mixed Integer Program (Glass and Knight, 2010)
ORTEC02	**270**	Mixed Integer Program (Glass and Knight, 2010)
GPost	**5**	Mixed Integer Program (Glass and Knight, 2010)
GPost-B	**3**	Mixed Integer Program (Glass and Knight, 2010)
QMC-1	14	
QMC-2	**29**	
Ikegami2d1	**0**	Decomposition of problem and branch and bound (Ikegami and Niwa, 2003)
Ikegami3d1	**2**	
Ikegami3d1.1	**3**	
Ikegami3d1.2	**3**	
Millar2s1	**0**	Decomposition of problem and branch and bound (Ikegami and Niwa, 2003), Constraint Programming (Métivier et al. 2009), Scatter Search (Burke et al. 2009b)
Millar2s1.1	**0**	Scatter Search (Burke et al. 2009b)
Valouxis	**20**	
WHPP	**5**	Constraint Programming (Weil et al. 1995)
LLR	**301**	Scatter Search (Burke et al. 2009b)
Musa	**175**	Constraint Programming (Métivier et al. 2009)
Ozkarahan	**0**	Constraint Programming (Métivier et al. 2009)
Azaiez	**0**	Constraint Programming (Métivier et al. 2009), Linear Goal Program (Azaiez and Al Sharif, 2005)
SINTEF	**0**	
CHILD-A2	1095	
ERMGH-A	795	
ERMGH-B	1459	
ERRVH-A	2142	
ERRVH-B	3121	
MER-A	9017	Hyper-heuristic (Chan et al. 2012)
QMC-A	27	

All the nurse rostering problems described are different, a formal description of the problem can be made using an IP model. As an example the GPost problem will be selected and to requirements illustrated.

Parameters

A= Set of nurses available = {1, 2, 3, 4, 5, 6, 7, 8}.

At| t ϵ {1,2}= Subset of nurses that work part time and full time.

B= Set of available shifts= {0 *(no-shift)*, 1 *(day)*, 2 *(night)*}.

B'= Set of undesirable consecutive shifts={(2,1),(0,1),(0,2)}.

J = Days is scheduling period = {1,…,28}.

d_{jb} = Coverage required on day *j* for shift *b*.

m_a = maximum number of working days for nurse *a* for scheduling period.

n_a = minimum number of working days for nurse *a* for scheduling period.

h_1 = maximum number of consecutive day shifts.

h_2= maximum number of consecutive night shifts.

Decision variables:

x_{abj} = 1 if nurse *a* is working shift *b* on day *j*, 0 otherwise

As all constraints are modelled within the objective function. The objective function is the weighted sum of all *n* constraints:

$$\text{Min } O(x) = \sum_{i=1}^{n} \quad w_i O_{i(x)}$$

The only constraint is to ensure a nurse only works one shift per day

$$\sum_{a \in A} \quad x_{ajb} \leqslant 1, \quad \forall\, a \in A, j \in \{1,..,28\}$$

Where 2 examples of the goals are:

Maximum number of working days.

$$O(x) = \sum_{a \in A} \quad max\{0, \sum_{j=1}^{28} \quad \sum_{b \in B} \quad x_{ajb} - m_a$$

Shift cover requirements

$$O(x) = \sum_{j=1}^{28} \quad \sum_{b \in B} \quad | \sum_{a \in A} \quad x_{aib} - d_{jb} |$$

As the coverage demand is known: 3 nurses for a day shift and one nurse for the night shift every day. The shift cover requirements can be formulated as

$$O(x) = \sum_{j=1}^{28} \quad \sum_{b \in B} \quad | \sum_{a \in A} \quad x_{ai1} - 3 |$$

$$O(x) = \sum_{j=1}^{28} \quad \sum_{b \in B} \quad | \sum_{a \in A} \quad x_{ai2} - 1 |$$

From the brief overview of the 43 nurse rostering problems, it is clear that although all constraints in each problem falls under three headings (coverage, work/contract regulations and preferences) the problems are all different. These instances will be used to evaluate the level of generality within the same problem domain of the hyper-heuristics proposed.

2.5 Surgery Admission Planning Problem Data sets

The surgery scheduling issue is complex and varies greatly in different hospitals, clinics, wards and countries. The domain of surgery scheduling has not been studied as extensively as the nurse rostering problem within the research community, part of the issue is the fact that no consensus exists on the exact definition of the problem. Some definitions include nursing and associate members of staff, bed recovery time required, outpatients and inpatients surgeries. The planning horizon may also differ, where the scheduling may be done for one day of surgeries in a hospital, a week or a month. The field of surgery scheduling is also vast as many aspects need to be considered such as the planning of the surgery i.e. the initial schedule, the surgery itself which will involve variation in durations depending on the complexity of the operation, the patient's response, the equipment required will also be affecting this stage of the surgery to determine the operating theatre and the day of operation. The last stage is the recovery time required for the patients following the operation, often with complex interventions the

time required is greater than the duration of the surgery. This recovery time may involve a few days spent in the hospital.

In this field it has often been found that the real bottleneck of surgery scheduling i.e. delays in surgery scheduling is due to the fact that there are not enough beds available for patient recovery, this is also coupled with the shortage of staff such as nurses to take care of patients following the surgery. Unfortunately very little real data has been collected in this field that includes bed recovery time and assigning nurses to care for recovering patients. Concurrent to the scheduling of a surgery is the scheduling of surgeons and other team members such as anaesthetists and nurses. The surgery scheduling problem proposed for this research does not include all this information.

The interest in optimising surgery scheduling problems has risen. Surgeries are the primary cost and revenue service of a hospital; furthermore there is an increase in pressure on healthcare administrations worldwide to ensure that patients are treated in a timely manner. To ensure quality healthcare, there is an expectation that the waiting time of a patient for a surgery should be decreased. For example in the UK a quota exists in which a time interval is specified for a surgery, this interval will depend on the emergency of the intervention. The quota is based on the number of days between the surgery and the time the patient was referred to a specialist from their general practitioner.

In this thesis a new method is proposed to solve specifically surgery admission planning problems. The data for the surgery admission planning problem is provided by SINTEF and is defined by Riise and Burke (2011). The data is based on real world surgery admission planning problems obtained from a Norwegian hospital. The data reflects their knowledge of the actual problem faced by a surgical department in the hospital. The surgery admission planning problem is a two-component problem. Initially a set of surgeries are assigned to a day in the planning period. The second step consists of attributing a time and operating theatre to the surgeries planned for a specific day.

The surgeries are known. The duration, preparation, clean-up and recovery time within the operating theatre depend on the surgery and are static. These times are known. The surgeon is pre-assigned to the surgery. Each surgical team is divided by speciality. In the case of this research the specialities include general surgery, surgeries in urology and gastro surgeries. More specifically, there is a small, medium and large surgery possible in the gastrological ward. Only one type of operation is done in urology and there is only a small and a large surgery possible for the general surgical ward. Each initial surgery schedule i.e. assigning surgeries to a day of surgery in the planning period has some surgeries already booked to reflect a realistic picture of the problem. These have been booked from the previous planning period of 14 days.

Detailed information on to the surgeon's working period i.e. work days and days off is also known. There are 7 surgeons in total with 3 surgeons in gastro, 2 surgeons in urology and 2 surgeons for general surgery. Each surgeon can do any operation within their speciality. The planning horizon is 365 days. There are 4 operating rooms that can be used for elective surgeries. Each data set contains information relating to all patients awaiting surgery. For each patient the following information is available: the referral date, the maximum delay in operation, the patient's age, if the patient has diabetes, the number of previous cancellations, the surgeon that will operate, the duration of the surgery and the type of surgery. A preferred time window for the surgery is also included for each patient. The information regarding diabetes could also be replaced to indicate any other health condition. Preparation and post-operative tasks are known i.e. the time to clean equipment before and after surgery is included as well as the time to remove equipment. There are 10 instances of the surgery admission problem provided by SINTEF. The size of each instance is provided in Table 2.3, where the first column includes the instance and the second column represents the number of surgeries that need to be scheduled.

TABLE 2.3 Size of surgery admission planning instances

Instances	Surgeries
WOT1	152
WOT2	157
WOT3	158
WOT4	160
WOT5	162
WOT6	166
WOT7	170
WOT8	177
WOT9	180
WOT10	186

The data allows for each operating room to have a maximum of three hours and twenty minutes overtime i.e. over the maximum amount of time of work allowed per day for each surgeon. The operating rooms are only open when the surgeons are working; therefore no surgery is scheduled on a surgeon's day off. For the problems at hand the surgeons can all work the same days and the same number of hours.

The goal is firstly to assign a day and time for each elective surgery while keeping time for emergency surgeries. Secondly, each surgery is assigned to an operating room and resources. The total duration of each surgery in each operating room in a day must not exceed the opening time of the operating rooms. Opening hours of the rooms depend on hospital regulations. At the same time the surgery must be scheduled in a way to ensure that surgeons are assigned to one surgery at a time.

For this research, the surgery admission planning problem is defined as a combined objective model as proposed by Riise and Burke (2011). The problem is modelled in the same way as Riise and Burke to facilitate a comparison between their results and the results of the four hyper-heuristics strategies proposed in the thesis.

More formally the problem is defined in Table 2.4. Each goal has the same cost within the objective function.

TABLE 2.4 Surgery Admission Planning Model

Objectives	Minimise patient waiting time between referral and scheduled surgery
	Minimise surgeon overtime
	Schedule children early on day of surgery
Constraints	Surgeon is available on day and time of surgery
	OR is available on day and time of surgery
	Each surgery is scheduled only once
	Only one surgery is scheduled at a time in the operating room
	For each operating room, the sum of durations for all surgeries assigned to that OR does not exceed the operating hours of the assigned OR.
	Schedule surgery within the maximum delay between referral date and date of intervention.
	Surgeon can perform only one surgery at a time

For this problem domain, the problem evaluated is the same for all ten instances. The constraints and the model. The difference lies is the size of the problem. These instances will be used to evaluate the level of generality of the hyper-heuristics proposed within different problem domains.

2.6 Remarks

A variety of methods have been used to solve the nurse rostering problems. In the last few years, the focus has been on solving real-world problems and creating more general approaches. There has also been an increase within the field of healthcare to address the surgery scheduling problem. A review has been done on nurse rostering publications, these were grouped by methodology. The complexity of this problem makes it interesting to solve using a hyper-heuristic methodology. The datasets proposed will be used to determine the level of generality of the four hyper-heuristics presented in this thesis against unseen problem solving scenarios with different characteristic within the same problem domain. There is no consensus on the definition of the surgery scheduling problem; the models focus on different aspects or sub-problems, the publications were therefore grouped by the objectives emphasised. The datasets presented for this domain will be used to evaluate the performance of the hyper-heuristics against a variety of instances within different problem domains.

Chapter 3

Hyper-heuristics

Over the last decade, there has been an increase in the development of more general approaches to solve combinatorial optimisation problems. This has been done successfully through the use of hyper-heuristics for a diverse range of problem domains, such as personnel scheduling and timetabling problems (Burke et al., 2003a, Cowling et al., 2000).

A *hyper-heuristic* is an algorithm that operates on a search space of heuristics (Burke et al., 2003a). There are two main types of hyper-heuristics in the literature; hyper-heuristic methodologies that *select* or *generate* low level heuristics (Burke et al., 2009c). In this thesis, the described four hyper-heuristics are selection hyper-heuristics which are iterative *perturbation* based approaches, i.e. following the construction of an initial solution, it is improved step by step using a set of perturbative low level heuristics which are enabled to process and return complete solutions at any time (Burke et al., 2009c, 2009d). When designing such a selection hyper-heuristic two components need to be considered. The first component is the strategy employed by the hyper-heuristic to select a (subset of) low level heuristic(s) to perturb a given candidate solution. The second component is the defined criterion that enables the acceptance or rejection of a newly created candidate solution after application of a chosen low level heuristic. Any learning hyper-heuristic component gets feedback and maintains problem domain independent information to make better choices at a given decision point during the search process.

Many publications have been made on hyper-heuristics, the review of the research done within this more general methodology is not exhaustive but rather an overview to place in context the contribution of the research proposed

for this thesis[3]. Section 3.1 will focus on outlining selection hyper-heuristics. Section 3.2 highlights one example of generation hyper-heuristic. Section 3.3 will describe HyFlex v1.0, a hyper-heuristic framework that is used to evaluate the four proposed hyper-heuristics and a review of algorithms submitted to the Cross-domain Search Heuristic Competition 2011 will be made. Section 3.4 overviews related work and remarks are found in section 3.5

3.1 Selection Hyper-heuristics

Burke et al. (2009c and 2009d) reviewed the hyper-heuristic methods and problems solved by them. The paper also defines the principles of the methodology, classifies the hyper-heuristics that have been developed and provides a classification framework for future work on hyper-heuristics. A hyper-heuristic approach is a method that does not use problem specific information to solve a problem. The hyper-heuristic which can be different heuristics such as tabu search, simulated annealing or others will choose or generate heuristics. The heuristics chosen or generated will solve the problems. So a hyper-heuristic method is a general method that can be used to solve similar problems. Hyper-heuristics can be divided in two main categories, hyper-heuristics that choose heuristics to solve problems or hyper-heuristics that generate heuristics to solve problems. Subcategories of each main category are constructive or perturbative; this evaluates the method of building the solution by the heuristics. A constructive hyper-heuristic approach will start to build at each iteration a solution using the low level heuristics. A perturbative hyper-heuristic approach starts with an initial solution and improves it using the low level heuristics. Further classification is also done based on the move acceptance criteria where it can be a deterministic or non-deterministic. A deterministic criterion takes the same decision any time during the search process. A non-deterministic criterion involves taking a decision that can vary during the search process even while considering the same candidate solution.

[3] A more extensive review of work done on hyper-heuristics can be found in Appendix B.

This paper is innovative as it defines the hyper-heuristic methodology, reviews all hyper-heuristic approaches developed and provides a framework to understand the method and classify future hyper-heuristic work.

The publications outlined in this chapter will be grouped by the classification provided by Burke et al. (2009c). The papers highlighted are selection hyper-heuristics.

3.1.1 Online Learning Selection Hyper-heuristics Deterministic Move Acceptance

Burke et al. (2007a) present and evaluate a hyper-heuristic method to solve an exam timetabling problem and a course timetabling problem. The hyper-heuristic proposed determines the value of the heuristics and the order in which to apply the heuristics to provide a good solution i.e. a solution that is feasible and has the lowest cost. The hyper-heuristic constructs an initial solution by creating heuristic combinations, choosing the best and applying these to obtain the solution. Evaluated combinations are placed in a tabu list.

Qu and Burke (2009) compare heuristics to be used as a hyper-heuristic to solve a course and exam timetabling problem. The hyper-heuristics evaluated are the Steepest Descent Method, the Iterated Local Search (ILS) method, the Tabu Search method and the Variable neighbourhood Search method. The ILS method outperforms the other heuristics.

Ochoa et al. (2009a) analyse the search space of a hyper-heuristic in order to build the best sequences of low level heuristics. This is done on an educational timetabling problem. The authors found that the best solution to an instance of the problem are concentrated in a small area of the search space and that the first scheduled events are more important to solution quality than events scheduled at a later stage. In Ochoa and al. (2009b) the flow shop scheduling problem is used to evaluate the search space. The analysis reveals similar conclusions.

Cowling et al. (2000) present and compare the performance of a set of simple and mostly non-learning heuristic selection methods within a selection hyper-

heuristic framework on a scheduling problem. Simple random chooses a random low level heuristic. Random gradient heuristic selection employs simple random for choosing a heuristic, but the same heuristic gets invoked repeatedly as long as there is improvement. Random permutation heuristic selection uses all the low level heuristics and creates a random permutation with them, and then each heuristic is invoked successively. Random permutation gradient is based on random permutation heuristic selection with the difference that the same chosen heuristic is invoked repeatedly until there is no improvement. The greedy strategy applies all low level heuristics and chooses the one which generates the best improvement (or least worsening). A more elaborate online learning mechanism referred to as *choice function* performed the best among these heuristic selection methods. Choice function maintains a utility value for each low level heuristic. A heuristic with the maximum score is selected at each step. This score is a weighted average of three performance indicators. The first component relates to the previous improvement made by a low level heuristic. The second one considers the interdependencies between two low level heuristics and the third one looks into the last time when a given low level heuristic was used.

A widely used learning heuristic selection method within hyper-heuristics is based on reinforcement learning Di Gaspero and Urli (2012), Özcan et al. (2008), Nareyek (2003). Nareyek (2003) assigns a utility score for each low level heuristic. The proposed hyper-heuristic increases the score of the chosen and applied heuristic if there is improvement and decreases it, otherwise, at a certain rate. Different mechanisms, such as, max or roulette wheel can be used to select a low level heuristic based on their utility scores. Simple move acceptance methods include accepting all moves, only improving moves and improving and equal moves.

3.1.2 Online Learning Selection Hyper-heuristics Non-Deterministic Move Acceptance

Bai et al. (2007) evaluate the impact on the solution quality of the memory length of online heuristics. Online hyper-heuristics are hyper-heuristics that select low level heuristics (LLH) at each stage of the problem resolution based

on the past performance of the LLH. The authors note that the performance of all the LLH is always better at the beginning of the resolution and gets worse during the middle and late stages of the resolution. The authors wish to find the impact of different memory length on the quality of the solution. The method studied a simulated annealing hyper-heuristic. Tto find better solutions it is better to dynamically increase the learning rates during the resolution process.

Different threshold move acceptance have also been used in selection hyper-heuristics, such as great deluge and simulated annealing Burke et al. (2012), Kalender et al. (2012).

3.2 Generation Hyper-heuristic

Although the focus of the overview of hyper-heuristics is on selection hyper-heuristics it is worth outlining the following hyper-heuristic that generates low-level heuristics.

Burke et al. (2009e) explore the uses made of Genetic Programming (GP) as a Hyper-heuristic. The authors also define the framework for using GP as a hyper-heuristic method i.e. the steps that need to be taken in order to use GP to build low level heuristics that will find solutions to a given problem. The first step to build a hyper-heuristic with GP is to analyse the existing heuristics used to solve a set of problems i.e. to know the weaknesses and strengths of the heuristics. The second step involves evaluating the way in which each heuristic is used to solve problems. The third step is to determine how to evaluate the problem at different stages of the resolution. The last steps are to identify the settings of the parameters and the links that will be created between the variables. The paper illustrates the methodology using two sets of problems the Boolean satisfiability problem and the bin packing problem.

3.3 Cross-Domain Search Heuristic Competition

HyFlex v1.0 was recently used in the Cross-domain Heuristics Search Challenge 2011 (CHeSC) competition.

HyFlex (Burke et al., 2009a, Ochoa et al., 2012) is an interface which is implemented in Java as v1.0 for rapid development and research on hyper-heuristics. HyFlex v1.0 contains the implementation of six problem domains: one dimensional bin packing, personnel scheduling, permutation flow shop, boolean satisfiability, traveling salesman and vehicle routing problems. For each problem domain, a set of low level heuristics and instances were provided. HyFlex v1.0 was recently used in the Cross-domain Heuristics Search Challenge 2011 (CHeSC)[4] competition. Before the competition, four problem domain implementations (one dimensional bin packing, personnel scheduling, permutation flow shop, boolean satisfiability) with ten instances for each domain were made public. CHeSC used three public instances and two hidden instances for each domain during the competition. The implementations and instances for the traveling salesman and vehicle routing problem domains were hidden. The authors Burke et al. (2009a, 2010a) compare the CHeSC competition to a decathlon challenge in which the athletes or in this case the competing algorithms need to achieve good results in a timely manner across all problem domains. For each problem domain, a set of low level heuristics and instances were provided. The objective of the competition was to spur on the research in the development of more general purpose algorithms i.e. heuristics that do not take any problem specific information into account. Consequently, new general hyper-heuristic methods have been introduced. These methods along with the six problem domain implementations can be considered as a *benchmark* to evaluate the level of generality of new hyper-heuristics. The competition sought a general method that frequently provides the best median solutions to the problem instances across all HyFlex problem domains. The success of the four hyper-heuristics in regards to their level of generality will be evaluated by applying them to the

[4] CHeSC website: http://www.asap.cs.nott.ac.uk/external/chesc2011/defaulthh.html

competition instances and comparing them to the algorithms submitted to CHeSC Ochoa et al. (2012).

3.3.1 HyFlex

HyFlex is a Java software library that is an interface between the problem domain layer and high level methodologies to be implemented for solving a problem using a hyper-heuristic methodology (Burke et al., 2009a, 2010a, Ochoa et al., 2012). The purpose of this hyper-heuristic framework is to facilitate the development of a general algorithm that can solve problems within different domains. The hyper-heuristic framework enables the user to concentrate only on the implementation and the evaluation of a hyper-heuristic for one or more problem domains included in the software. For each domain, the software includes implementation of a set of different types of low-level heuristics, various instances, an algorithm to create an initial solution and more. The software allows users to maintain a list of previous solutions. In this thesis, four hyper-heuristics are implemented as an extension to HyFlex in order to be able to evaluate their level of generality and see how they perform as compared to the previously proposed algorithms. Although each category of low level heuristic is the same for each domain, the number of low level heuristics and the way they operate vary according to each problem domain. For each problem domain, four types of low level heuristics are included: cross-over heuristics, mutation heuristics, local search heuristics and ruin recreate heuristics. All four of these heuristic types are perturbative heuristics i.e. they start with an initial feasible solution and modify it. The cross-over heuristics combine two solutions to produce a new solution. The mutation heuristics randomly mutate elements of a solution to produce a different solution. The local search heuristics will from a given solution try to find solutions that improve the objective function. The ruin and recreate heuristics remove a section of the solution and recreate a new solution.

HyFlex provides six problem domains: personnel scheduling, one dimension bin packing, flow shop scheduling, Boolean satisfiability, vehicle routing and the travelling salesman problem. The personnel scheduling problems are nurse rostering problems. The nurse rostering problem consists of creating schedules

for each nurse in a ward over a pre-defined period of time. The schedules must respect constraints relating to hospital and work regulations, coverage requirements as well as nurse preferences Curtois et al. (2010). The one dimensional bin packing problem consists of packing a determined number of pieces in as few bins as possible while respecting the total weight capacity of each bin (Hyde et al. 2010a). The permutation flow shop problem consists of ordering the jobs to be processed on consecutive machines while ensuring that no machines stay idle when a job is ready to be processed and each machine can process only one job at a time. Once the order of the processing of each job is determined it cannot be changed Vazquez-Rodriguez et al. (2010). The Boolean satisfiability problem consists of assigning a value to variables that will enable a formula to be true. Each variable can only have a true or false value. In HyFlex, it is the maximum satisfiability problem that is defined i.e. the objective is to maximise the number of clauses that are satisfied (Hyde et al. 2010b). The general vehicle routing problem (VRP) consists of creating delivery routes for a fleet of vehicles ensuring deliveries in a timely manner as well as respecting vehicle capacity. The goal is to use the smallest fleet of vehicles. The problem domain implemented in HyFlex is the VRP with time windows, which means that a customer's delivery must be done within a timeframe Walker et al. (2012). The vehicles leave from one depot and the objective is to minimise the cost of the deliveries. The problem of the travelling salesman consists of designing an itinerary in which all the cities that need to be visited are visited only once. The goal is to minimise the distance travelled.

3.3.2 Problem domain implementations in HyFlex

For the personnel scheduling problem Curtois et al. (2010) each low level heuristic category contains meta-heuristics that have been previously successfully used to solve specific nurse rostering problems Brucker et al. (2010), Burke et al. (2008a, 2009b), Curtois et al. (2010). The mutational heuristic category includes three mutational heuristics. Each mutational heuristic will be doing a swap between shifts in the candidate solution. Either a swap of a shift or block of shifts between two nurses' schedules or a swap

between a shift or a block of shifts with another day's shift or block of shifts in one nurse's schedule. The last possible mutation is to add or delete a shift or a block of shifts to one schedule. This last heuristic is also used to initialise the solution.

The local search heuristics set contains five local search heuristics of which three are hill climbers that use the operators described above to improve the objective function. The swaps are tested on one shift to blocks of a maximum of five shifts. The other two local search heuristics are variable depth search heuristics Burke et al. (2008a). The variable depth search heuristics will try to improve one nurse's schedule at a time in a roster. By choosing to improve one schedule this will worsen another nurse's schedule, this last schedule is then improved by the algorithm, the schedules will all be evaluated and an attempt to improve them will be done. If no improvements occur, the algorithm returns to the original roster and chooses a different initial schedule to improve. In order to obtain improvements, swaps are done in the shift or block of shifts between days and nurses.

The crossover heuristic set contains three crossover algorithms. The first crossover heuristic Burke et al. (2001a) finds the best assignments in the two nurses' schedules being evaluated and chooses these to place in the new schedule. To evaluate the best assignments each shift is temporarily removed from the roster and the impact on the objective function is calculated, the best assignments are the ones for which the removal has the largest impact on the objective function. The second crossover Burke et al. (2009b) selects the shifts that are common to both parent rosters to place in the new roster, the next step is to take shifts from each parent solution until the new roster is complete. The last crossover creates a new roster by choosing only shifts that are common to both rosters being evaluated.

The last set of heuristics consists of the ruin and recreate heuristics Burke et al. (2008a). The heuristics in this category randomly un-assign shifts in one or more nurses' schedules. The schedules are rebuilt by trying to satisfy the preferences of these nurses to work specific days or shifts and the second

consideration will be to try to satisfy preferences relating to week-ends. Each heuristic un-assigns a different number of shifts.

For the one dimensional bin packing Hyde et al. (2010a) the solution is initialised by randomly ordering the pieces to be placed in the bins and then placing them in the first bin in which they fit. Two local search heuristics are included in the one dimensional bin packing problem domains. The first local search heuristic randomly swaps two pieces if there is space and the objective function is improved. The second local search heuristic takes from the lowest filled bin the largest piece and swaps it with a randomly selected smaller piece from another bin, if this is not possible the heuristic chooses two pieces that have a total smaller size and swaps these with the large piece. The mutational heuristics include three low level heuristics. The first mutational heuristic swaps two pieces from different bins, if one piece does not fit the heuristic places it in a new bin. The second heuristic in this category randomly selects a bin that has more pieces than average and splits the contents into two new bins. The last mutational heuristic selects the lowest filled bin, removes its content and packs it in other bins where possible. The ruin recreate heuristics include two low level heuristics. The first heuristic removes the content of the highest filled bins and refits them using a best-fit heuristic. The number of bins affected depends on the intensity of mutation parameter in HyFlex. The second heuristic does the same thing as the first except that it unpacks the lowest filled bins. The only crossover heuristic included for this domain is the Exon Shuffling Crossover heuristic Rohlfshagen and Bullinaria (2007). This algorithm segregates the total number of items that need to be packed into subsets ensuring that the union of all these subsets will include all items and each item only once. First the solution is initialised; all items are placed in the bins without allowing overflowing i.e. a random permutation of items is done and each item is placed in the first available bin. Then two subsets of the total items to be packed are chosen and merged, mutually exclusive bins are added first, remaining items are then added and the items that are duplicated are eliminated based on the costs.

For the permutation flow shop problem, the domain includes five low level mutational heuristics, two ruin and recreate heuristics, four local search

heuristics and four crossover heuristics Vazquez-Rodriguez et al. (2010). To find an initial solution, a random permutation is created. The second step consists of creating a new schedule; this is done based on the algorithm proposed by Nawaz et al. (1983). The total process time for each job is calculated and the two jobs with the largest total processing time are selected. Two partial schedules are created based on the possible sequence of these two jobs. The sequence with the lowest total process time is selected; this will determine the position of these two jobs in relation to each other. The job with the third largest total processing time is then selected. Its three possible positions are evaluated within a partial schedule. The sequence selected will be the one with the lowest total process time. The other jobs are then selected in the same way to complete the schedule.

The first mutational heuristic randomly selects a job and reinserts it at a different position. The second randomly swaps two jobs. The third heuristic swaps the entire permutation i.e. order of the jobs. The fourth heuristic creates a new permutation with the same heuristic used to initialise the solution. The last heuristic in this category swaps randomly a fixed number of jobs; this is determined by the intensity of mutation parameter in HyFlex. The first ruin and recreate heuristic randomly deletes a number of jobs from the permutation and reinserts these using the same algorithm to initialise the solution. The second algorithm in this category randomly deletes a number of jobs of the permutation and reinserts these using the same algorithm to initialise the solution in such a way that at each iteration the best sequence of jobs are reinserted. The local search low level heuristics include a steepest descent algorithm where at each iteration each job is removed from its current location and reinserted in all positions, the best improvement is accepted. The second local search algorithm removes each job and replaces it into the first available position that creates an improvement. This process is done for each job. The third low level heuristic randomly selects a number of jobs to remove from the permutation; these are then tested in all positions and placed in the position that provides the most improvement. The last local search heuristic also randomly removes a number of jobs from the permutation but replaces them at the first place that creates an improvement in the solution. The crossover low

level heuristics types include a precedence preservative crossover in which two solutions are taken and which will form a third solution. The child solution is originally empty, a vector is randomly filled. This vector will determine the order from which each job is taken from each parent. If a job is taken from one of the initial solution it is then deleted from the second initial solution until both initial solutions are empty. The partially mapped crossover algorithm is also included. A section of the sequence of jobs from both parents is selected and placed into each parent solution and the remaining jobs of each parent are then exchanged to complete the solution. Also included is a one point crossover in the parent solutions, one point is selected, it is the same for each parent and the jobs after this point are exchanged from both parents to form two new solutions. For the order crossover a portion of the sequence of one parent is combined to a portion of the second parent and the remaining jobs are added to the child in order to preserve the new sequence produced by the combination of both solutions.

For the maximum satisfiability problem nine low level heuristics are included in HyFlex Hyde et al. (2010b). The local search low level heuristics include two algorithms. The first local search heuristic consists of flipping randomly a variable and accepting the change if the solution is improved. The second local search heuristic selects randomly a variable from a broken clause and flips it. Again the change is accepted only if the solution is improved. The mutational heuristics use the same operators as the local search i.e. one mutational heuristic flips randomly a variable, the change is automatically accepted. The second mutational heuristic flips a variable from a broken clause and accepts the change in the variable. The third mutational heuristic flips the variable that most improves the solution. In the case where two variables provide the same improvement, the tie is broken randomly. Another mutational heuristic uses the same logic except when they are ties; the variable that has been flipped the most times is selected to be changed. From a broken clause this mutational heuristic will flip a variable that does not have any impact on the solution. If no variables fulfil this characteristic a random variable will be changed with a probability of 0.5, if a variable is not chosen, the variable that when flipped has the least negative impact will be changed.

The last mutational heuristic will seek from a random broken clause the variable that when changed will improve the most the solution and change it. The ruin recreate low level heuristic used will randomly change a set of variables. The crossover heuristics either change one variable or two variables within the formula.

For the vehicle routing problem twelve low level heuristics are included in Hyflex Walker et al. (2012). The mutational operators for the four mutational heuristics do swaps between either two customers on a single route, two adjacent customers in a single route, place one customer from one route to another route and lastly, swap two clients from different routes. Two ruin and recreate heuristics are included. Both heuristics remove customers based on time or location. The route is rebuilt and only feasible solutions are created. The local search heuristics explore the following moves to improve the objective function value. The possible moves are a change of route for one customer, a swap between two customers on different routes, a swap between the end of two routes and a customer is taken from one route and placed into another route between the two customers closest in proximity to it. For the cross-over heuristics a random number of routes are considered from one solution and are combined with routes that do not create a conflict from the second solution, the customers left are inserted in the new solution. The second cross-over heuristic combined two solutions by placing the longest routes from both solutions to create the new solution, feasibility is kept. The customers that are left are inserted in the new solution.

3.3.4 CHeSC

The algorithms submitted to the competition were evaluated using the Formula 1 (F1) scoring system in which points are attributed to the first 8 top competitors. The best ranking algorithm is given a score of 10, the second best ranking 8 followed by 5, 4, 3, 2 and 1. The algorithms that are not in the top 8 competitors are not allocated any points. Each hyper-heuristic is ranked using the median objective function value from 31 runs for each instance. The maximum score for each algorithm for each problem domain is 50, since there

are five instances for each domain. All twenty hyper-heuristics competed across six problem domains. The results are provided in Table 3.1.

Table 3.1 The rank, name and F1 score of each hyper-heuristic which joined CHeSC

Rank	Hyper-heuristic	Score	Source	Rank	Hyper-heuristic	Score	Source
1	AdapHH	181.00	Misir et al. (2012)	11	ACO-HH	39.00	Núñez and Ceballos (2011)
2	VNS-TW	134.00	Hsiao et al. (2012)	12	GenHive	36.50	Cichowicz et al. (2012)
3	ML	131.50	Larose (2011)	13	DynILS	27.00	Johnston et al. (2011)
4	PHUNTER	93.25	Chan et al. (2012)	14	SA-ILS	24.25	-
5	EPH	89.75	Meignan (2011)	15	XCJ	22.50	-
6	HAHA	75.75	Lehrbaum and Musliu (2012)	16	AVEG-Nep	21.00	Di Gaspero and Urli (2012)
7	NAHH	75.00	Mascia and Stutzle (2012)	17	GISS	16.75	Acuña et al. (2011)
8	ISEA	71.00	Kubalik (2012)	18	SelfSearch	7.00	Elomari (2011)
9	KSATS-HH	66.50	Sim (2011)	19	MCHH-S	4.75	McClymont and Keedwell (2011)
10	HAEA	53.50	Gomez (2011)	20	Ant-Q	0.00	Khamassi (2011)

Misir et al. (2012) joined CHeSC with a learning hyper-heuristic (AdapHH) which became the winner of the hyper-heuristic competition. The method is a multi-phase approach which adaptively decides on the subset of low level heuristics to use at each phase and its duration. The heuristic selection computes a *quality index* for each heuristic based on a weighted average of a performance measure. This measure uses the number of new best solutions found, the total number of improvement and worsening until a given time, and during a single phase, overall remaining time, time spent by a heuristic until that moment and also during a phase. A heuristic gets excluded with a value below the average at a given stage or if it is relatively slow. A probability vector based on number of new best solutions found, remaining time, overall time and time spent is maintained to choose an active low level heuristic at each step. This elaborate hyper-heuristic also follows the performance of successive applications of heuristics in pairs. The choice for this relay hybridisation is probabilistic and these values are maintained via a learning automaton. The parameters of heuristics are also controlled via a reinforcement learning mechanism. The move acceptance component of the hyper-heuristic accepts improvements. A worsening solution is accepted if a new best solution

cannot be found after a certain number of iterations (which is adapted during the search process) with consecutive worsening solutions. More on the hyper-heuristic components and their analyses can be found in Misir et al. (2012). Although the approach is the winner of CHeSC, it is important to note that this hyper-heuristic is a complicated method embedding many parameters which are tuned and fixed before execution.

The hyper-heuristic developed by Hsiao et al. (2012) for the CHeSC competition was based on an iterated local search framework. The overall time is split into two. During the first phase, the hyper-heuristic starts by applying a mutational or ruin and recreate type low level heuristic to a population of initial solutions. Then all local search heuristics are applied until there is no further improvement. During the next phase, the best solution found so far as a single solution is used. Mutational heuristics are put into a circular priority queue based on their capability of severity of change. A perturbation is followed by a local search. This hyper-heuristic ranked the first in the Personnel Scheduling problem domain and second in the overall.

The hyper-heuristic created by Larose (2011) also relied on intensification and diversification components explicitly during the search process. An initially generated solution passes through a diversification stage via application of a mutational or ruin-recreate heuristic. Then this solution is improved using a local search heuristic until no further improvements can be achieved. A new solution is accepted only if there is improvement or the solution has not improved over the last 120 iterations.

Chan et al. (2012) developed a hyper-heuristic which relies on combining intensification and diversification components properly during the search process. From a set of candidate solutions an intensification process is made using the local search algorithms available in HyFlex. When there are no possible further improvements a diversification is made using another type of low level heuristic. Combinations of sequences of low level heuristics are evaluated.

Meignan (2011) proposed a hyper-heuristic based on a co-evolutionary algorithm and joined the CHeSC competition. Two set of populations are

used. One of them contains a set of solutions, while an individual in the other population represents a sequence of heuristics. A set of sequence of heuristics is co-evolved with a set of candidate solutions to improve o the solutions using the best sequence of heuristics during the search process.

Lehrbaum and Musliu (2012) proposed another hyper-heuristic based on an iterated local search framework. A roulette wheel strategy is used to choose a mutational low level heuristic based on the relative performance of heuristics during the diversification phase. Each low level heuristic is applied in descending order of previous performance and non-worsening solutions are accepted during the intensification phase. Poor performing low-level heuristics are temporarily placed on a tabu list with a given probability at each step of the search process. This hyper-heuristic performed particularly well on instances of MAX-SAT and personnel scheduling.

Mascia and Stutzle (2012) developed a hyper-heuristic based on an iterated local search framework which allowed restarts with a fixed probability. A randomly selected local search heuristic is applied successively for a number of times after the successive application of randomly selected ruin-create heuristic for a number of times. A worsening solution is accepted with a given probability at those stages. After the application of local search, a random mutational heuristic is employed with a given probability. Mascia and Stutzle show that the performance of the algorithm could be improved even further using different heuristic templates and parameter tuning. The original hyper-heuristic performed well on bin packing and permutation flow shop instances.

Kubalik (2012) presented two hyper-heuristics including the evolutionary based approach, ISEA which was used in CHeSC. Both hyper-heuristics randomly choose between ruin recreate, mutational and local search low level heuristics in a certain order. ISEA evolves sequences of low level heuristics, where each sequence is a permutation of low level heuristics using a local search heuristic as the first and last entries. All moves are accepted during the search process. This method performed well on the one dimensional bin packing instances. In Kubalik (2012), an improved variant of the hyper-heuristic is described as well.

Sim (2011) provided a hyper-heuristic which ignored the crossover heuristics. Each low level heuristic is selected based on its rank which depends on the improvement by the heuristic. The worst performing low level heuristic is placed in a tabu list. The tournament mechanism with a tour of size 2 is used to select a low level heuristic based on their ranks. Simulated annealing is used as the move acceptance component.

The hyper-heuristic proposed by Gomez (2011) made use of the evolutionary process based on a reinforcement learning scheme. The method maintains a set of all low level heuristics and another subset containing local search heuristics only. At each step, the proposed hyper-heuristic selects a local search heuristic and then another heuristic to apply to a candidate solution. Each low level heuristic is associated with a selection probability. If the chosen low level heuristic generates an improvement, then the selection probability of that heuristic is increased. If the selected low level heuristics generate a worsening solution, both of their selection probabilities are decreased. The author described a way adaptively deciding the depth of search and intensity of mutation parameters.

Núñez and Ceballos (2011) presented an Ant Colony Optimization inspired hyper-heuristic which aimed at improving a solution through a sequence of low level heuristics at each step. Each link between a pair of heuristics in the sequence is strengthened depending on the change in the objective function value after their applications to the solution at hand.

Cichowicz et al. (2012) provided a hyper-heuristic inspired from the behaviour of bees. A set of sequence of low level heuristics and initial solutions are created. A subset of sequences is selected and is randomly applied to the initial solutions. If the solutions are improved by a sequence, the sequence is allowed to continue searching for potential solutions. If the sequence is not satisfactory, a new sequence replaces it.

Johnston et al. (2011) joined CHeSC with a hyper-heuristic based on iterated local search. The diversification and intensification processes are explicitly enforced. The diversification process uses ruin-recreate or mutational heuristics. Each low level heuristic in these categories is associated with a

weight which determines its selection probability. The weights are updated during the search process based on the performance of the low level heuristics. After each operation using a selected mutation heuristic, a local search heuristic is applied to the solution at hand.

Di Gaspero and Urli (2012) provided a reinforcement learning based hyper-heuristic. Each heuristic is associated with a rank based on the improvements made by a low level heuristic in a specific solution phase as well as the time taken to find a candidate solution and a pairing between low level heuristics. A tabu list is also maintained to keep track of poorly performing low level heuristics with a low ranking. The performances of different reinforcement learning mechanisms are compared.

Acuña et al. (2011) proposed a hyper-heuristic improving an initial solution via applications of randomly selected low level heuristics. A temperature parameter is used to decide whether the selected low level heuristic continues the search process or another low level heuristic should be selected. This hyper-heuristic supports restarts when the search process stagnates.

Elomari (2011) described a hyper-heuristic performing a population based search. A low level heuristic is randomly chosen based on a distribution and applied to each solution. The best new solutions are accepted and included in the population. A selection probability is attributed to each low level heuristic depending on its capacity to diversify and intensify a solution.

McClymont and Keedwell (2011) wish to determine the best sequence of low level heuristics to use for each problem. The algorithm developed is named single objective Markov chain hyper-heuristic (MCHH-S), where each low level heuristic is considered a state and a probability is attributed to each possible sequence i.e. each low level heuristic is graphically represented as a vertex and en edge links each vertex including an edge that links the vertex to itself. Each vertex is given a probability that represents the possibility of using the next low level heuristics in the solution process. An initial population of solutions is determined, a random heuristic is applied to a random solution in the population, if the candidate solution is improved the child solution is returned to the population and the probability of using the low level heuristic is

increased. The next low level heuristic to use is selected randomly and it is applied to the next current solution in the population.

Khamassi (2011) introduced an ant-based hyper-heuristic. A directed graph representation for the successive selection of low level heuristics is utilised. Each vertex in the graph represents a low level heuristic and the heuristics are selected based on the choice made by travelling ants. Each tour of the graph is evaluated and the best ones are kept. The pheromone placed on each edge is updated according to the results obtained, i.e.; the best sequence of heuristics will have a higher pheromone rate.

3.4 Other Studies

Burke et al. (2010a) compared the performance of seven different hyper-heuristics using HyFlex on three domains; personnel scheduling, one dimensional bin packing and permutation flow shop. The first hyper-heuristic was an iterative local search algorithm which used hill climbing heuristics in each problem domain. The hyper-heuristic improved a solution at each iteration until no further improvements could be made. The other six hyper-heuristics combined two heuristic selection methods with three move acceptance methods. The first heuristic selection method used was simple random. The second method used a dynamic tabu list and assigned a score (rank) to each low level heuristic. The heuristic with the highest score that is not in the tabu list is selected and applied to the problem. If the solution is improved, then the rank of this low level heuristic is increased, if not, this low level heuristic's score is decreased. The first proposed acceptance criterion accepts all improvements and deteriorations with a 5% probability. The second approach also accepts improvements and no deterioration, initially. However, after 0.1 second, if there is no improvement, deteriorations are accepted at the rate of 5%, if after a further 0.1 second, still no improvement is obtained, the deterioration rate accepted is increased to 10%. The third acceptance criterion is based on a threshold acceptance method known as great deluge. The best overall performance was obtained using the iterative local search hyper-heuristic across all domains.

Burke et al. (2011a) implemented two hyper-heuristics in HyFlex and tested them on five instances from the following problem domains: personnel scheduling, permutation flow shop, one dimensional bin packing problem and maximum satisfiability problem. Two hyper-heuristics based on the same iterative local search framework as in (2010a) embedding two different heuristic selection methods are investigated. The first one is choice function and the second one is a reinforcement learning mechanism which rewards operators proportional to the change in the objective function. A high improvement assigns a high reward to the move operator. Each reward for each operator is accumulated and the probability of an operator being selected increases with the reward. The second technique provided better results.

Özcan and Kheiri (2011) described a hyper-heuristic method applied to the Hyflex training instances provided prior to the CHeSC competition. A smaller subset of *useful* low level heuristics is determined and put into a list of active heuristics using a greedy strategy and dominance which considers the trade-off between the improvement achieved by a heuristic and the time required for that achievement. For example, an extremely quick and effective heuristic which improves a solution reasonably is considered to be the same as a slow and extremely effective heuristic. Using the active list, a low level heuristic is selected using random gradient Cowling et al. (2000). When this second stage stagnates, the algorithm goes back to the greedy approach to reprocess all low level heuristics and decide which ones will be used in the next stage. This hyper-heuristic performed better than eight previously proposed hyper-heuristics which were put into a mock competition prior to CHeSC by the organisers.

Drake et al. (2012) presented a variant of *choice function* which proposed a different scheme for adjusting the weightings of choice function performance indicators. This variant outperformed the generic choice function Cowling et al. (2000) on the CHeSC benchmark.

Loudni (2012) created a modified iterative local search approach that addressed the trade-off between intensification and diversification processes for graph colouring. The approach determines the size of a perturbation step

and allows local search algorithms to run for a predetermined number of iterations. Each time the solution is improved, a large perturbation is employed and the number of iterations during local search is diminished. When the candidate solution does not improve for a certain number of iterations, a small perturbation is made. The algorithm is applied to a benchmark of graph colouring problem instances. The proposed approach delivers a comparable performance to the best known algorithms.

Kheiri and Özcan (2013 under review) proposed another hyper-heuristic based on a round-robin strategy for neighbourhood selection, namely; Robinhood hyper-heuristic. The authors used all low level heuristics provided in HyFlex. The low level heuristics are ordered by category as in mutational, crossover and hill climbing. The ruin and recreate heuristics are considered as mutational heuristics. Then low level heuristics are chosen randomly from each category and applied to the candidate solution successively until there is no further improvement. The heuristic selection component allocates equal execution time from the overall time for each low level heuristic, while the move acceptance component enables partial restarts via an automatically adjusted acceptance probability rate, when the search process stagnates. The Robinhood hyper-heuristic would have ranked the fourth in the competition.

3.5 Remarks

As can be ascertained, many hyper-heuristic methodologies have emerged over the last few years. The CHeSC competition has encouraged the exploration of this method to successfully solve six different problem domain and unseen instances in each domain. A review of selected work done on selection hyper-heuristics and specifically within the CHeSC competition was made in order to place in context the four hyper-heuristics proposed in this thesis. The four hyper-heuristics presented can be categorised in two more general strategies. All four hyper-heuristics can be classified as selection hyper-heuristics with deterministic move acceptance criteria. The four easy-to-implement selection hyper-heuristics are based on iterated and greedy search strategies. These hyper-heuristics are easy to implement. They necessitate setting of less number of parameters when compared to many of the existing approaches. The

empirical results on the benchmark problems included in CHeSC will be presented in Chapter 7.

Chapter 4

Analyses of Problem Domain Implementation in HyFlex

This chapter provides a background on the design of the hyper-heuristic strategies proposed to solve seven problem domains. As mentioned previously for six problem domains the hyper-heuristics are developed using the HyFlex framework. To establish strategies, it is important to understand the way in which each low level heuristic works. This implies an evaluation of the performance of each low level heuristic as well as evaluating the impact of specific ordering and combinations of low level heuristics. Subsection 4.1 presents the initial experiments done on four problem domains included in HyFlex i.e. the Boolean maximum satisfiability, the permutation flow shop, the one dimensional bin packing and the nurse rostering problems. The subsection 4.2 looks briefly at performance of the low level heuristics designed for the surgery admission planning problem. These are not included in the HyFlex framework, they have been developed specifically for the thesis. Remarks are included in section 4.3.

4.1 Initial experiments on the HyFlex low level heuristics

Prior to developing the hyper-heuristic strategies presented in this thesis, other approaches were used as inspiration. As previous studies have shown local search approaches can be very powerful tools. The iterated local search hyper-heuristic used by Burke et al. (2010a) established this method as being the winner when compared to other more complex methods. Burke et al. (2010a) applied their hyper-heuristics to instances of different problem domains included in HyFlex. Following the results of Burke et al. (2010a) and after having tested different types of low level heuristics included in HyFlex two main hyper-heuristic strategies are developed. From these two strategies four unique and novel hyper-heuristics are created. This chapter will focus on

explaining the approaches by providing details of the underlying experiments done and the results obtained prior to creating the four hyper-heuristics.

In order to develop the two main hyper-heuristic approaches, all low level heuristics from each problem domain were tested. The initial experiments and the hyper-heuristic strategies were tested and developed for healthcare issues, more specifically, the nurse rostering problem. The tests were therefore, conducted using the 43 real world benchmark nurse rostering problems available in an extended library of HyFlex. As mentioned previously, the nurse rostering problem has been researched over the last 45 years in the area of operational research. The 43 real world benchmark problems were chosen to conduct the initial tests as opposed to using surgery scheduling problems because there is no current general definition of the surgery scheduling problem. No consensus exists on the definition of the surgery scheduling problem and the data sets that can be accessed do not constitute a wide array of different problems from different hospitals and countries as is the case for the nurse rostering problem.

Once these preliminary experiments were completed, public instances for other problem domains were obtained from the CHeSC competition website. When completed, the tests enabled the evaluation of potential approaches and their level of generality. The goal was to evaluate and understand the low level heuristics behaviour during different stages of the search (or solving) process.

The same methodology was used for all experiments described in this chapter. The experiments were repeated 31 times for all instances of the forty three nurse rostering problems. The same experiments were also done using 10 public instances included in the CHeSC competition with the goal of understanding their behaviour during different stages of the search process, these were repeated 31 times. The primary goal was to evaluate each low level heuristic individually and to find which low level heuristics performed better than the others. Each time, a low level heuristic was successfully invoked for 1000 iterations to observe how they perform at the start of the search process.

For the first set of experiments, starting from a random initial solution, each individual low level heuristic was applied individually iteratively. The improvement is calculated between the current candidate solution and the previous candidate solution. Table 4.1 to Table 4.4 provides, as an example, the improvement between the initial solution and the final result for one instance of four problem domain evaluated i.e. the one dimensional bin packing, nurse rostering, the Max-SAT and the permutation flow shop problem domains. The problem instances used as examples are selected based on their level of complexity and the overall size of the problem. Each column represents the total improvement achieved by applying the low level heuristic analysed iteratively over each run between the initial candidate solution and the final solution in terms of objective function value. The first cell represents the improvement between the initial random solution and the final solution after 31 runs i.e. 1000 iterations/run. The following notation is used for the remainder of this chapter, LS denotes a local search algorithm, MU represents a mutational algorithm, RR denotes a ruin and recreate algorithm and CO indicates a crossover heuristic, the indices defined in HyFlex to identify each heuristic are added. If a negative value is encountered it means the current solution is worse than the previous solution. The objective function values obtained between each run by low level heuristic for each problem domain can be found in Appendix A.

TABLE 4.1 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each low level heuristic for one instance, of the one dimensional bin packing domain.

LS4	LS6	MU0	MU3	MU5	RR1	RR2	CO7
3.8%	4.0%	3.8%	3.9%	2.3%	3.6%	3.0%	1.8%

TABLE 4.2 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each low level heuristic for one instance of the nurse rostering problem.

LS0	LS1	LS2	LS3	LS4	CO8	CO9	CO10	MU11	RR5	RR6	RR7
18.0%	19.0%	22.0%	24.0%	17.0%	7.0%	8.0%	9.3%	2.9%	4.2%	3.2%	3.9%

TABLE 4.3 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each low level heuristic for one instance of the Max-SAT problem.

LS7	LS8	MU0	MU1	MU2	MU3	MU4	MU5	RR6	CO9	CO10
13.0%	12.0%	4.9%	3.6%	2.1%	2.9%	3.1%	0.3%	0.2%	2.8%	2.1%

TABLE 4.4 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each low level heuristic for one instance of the permutation flow shop problem.

LS7	LS8	LS9	LS10	CO1	CO1	CO1	CO1	MU 0	MU 1	MU 2	MU 3	MU 4	RR 5	RR 6
11.0 %	9.9 %	6.9 %	6.7 %	3.1 %	2.6 %	1.8 %	0.9 %	3.7 %	2.9 %	1.9 %	2.0 %	1.0 %	4.0 %	2.9 %

A few conclusions can be drawn from these sets of experiments. The first observation is that the local search heuristics from all domains are more powerful and enabled the best overall improvement on the initial solution when compared to any other type of heuristics. Following these results, the hyper-heuristics created will start with an iterated local search. This will enable the candidate solution to be used in the rest of the algorithms to already be a better solution than can be achieved consistently with any other low level heuristic.

HyFlex does not provide in depth information on the category of all low level heuristics. A heuristic in the crossover or ruin and recreate categories could be a local search (hill climbing) or a mutational heuristic. Looking at the behaviour of heuristics enabled the identification of the category of all low level heuristics as local search or mutational heuristics. When only improvements are made to the candidate solution, the low level heuristic is defined as a hill climber. If a worsening candidate solution is allowed, the low level heuristic is considered a mutational heuristic. The first and second sets of experiments showed that all ruin and recreate heuristics are mutational except one ruin and recreate heuristic included in the permutation flow shop domain. This heuristic has behaved from our experiments as a local search heuristic, always finding a non-worsening candidate solution. Table 4.5 provides the category of each ruin and recreate and cross-over low level heuristics.

TABLE 4.5 Category of cross-over and ruin and recreate type heuristics for each domain

	Mutational	Local Search
1D Bin packing	RR1, RR2	CO7
Perm. Flow Shop	RR5, CO11	RR6, CO12, CO13, CO14
Max-SAT	RR6, CO9, CO10	
Pers. Sched.	RR5, RR6, RR7	CO8, CO9, CO10

A second set of experiments is conducted, where a time limit of 30 seconds is set as a termination criterion to obtain a good quality solution for a given instance. At each step, a heuristic is applied to a solution at hand and an improving solution is accepted. The reason for these tests was to ascertain how quickly a good solution can be achieved by each low level heuristic in each domain. Tables 4.6 to 4.9 illustrate the total improvements between the initial solution and the candidate solution found after 30 seconds. The same instance for each problem domain is illustrated in this experiment as was in the first set of experiments. Detailed tables containing the improvement at each step is included in Appendix A.

TABLE 4.6 Results obtained after 30 seconds for each low level heuristic for the one dimensional bin packing problem, one instance.

LS1	LS2	MU1	MU2	MU3	RR1	RR2	CO1
2.7%	3.1%	2.7%	1.4%	1.3%	1.2%	1.4%	0.5%

TABLE 4.7 Results obtained after 30 seconds for each low level heuristic for the nurse rostering problem, one instance.

LS0	LS1	LS2	LS3	LS4	CO8	CO9	CO10	MU11	RR5	RR6	RR7
14.0 %	16.0%	18.0 %	16.0 %	4.8%	5.2%	5.4%	6.1%	1.2%	2.1%	1.6%	1.3%

TABLE 4.8 Results obtained after 30 seconds for each low level heuristic for the Max-SAT, one instance.

LS7	LS8	MU0	MU1	MU2	MU3	MU4	MU5	RR6	CO9	CO10
8.6%	7.9%	2.6%	2.1%	1.9%	1.8%	2.2%	0.0%	0.0%	0.6%	0.1%

TABLE 4.9 Results obtained after 30 seconds for each low level heuristic for the permutation flow shop, one instance.

LS 7	LS 8	LS 9	LS1 0	CO 11	CO 12	CO 13	CO 14	MU 0	MU 1	MU 2	MU 3	MU 4	RR 5	RR 6
9.7 %	6.5 %	5.4 %	3.4 %	0.7 %	0.8 %	0.0 %	0.0 %	1.0 %	1.3 %	1.3 %	0.2 %	0.0 %	1.1 %	0.9 %

As is seen from the results, the improvements using the local search algorithms on an initial random solution are found quickly, within 30 seconds often the search stagnates or does not find big improvements, the bigger improvements to the candidate solution can be found at the beginning of the search process. This has been true for all domains. The only exceptions are the large instances of the nurse rostering problems. As an example, the benchmark nurse rostering instance, MER-A is a larger and more complex problem than most of the benchmark instances, the schedule for this ward is determined over a 48 days period, 12 shift types are defined and 54 members of personnel of different skill categories must be scheduled. The schedule for each person needs to reflect their work contract, the hospital's regulations, the personnel's preferences and the coverage requirements. In this case, the iterated search process was found to find a local optimum solution in 14 minutes on average; this is using all 5 local search heuristics available for the nurse rostering domain.

The third set of experiments consists of starting the search process with the best candidate solution found from the second experiment instead of a random initial solution. Then each heuristic is tested to observe how a heuristic behaves starting from a locally optimum solution. A summary of the average percentage of improvement for each low level heuristic for the four problem domains is provided in Table 4.10. In each column the average improvement in percentage starting from a random initial solution is included; it is followed by the average improvement percentage when starting from a better initial solution (found through the second set of experiments). Appendix A includes a detailed table of the improvement between the initial solution and the current candidate solution between each run.

TABLE 4.10 Average percentage improvement for each low level heuristic.

	1DBin Packing		Perm. Flow Shop		Max-SAT		Pers. Sched.
MU0	4,4	MU0	4,4	MU0	5,5	MU11	3,3
MU3	2,3	MU1	3,3	MU1	3,3	RR5	4,4
MU5	2,2	MU2	2,2	MU2	2,3	RR6	3,4
RR1	3,3	MU3	2,3	MU3	2,2	RR7	4,4
RR2	2,2	MU4	1,2	MU4	3,4	LS0	15,15
LS4	5,6	MU5	2,3	MU5	3,3	LS1	16,16
LS6	4,5	RR5	5,5	RR6	3,3	LS2	15,16
CO7	2,3	RR6	3,4	LS7	11,12	LS3	16,17
		LS7	10,12	LS8	11,11	LS4	18,18
		LS8	10,11	CO9	3,4	CO8	8,8
		LS9	8,9	CO10	2,2	CO9	8,9
		LS10	7,7			CO10	9,10
		CO11	3,3				
		CO12	2,2				
		CO13	2,3				
		CO14	1,2				

Starting with a better initial solution did not necessarily improve the search process. The same conclusions from the first set of experiments can be made for these third sets of experiments. The local search heuristics enabled the best overall improvement on the initial solution. However, it can also be ascertained that on certain domains from one random or better initial solution, the local search heuristics have not provided as much improvement on the solution as in other domains, as an example this is more prevalent when solving a one dimensional bin packing problem. The local search heuristics for this domain are not as powerful as for other domains, such as the nurse rostering domain.

Another idea explored is to start the search process with multiple initial solutions. A few initial tests were done to measure the impact on the search process when more than one initial solution is used. The goal is to find if when using multiple solutions followed by the application of an iterative local search process, one of the final solutions obtained is better than the other.

To establish the optimal number of initial solutions that should be used, extensive experiments were done on 43 personnel scheduling benchmark

instances. It is found that using two initial random solutions followed by an iterative local search step provided an average improvement of 20% in the objective function value between both final solutions. The experiments were done 31 times on all 43 benchmark instances. The total computational time required increased on average by 20% when working with two candidate solutions instead of one.

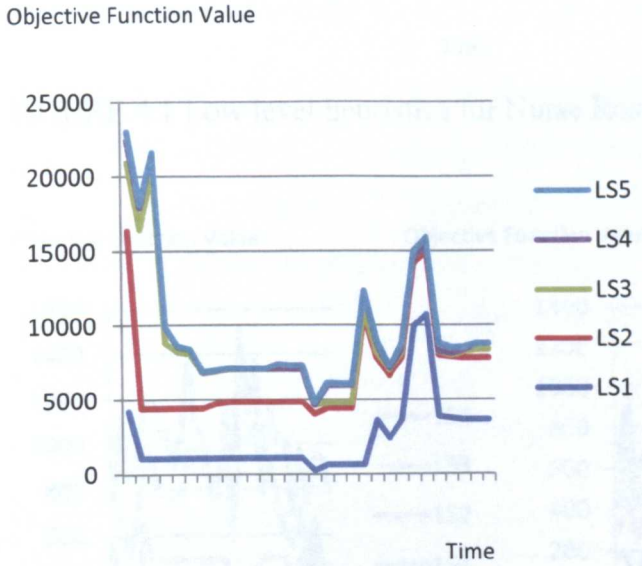
More experiments were done using 3, 4 and 5 initial solutions. Using 3 initial solutions provided marginally better results than using two solutions. The average improvement between the initial and the final objective function value is only of 20.3%. The time taken to obtain three final solutions using 3 random initial solutions is increased by an average of 40% when compared to using only one initial solution. This is not a good trade-off between solution quality and computational time.

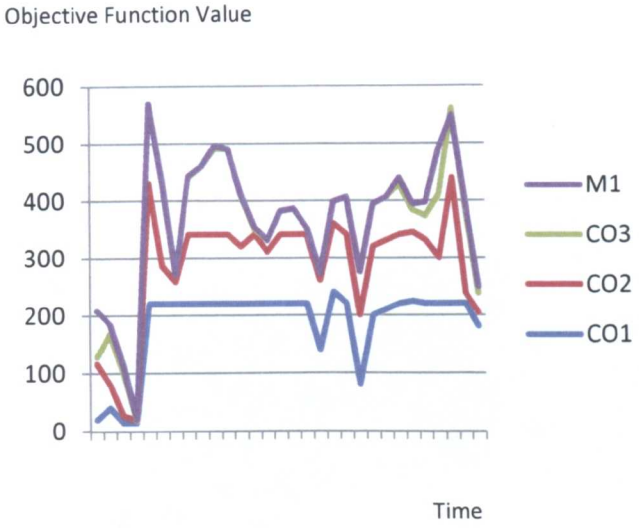
Starting the process with 4 or 5 initial solutions is too long. The time required increments on average by 20% to obtain the best final solution for each new initial solution generated and no significant and sustained improvements are found using more than 3 initial solutions. It is noted that an improvement is found when using 2 initial solutions for the other problem domains. However, the average improvements are not the same for every domain, these depend on the local search heuristics available and the heuristic that creates the initial solution. For example, for the bin packing problem, the increase in improvement is around 15% on average, it is 12% for the Max-SAT problem domain and 16% for the permutation flow shop problems. For the VRP and TSP, the average improvement achieved, when using 2 initial solutions, are respectively 15 and 13%. Using multiple random initial solutions enables the search process to explore different solution neighbourhoods and potentially obtain a better local optimum. Following these results, the hyper-heuristics developed start the search process with two initial random solutions. next chapter.

For the personnel scheduling instances the cross-over heuristics provided the second best improvements. This is the reasoning behind the creation of one hyper-heuristic approach that will be defined in the next subsection (HH1

approach), which will make use of the cross-over heuristics. All other low level heuristics applied iteratively, provide some degree of improvement on the candidate solution. With these results in mind and to evaluate the importance of the acceptance criterion on the solution quality, a second hyper-heuristic approach is created, which will be explained in the next subsection (HH2 approach), which will use all low level heuristics available.

To further illustrate the improvements achieved by the low level heuristics, a graphic is depicted for the personnel scheduling and the flow shop domains. It depicts the instances used as examples in the previous tests. The graphic helps show the improvements of each low level heuristic. As there are 12 low level heuristics for the personnel scheduling problem and 15 low level heuristics for the permutation flow shop problem, to view the behaviour of the low level heuristics, they are divided into three graphics for each domain. Figure 4.1 is for the nurse rostering instance and Figure 4.2 is for the permutation flow shop domain.





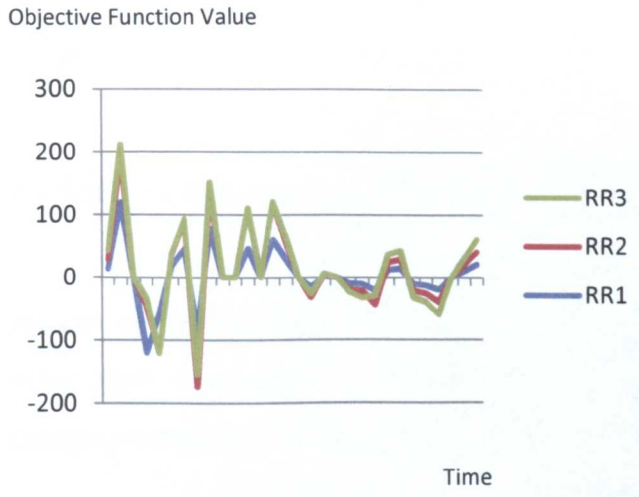
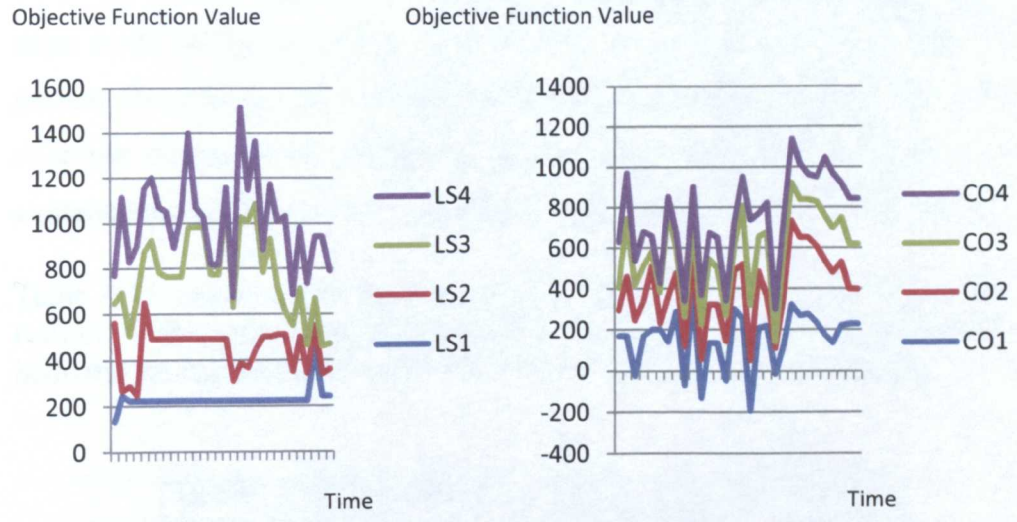


FIGURE 4.1 Low level heuristics for Nurse Rostering instance



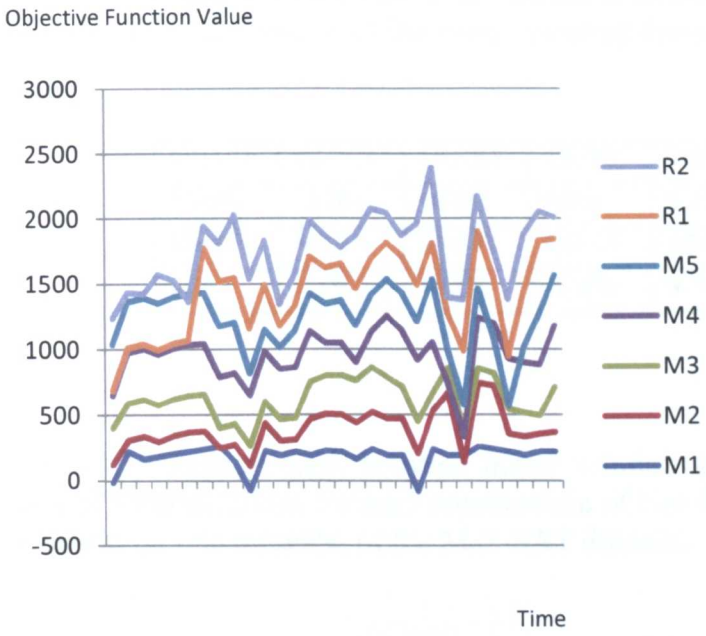


FIGURE 4.2 Low level heuristics for Permutation Flow Shop instance

As can be seen in figures 4.1 and 4.2, the local search heuristics outperform other types of heuristics. It can be noted that each low level heuristic in each category behaves similarly and the improvements follow a similar distribution.

Once the behaviour of each low level heuristic is understood, the next step is to evaluate the different combination of heuristics. The experiments consist of randomly combining two low level heuristic from different categories. The same methodology is used as for the previous experiments. Table 4.11 to 4.14 present the average percentage improvement achieved. This represents an overview of the various experiments done. An exhaustive description of all experiments on all possible combination is omitted due to space limitation.

Table 4.11 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each combination of two different types of low level heuristic for one instance, of the one dimensional bin packing domain.

LS+MU	MU+LS	MU+RR	RR+MU	CO+MU	MU+CO
1.8%	2.2%	1.7%	1.2%	1.9%	2.0%
LS+RR	RR+LS	LS+CO	CO+LS	RR+CO	CO+RR
2.0%	2.9%	4.2%	4.1%	1.7%	1.7%

Table 4.12 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each combination of two different types of low level heuristic for one instance, of the nurse rostering domain.

LS+MU	MU+LS	MU+RR	RR+MU	CO+MU	MU+CO
15.0%	16.0%	10.0%	9.8%	8.4%	9.5%
LS+RR	RR+LS	LS+CO	CO+LS	RR+CO	CO+RR
4.3%	5.1%	10.1%	11.6%	9.7%	9.2%

Table 4.13 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each combination of two different types of low level heuristic for one instance, of the Max-SAT domain.

LS+MU	MU+LS	MU+RR	RR+MU	CO+MU	MU+CO
12.0%	13.6%	1.8%	1.4%	1.2%	2.1%
LS+RR	RR+LS	LS+CO	CO+LS	RR+CO	CO+RR
10.0%	12.1%	10.0%	12.3%	3.1%	2.9%

Table 4.14 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each combination of two different types of low level heuristic for one instance, of the permutation flow shop problem.

LS+MU	MU+LS	MU+RR	RR+MU	CO+MU	MU+CO
7.1%	7.5%	3.8%	4.2%	2.0%	2.2%
LS+RR	RR+LS	LS+CO	CO+LS	RR+CO	CO+RR
7.0%	7.6%	7.2%	7.9%	3.2%	3.1%

The last set of experiments conducted consists of running iteratively all non-local search heuristics for 31 runs over 1000 iterations for every domain. A randomly selected local search algorithm is then applied to the candidate solution found after the 31 runs. A combination of two or more local search was also applied. These experiments demonstrated that using one local search algorithm after using any other local search heuristics will improve the

candidate solution. A combination of more than one local search heuristic provides a larger improvement than using one local search heuristic.

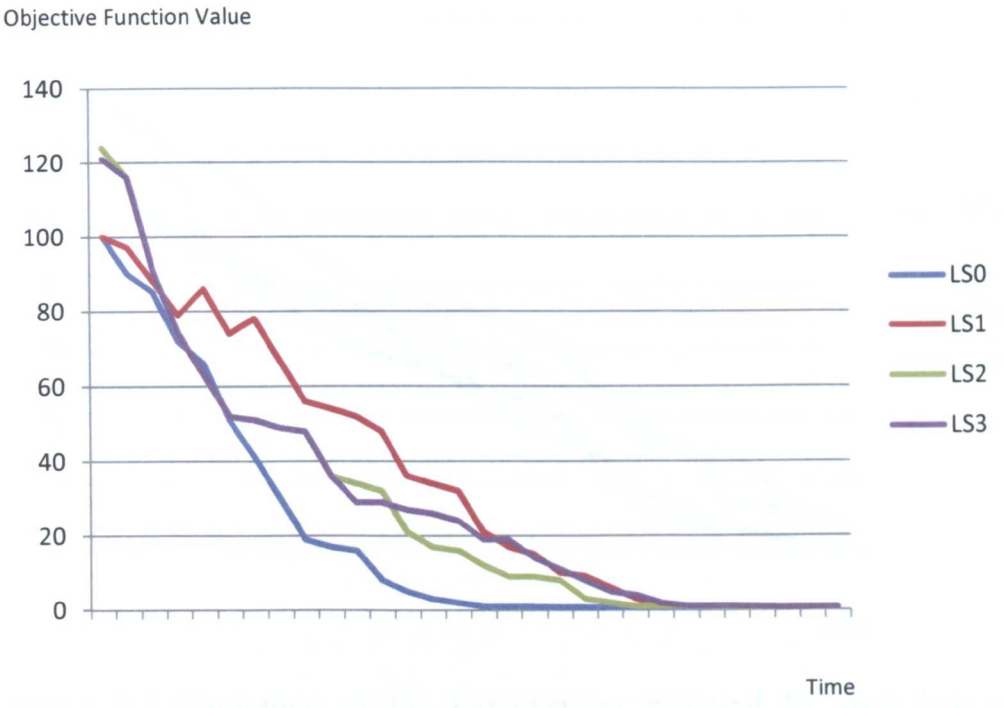
4.2 Initial experiments on the low level heuristics developed for the surgery admission planning problem

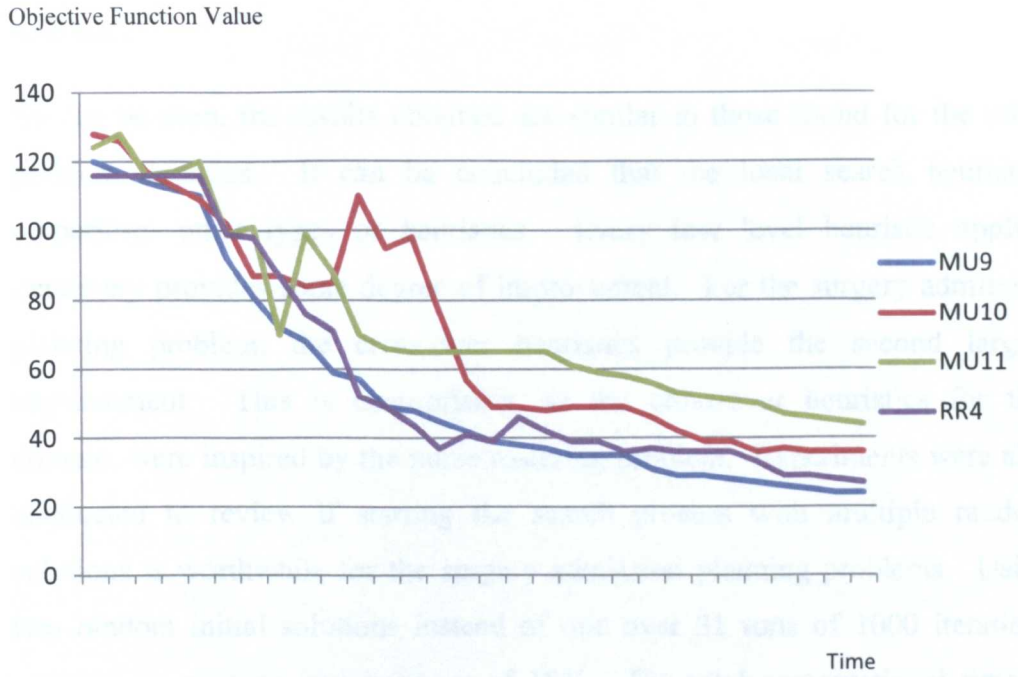
The same three experiments were done using the low level heuristics developed for the surgery admission planning problem. All low level heuristics were applied iteratively to the 10 instances of the surgery admission planning problem described in the previous chapter. Table 4.15 provides the average percentage of improvement for each low level heuristic starting from an initial random solution and with a better candidate solution.

TABLE 4.15 Average percentage improvement for each low level heuristic.

	Surgery
MU9	1,1
MU10	1,2
MU11	1,2
RR4	1,2
RR5	1,2
RR6	2,3
LS0	15,16
LS1	17,18
LS2	15,15
LS3	15,16
CO7	5,6
CO8	5,5

As an example, the improvement achieved by each low level heuristic at each iteration is illustrated in Figure 4.3 for one instance of the problem. Due to the large number of low level heuristics (12), the illustrations will be separated into three graphics (Figure 4.3).


Objective Function Value
140
120
100
80
60
40
20
0
Time
LS0
LS1
LS2
LS3


Objective Function Value
140
120
100
80
60
40
20
0
Time
MU9
MU10
MU11
RR4

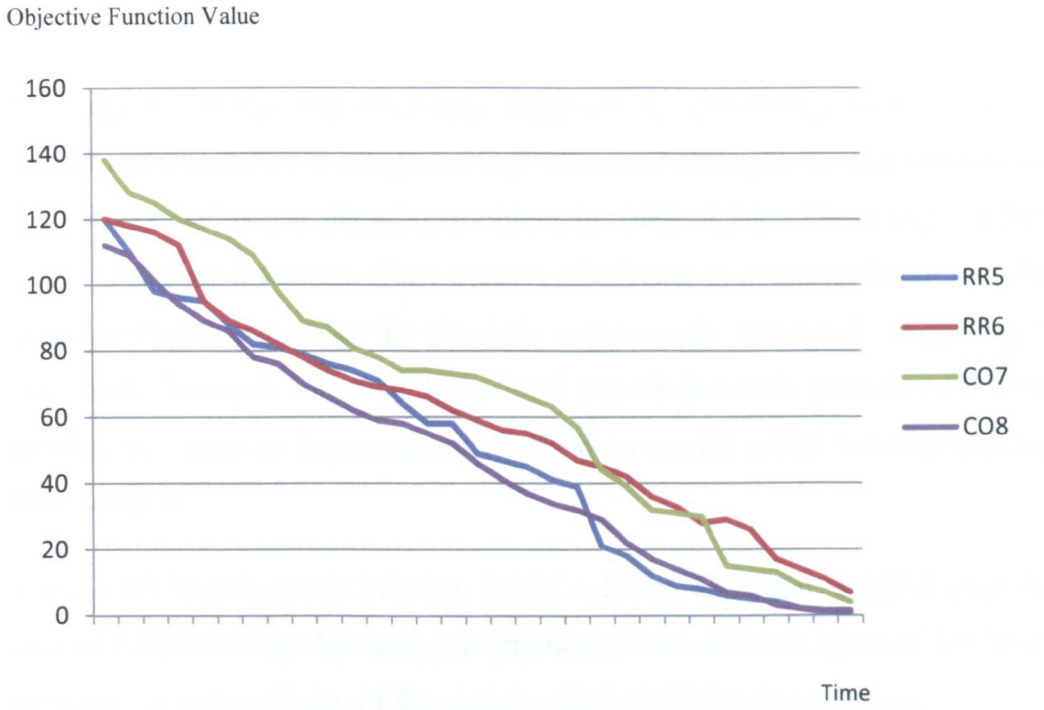


Figure 4.3 Illustration of the improvement achieved by each low level heuristic.

As can be seen, the results obtained are similar to those found for the other problem domains. It can be concluded that the local search heuristics outperform other types of heuristics. Every low level heuristic applied iteratively provides some degree of improvement. For the surgery admission planning problem, the cross-over heuristics provide the second largest improvement. This is unsurprising, as the cross-over heuristics for this domain, were inspired by the nurse rostering problem. Experiments were also conducted to review if starting the search process with multiple random solutions is worthwhile for the surgery admission planning problems. Using two random initial solutions instead of one over 31 runs of 1000 iterations provides an average improvement of 18%. The total computational time is increased by 22% on average. Starting with more than two initial random solutions provides marginally better results but increases the computational time of 22% on average for every new initial random solution used. Starting the algorithm with 3, 4 or 5 random initial solutions provides 19.6% improvement. This trade-off between computational time and improvement is not worthwhile.

A summary of the improvements achieved by combining two low level heuristics from different categories is given, as an example, for one instance of the surgery admission planning problem in Table 4.16. These experiments were run for all instances of the surgery admission planning problem and the same methodology as for the previous experiments was used. Applying a cross-over heuristic followed by a local search heuristic provides the best combination with an improvement of 8% between the initial solution and the final solution.

Table 4.16 Improvement between initial solution and final solution over 31 runs of 1000 iterations for each combination of two different types of low level heuristic for one instance, of the surgery admission planning problem.

LS+MU	MU+LS	MU+RR	RR+MU	CO+MU	MU+CO
5.0%	5.0%	1.0%	1.2%	3.0%	5.0%
LS+RR	RR+LS	LS+CO	CO+LS	RR+CO	CO+RR
3.0%	4.0%	4.0%	8.0%	1.2%	3.4%

As has been done for the other problem domain included in this study, following each set of experiments, the behaviour of each low level heuristic included in the surgery admission planning problem domain will be defined. The ruin and recreate cross-over type heuristics are found to behave either as a mutational heuristic or as a hill-climbing heuristic as was the case for the other problem domain. Table 4.17 provides the category of the behaviour of each cross-over and ruin recreate type heuristics.

TABLE 4.17 Category of cross-over and ruin and recreate type heuristics for surgery admission planning problem

	Mutational	Local Search
Surgery Admission Planning	RR4, RR5, RR6	CO7, CO8

4.3 Remarks

The next chapter will present the hyper-heuristic strategies developed for the thesis. These approaches are inspired by the results obtained when running iteratively each low level heuristic and the improvements provided when combining low level heuristics. A few conclusions can be drawn. It is found that starting from a better initial solution does not provide better final results than starting from a random initial solution; this is the case for all seven problem domains. The local search heuristics for all domains provide the largest improvement on any candidate solution.

Chapter 5

Selection Hyper-heuristics

Four easy-to-implement selection hyper-heuristics are introduced in this chapter, these are based on iterated and greedy search strategies. A crucial feature of the hyper-heuristics developed is that they necessitate less number of parameters when compared to many of the existing approaches. This entails an easier and more efficient implementation, since less time and effort is required for parameter tuning. The empirical results presented in the following chapters show that the most efficient and effective hyper-heuristic which contains only a single parameter outperforms the top ranking algorithms. Section 5.1 introduces the four hyper-heuristics and remarks are included in section 5.2.

5.1 Hyper-heuristic Methodologies

All proposed hyper-heuristics carry different characteristics, although they are two-stage algorithms sharing the same initial stage. They utilise a random perturbation iterative local search, inspired from the Greedy hyper-heuristic which was initially tested by Cowling et al. (2000). This strategy applies all low level search heuristics on a given solution simultaneously and chooses the one which produces the largest improvement. In all approaches, the iterative local search uses only the local search heuristics and ignores the rest of the low level heuristics. The first approach acts similarly to the iterated local search algorithm (Lourenco et al., 2003) and memetic algorithms (Moscato and Norman, 1992) combining multiple perturbation, local search and ruin and recreate operators in this fixed ordering under an iterative search framework. This hyper-heuristic uses random (or headless chicken) crossover (Jones 1995) heuristics as a perturbation operator. The second one uses a Greedy based approach but allows acceptance of worsening solutions. Two main approaches are presented. One strategy consists of evaluating the success of using a pre-selection of low level heuristics. The second approach evaluates the impact of the acceptance criterion on the final solution. For each approach, two hyper-

heuristics are described. These are introduced in an attempt to raise the level of generality of the approaches even further by enabling them to deal with less execution time across a variety of problem domains. The empirical results show that one of the proposed hyper-heuristics outperforms most of the existing hyper-heuristics including the state-of-the-art when evaluated across seven problem domains. The results will be provided in chapter 6, 7 and 8.

As has been previously defined, one of the goals of the thesis is to develop a more general method to solve a variety of problem instances in different domains while having no access to problem specific information. Although for a part of the research included in the thesis, the HyFlex framework is used, the objective was to establish a guideline on the highest performing strategy in any hyper-heuristic environment, where the types, number and efficiency of the various low level heuristics are not known. The strategy developed and applied to the six problem domains included in HyFlex will also be used on the surgery admission planning problem, to further evaluate the strategies developed for this research.

The previous success of local search based hyper-heuristics inspired the design of the hyper-heuristic approaches presented. Intensification, i.e. using local search alone may get the search process stuck at a local optimum. The use of local search low-level heuristics at the beginning of the search process provides high quality initial solutions, so all four hyper-heuristics start with pure hill climbing. Moreover, multiple starting points are sampled in the search space during this process with the assumption that there will still be sufficient remaining time for further improvement on the solution at hand. The previous experiments confirmed this assumption. In both approaches, after the generation of high quality initial solutions, intensification and diversification processes are either explicitly enforced or they are supported implicitly through an extremely flexible move acceptance method. In this section, four hyper-heuristics are presented. The predetermined sequence non-worsening hyper-heuristics (HH1 and HH1adap) and the greedy absolute largest change hyper-heuristics (HH2 and HH2adap) are created to solve problems across different domains.

```
1.    s ← s_i; terminate ← false;
2.    REPEAT DO
3.        Apply all Local-Search-LLH
4.        // Local-Search-LLH is the index set of all low level local
      search heuristics
5.        FOR (∀h ∈ Local-Search-LLH) DO
6.            s' ← Local-Search-LLH_h ( s ); // Apply the local search
      heuristic to s
7.            IF (f(s').isBetterThan[f(s)] ) THEN
8.                s ← s';
9.            ELSE
10.               terminate ← true;
11.           END IF
12.       END FOR
13.   UNTIL ( terminate );
14.   Return s;
```

FIGURE 5.1 Pseudo-code of Greedy-LS(si)

Both original hyper-heuristics start with a common structure (Figures 5.2 and 5.3, lines 1-3) and create two solutions using initialisation algorithm of a given domain at the start. Then this step is followed by the application of an iterative local search as illustrated in Figure 5.1 on each initial solution. The hyper-heuristics employ a Greedy strategy, Cowling et al. (2000), (Figure 5.1, lines 3-8) and apply all low level local search heuristics on the initial solution successively. Consequently, the heuristic yielding the best improvement over the given candidate solution is selected and the corresponding new solution is used in the next iteration. This process is repeated under an iterated local search framework until no further improvements can be achieved. Having explored the neighbourhood of the initial solutions, diversification is needed.

Using the best improved solution from the local search stage, the first hyper-heuristic then applies a randomly selected cross-over from the available cross-over low level heuristics (Figure 5.2, line 6), the new solution is compared to the previous solution and only a non-worsening solution is accepted. As the cross-over heuristics combine two solutions to create a new solution, it is important to note that the best initial solution from the previous step is taken and the second solution is randomly created as an initial solution. As mentioned previously, each problem domain has a different way of creating an

initial solution. For example, in nurse rostering, this initialisation algorithm keeps a solution always feasible and so any solution out of this algorithm is already an improved solution rather than a totally random solution. A randomly selected local search heuristic from the set of low level local searchers is applied to the resulting solution (Figure 5.2, line 7). The new solution is accepted only if a non-worsening solution has been found. In 30% of the cases a ruin recreate heuristic is called (Figure 5.2, lines 13 and 14). This step is applied as an attempt to ensure exploration of a larger search space and avoid getting stuck at a local optimum. Again the new solution is accepted only if it is non-worsening after having applied a randomly selected local search algorithm from our low level heuristic set (line 14). Until a pre-set time limit is reached the steps are repeated from the selection of the crossover heuristic to the application of the last local search. This hyper-heuristic is named a predetermined sequence non-worsening hyper-heuristic and labelled as HH1. The pseudo-code of the HH1 algorithm is provided in Figure 5.2.

The hyper-heuristics were first tested on the nurse rostering domain. The extended benchmark includes the low level heuristics, 43 real-world problems and the best known solution for each problem. HH1 when applied to the nurse rostering domain, applies a ruin and recreate type heuristic when the candidate solution is 50% worse than the best known solution.

```
1.    s'1 ← Greedy-LS(s1);
2.    s'2 ← Greedy-LS(s2);
3.    s' ← bestOf(s'1, s'2); // Select the solution with the best quality
4.    WHILE remaining time limit is not reached DO
5.          Create solution s;
6.          Apply randomly selected cross-over to s and s': output s";
7.          Apply randomly selected local search to s";
8.          IF (f(s").isBetterThan[f(s') ] ) THEN
9.                s' ← s";
10.         END IF
11.         Create a copy of s' called scopy;
12.         Apply randomly selected ruin and recreate heuristic
                  with probability of 0.3 on scopy;
13.         Apply randomly selected local search to scopy;
14.         IF (f(scopy).isBetterThan[f(s') ] ) THEN
15.               s' ← scopy;
16.         END IF
17.   END WHILE
```

FIGURE 5.2 Pseudo-code of the predetermined sequence non-worsening hyper-heuristic (HH1) operating on a problem domain using an objective function $f(.)$

The second hyper-heuristic also initially creates two solutions. Then similar to HH1, the iterative local search process is applied on both initial solutions. The second hyper-heuristic also uses the best improved initial solution and applies all the low level heuristics to the candidate solution in the next stage. HH2 keeps all new solutions produced by each low level heuristic at each iteration. The low level heuristic generating the largest absolute change in the quality of a solution (objective value) is selected implying that a worsening solution is allowed. This last step continues until a pre-set time limit is reached. This hyper-heuristic is named as greedy absolute largest change hyper-heuristic and labelled as HH2. The pseudo code is shown in Figure 5.3.

HH1 and HH2 were successful in providing good results for the nurse rostering and the surgery admission planning problems. However, when these were tested on instances in the other five problem domains, with a time limit of 10 minutes, the results were not found to be as good as the best algorithms submitted to the CHeSC competition. This is mainly due to the fact that using local search heuristics at the first stage of the algorithm is time consuming.

The local search heuristics could be more efficient during the first stage of the hyper-heuristics. For a given problem and approach, if the basins of attraction are close to each other, which seems to be the case in nurse rostering, a large perturbation is not necessary at the beginning of the search process, but if the basins of attraction are far from each other or the search landscape contains a lot of plateaus, a larger perturbation is necessary to improve the search process and ensure that the algorithm does not get stuck at a local optimum. It is therefore proposed to modify the first stage of both hyper-heuristics creating HH1adap and HH2adap from HH1 and HH2, respectively. The modified versions of the hyper-heuristics terminate in the first or second stage if the time limit is exceeded. A learning process is introduced in the modified hyper-heuristics to manage the trade-off between diversification and intensification. After a small number of initial iterations, denoted as *procIter*, the current candidate solution is compared to the previous solution. This check is done right at the start of the stage to make a decision regarding how to proceed. If the improvement of the current solution in the objective value over the previous solution is worse than an expected value, which is a δ factor of the previous solution, a large perturbation using a randomly chosen mutational heuristic is made and the number of iterations for which the iterative local search continues is reduced. Otherwise, the iterative local search proceeds without any interference. A similar method has been applied to graph colouring problems by Loudni (2012).

```
1.    s'1 ← Greedy-LS(s1);
2.    s'2 ← Greedy-LS(s2);
3.    s' ← bestOf(s'1, s'2); // Select the best solution with respect to the
   objective values
4.    WHILE remaining time limit is not reached DO
5.          Initialise max-heuristic-index pointing to the first heuristic,
   Max-diff ← -1;
6.          FOR (∀h ∈ LLH) DO // LLH is the index set of all low level
   heuristics
7.                Apply low level heuristic h to s and save the resultant
   solution in s'[h]
8.                diff[h] ← |f(s'[h]) − f(s)|;
9.                IF ( diff(h) > Max-diff) THEN
10.                     Max-diff ← diff[h];
11.                     max-heuristic-index ← h;
12.               END IF
13.         END FOR
14.   s ← s'[max-heuristic-index]; // Accept heuristic h creating the largest
   diff
15.   END WHILE
```

FIGURE 5.3 Pseudo-code of the greedy absolute largest improvement hyper-heuristic (HH2) operating on a problem domain using an objective function $f(.)$

A set of initial parameter tuning experiments are performed to decide from one of the values for *procIter* = {4, 5, 6, 7} and δ factor = {0.01, 0.02, 0.03}. The results show that 4 and 0.01 are good choices for *procIter* and δ, respectively for the instances of the problem domains explored in this study. So, if after 4 iterations the candidate solution cannot be improved by at least 0.01 of the previous solution, a large perturbation is made. The proposed modification in both hyper-heuristics improves the efficiency of the local search stage. This has brought an improvement on the results for all problem domains; these will be reviewed in the next chapters.

5.2 Remarks

In this chapter two hyper-heuristic approaches are defined. The first approach privileges a pre-selection of low level heuristics that performed well in preliminary tests (HH1 and HH1adap). The second approach uses a more greedy strategy by applying all available low level heuristics (HH2 and HH2adap). In order to establish the best approach, the results will be

presented and analysed for each problem domain in the following chapters. Chapter 6 includes the results for all 43 instances of the nurse rostering problem. Chapter 7 provides the results for the Max-SAT, the permutation flow shop, the one dimensional bin packing, the travelling salesman and the vehicle routing problems. Chapter 8 includes the results found for the surgery admission planning problem.

Chapter 6

Selection Hyper-heuristics for Nurse Rostering

In this chapter, the results of the four hyper-heuristics that have been developed and presented in Chapter 5 are applied to 43 real world benchmark instances of the nurse rostering problem. All four hyper-heuristics will be compared and evaluated. The hyper-heuristics will also be evaluated against other algorithms; hyper-heuristics, exact methods or heuristics. The results will be discussed. As mentioned previously both hyper-heuristic approaches are different. Each strategy will be reviewed to see which approach is the best in the overall and/or in which cases an approach is better than the other one depending on the problem structure.

It is important to note that the method proposed does not require any fine tuning or adapting to the different problems being solved. When compared with other algorithms the results obtained by both hyper-heuristics are very competitive and in some cases the best known results are obtained.

6.1 Design Variation of Proposed Hyper-heuristics for Nurse Rostering

All four hyper-heuristic approaches are defined in Chapter 5. These approaches are used consistently to evaluate all seven problem domains investigated in this research. However, further explanations are required on the design process of the hyper-heuristics. As mentioned previously, the main objective of this research is to develop a more general algorithm to solve healthcare problems, specifically nurse rostering. When designing the first hyper-heuristic approach for nurse rostering i.e. HH1 and HH1adap, applying a ruin and recreate heuristic is only done when the current candidate solution evaluated is 50% worse than the best known solution (Figure 5.2 line 12 in chapter 5), this will be referred to as the comparison factor. A series of experiments were conducted on all 43 instances of the nurse rostering problem using different comparison factors ranging from 0.1 to 0.9. The experiments

were performed 31 times on all instances. The comparison factor that provides the best final objective function value is 0.5. Tables 6.1 and 6.2 provide the average and the best objective function values found for 5 nurse rostering problems. Figure 6.1 illustrates the average objective function value for one instance of the nurse rostering problem for the factors ranging from 0.1 to 0.9 (0.1, 0.11,0.12,....,0.99).

TABLE 6.1 Average objective function value found for each comparison factor for 5 nurse rostering instances. The best known solution is provided in the column headed BKN.

	Factors									
Instances	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	BKN
BCV5.4.1	66	66	66	66	48	48	52	52	52	48
ORTEC01	520	480	480	480	380	390	390	390	410	270
QMC-1	44	38	36	24	20	24	24	25	25	14
SINTEF	38	38	28	24	10	18	18	18	22	0
ERRVH-B	3952	3840	3678	3542	3428	3548	3670	3780	3780	3121

TABLE 6.2 Best objective function value found for each comparison factor for 5 nurse rostering instances. The best known solution is provided in the column headed BKN.

	Factors									
Instances	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	BKN
BCV5.4.1	66	62	60	60	48	48	52	52	52	48
ORTEC01	500	470	470	470	380	390	390	390	410	270
QMC-1	42	37	36	24	20	24	24	24	25	14
SINTEF	36	36	24	18	8	12	12	16	20	0
ERRVH-B	3882	3640	3578	3502	3402	3458	3570	3570	3600	3121

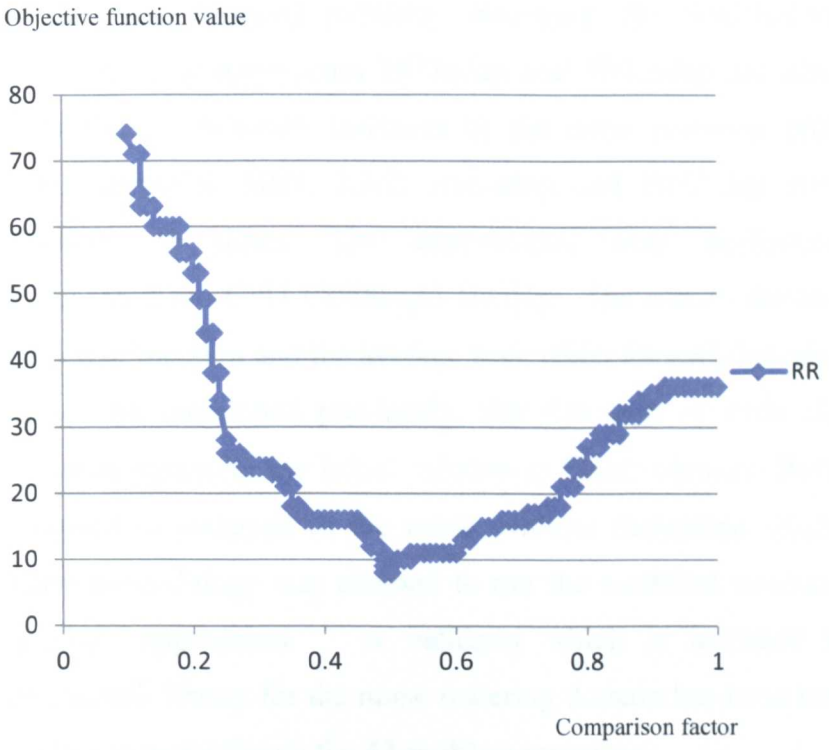


Figure 6.1 Average objective function value for each comparison factor for the SINTEF nurse rostering instance.

6.2 Evaluation Method and Experimental Methodology

The goal of this research is to provide a more general method to solve different nurse rostering problems without modifying the algorithms when solving each instance. The question of the method of the evaluation of the hyper-heuristics arises. The aim is to obtain a good solution to each instance without adapting the hyper-heuristics. It is important to note that the goal is not to compete with algorithms designed specifically for the problem at hand, therefore the objective is not to obtain the best known or optimal solution. The objective is to obtain a good solution to the problems in an acceptable computational time. Other hyper-heuristics or other meta-heuristics have been used to solve some of these benchmark instances but not one has been applied to all instances. It is for this reason that the results are compared with the best known or optimal solution for each instance.

Firstly, HH1 and HH2 were used to solve the forty three benchmark instances of the nurse rostering problem. Secondly, the modified versions of these hyper-heuristic approaches HH1adap and HH2adap are also applied to the forty three benchmark instances of the nurse rostering problem. All four hyper-heuristics, HH1, HH2, HH1adap and HH2adap were run on each problem 31 times. The experiments were performed on Intel(R) Core(TM)2Duo CPU E8500 @3.16GHz. The results shown are the average objective function and the average time taken for each instance over thirty one runs. As mentioned previously, the first part of both algorithms uses a common structure, the initial solution is found when no further shifts can be assigned or swapped in the roster without increasing solution quality. The same methodology was adopted to run the modified versions of both hyper-heuristic approaches. A validator which is included in the extended benchmark library for the nurse rostering domain has been used when running the hyper-heuristics on the 43 problem instances.

6.3 Average performance comparison of hyper-heuritics

The experimental results for HH1 and HH2 along with their adaptive versions are summarised in Tables 6.3 and 6.4, respectively. The average objective function value ("avr.") and time ("avr. t.") over 31 runs is provided for each hyper-heuristic for a given instance. The column labelled as "st. dev." is the standard deviation from the average objective value. All times are reported in seconds unless otherwise *m* is used to denote minutes. As a statistical test, a student's t-test is performed between a pair of hyper-heuristics assessed based on the results from the 31 runs for each benchmark instance. The following notation is used under the column of "vs." in Tables 6.3 and 6.4: A > B indicates that the algorithm A performs better than the algorithm B and this is statistically significant at a 95% confidence level while < denotes vice versa. A ≥ (≤ or ~) B indicates that the algorithm A performs slightly better (worse or no different) than the algorithm B.

Tables 6.3 and 6.4 show that each any-time hyper-heuristic terminates in less than 371 seconds on average, excluding the MER-A instance. The average overall duration used during the search process by an adaptive hyper-heuristic

is always either the same or less than its non-adaptive version for a given instance, yet the adaptive hyper-heuristics perform better than their non-adaptive variants on average in the overall. For example, the adaptive versions of the hyper-heuristics perform well on MER-A terminating after 10 minutes on average, while HH1 generates the worst average running time of 33 minutes for this instance and the average solution quality is still not as good as HH1adap and HH2adap achieves.

From Table 6.3, it has been observed that HH1adap is statistically significantly better than HH1 for 7 instances out of the 43 benchmark problems: BCV6.13.1, ORTEC01, ORTEC02, QMC-2, GPost-B, WHPP, and QMC-A. There is no instance for vice versa. HH1adap provides slightly better results on 18 instances, while HH1 performs slightly better in only 5 cases. Both HH1 and HH1adap obtain the optimal solutions in all runs on the 7 benchmark instances of BCV5.4.1, BCV.8.13.1, BCV8.13.2, Millar2s1, Millar2s1.1, Musa and Ozkarahan. There is no average performance difference between both algorithms for the remaining 5 instances. In the overall, HH1adap is better than HH1.

TABLE 6.3 Average performance comparison of HH1 and HH1adap, where a bold entry indicates that an algorithm obtained the optimal result in all runs on the relevant instance

	HH1			vs.	HH1adap		
Instance	**avr.**	**st.dev.**	**avr. t.**		**avr.**	**st. dev.**	**avr. t.**
BCV1.8.1	256.2	3.7	32	≥	258.0	3.2	32
BCV1.8.2	866.4	6.9	35	≤	864.7	7.3	35
BCV1.8.3	235.5	4.9	34	~	235.5	3.2	34
BCV1.8.4	249.0	7.8	32	≥	249.0	5.2	32
BCV2.46.1	1542.8	58.0	38	~	1542.8	39.0	38
BCV3.46.1	3302.0	101.8	54	≤	3298.3	91.4	54
BCV3.46.2	895.8	1.5	55	≤	895.6	1.1	55
BCV4.13.1	10.3	0.5	27	~	10.3	0.5	27
BCV4.13.2	10.3	0.4	27	≤	10.2	0.3	27
BCV5.4.1	**48.0**	**0.0**	**20**	~	**48.0**	**0.0**	**20**
BCV6.13.1	790.1	9.9	55	<	770.1	6.4	55
BCV6.13.2	392.0	0.0	20	~	392.0	6.3	20
BCV7.10.1	387.3	9.3	24	≥	389.0	9.6	24
BCV.8.13.1	**148.0**	**0.0**	**23**	~	**148.0**	**0.0**	**23**
BCV8.13.2	**148.0**	**0.0**	**34**	~	**148.0**	**0.0**	**34**
BCVA12.1	2047.7	385.9	1m57	≤	1897.7	402.3	1m57
BCVA12.2	2529.9	214.0	47	≤	2519.9	204.0	47
ORTEC01	429.2	32.0	44	<	390.0	16.0	44
ORTEC02	484.2	51.0	55	<	434.0	47.0	55
GPost	20.3	54.7	26	≤	18.0	48.6	26
GPost-B	12.2	3.6	29	<	10.0	4.2	29
QMC-1	27.1	3.1	60	≤	26.0	4.3	60
QMC-2	32.4	1.5	38	<	31.7	1.2	38
Ikegami2d1	6.6	2.2	1m12	≤	6.6	2.2	1m12
Ikegami3d1	29.5	4.1	1m3	≤	29.3	4.0	1m3
Ikegami3d1.1	33.3	4.2	60	≤	33.3	3.8	60
Ikegami3d1.2	34.5	5.8	1m27	≤	33.2	5.2	1m27
Millar2s1	**0.0**	**0.0**	**7**	~	**0.0**	**0.0**	**7**
Millar2s1.1	**0.0**	**0.0**	**5**	~	**0.0**	**0.0**	**5**
Valouxis	150.6	33.0	27	≤	140.9	36.0	27
WHPP	2070.9	62.7	38	<	2000.0	60.0	38
LLR	301.4	0.6	22	≤	301.2	0.5	22
Musa	**175.0**	**0.0**	**22**	~	**175.0**	**0.0**	**22**
Ozkarahan	**0.0**	**0.0**	**6**	~	**0.0**	**0.0**	**6**
Azaiez	0.9	0.8	27	≤	0.9	0.7	27
SINTEF	8.5	2.6	32	≤	8.2	2.4	32
CHILD-A2	1360.0	101.8	2m42	≥	1364.0	98.0	2m42
ERMGH-A	797.4	1.8	52	≥	798.0	1.7	52
ERMGH-B	1398.0	21.3	60	~	1398.0	23.5	60
ERRVH-A	2219.0	64.0	5m	~	2219.0	66.0	5m
ERRVH-B	3424.3	95.0	4m	≤	3404.5	93.0	4m
MER-A	9839.4	129.0	33m	≤	9836.0	118.0	10m
QMC-A	27.8	1.3	27	<	27.1	1.4	27

Table 6.4 shows that there are 13 instances on which HH2adap performs statistically significantly better than HH2: BCV1.8.2, BCV1.8.3, BCV3.46.2, ORTEC02, QMC-1, Ikegami3d1.1, Ikegami3d1.2, Valouxis, WHPP, LLR, SINTEF, CHILD-A2, and QMC-A. Additionally, HH2adap performs slightly better than HH2 on 13 instances, while vice versa is observed on GPost-B only. Both HH2 and HH2adap obtain the optimal solutions in all runs on 8 benchmark instances, one of them being Ikegami3d1 and the rest of the being the same instances that HH1 and HH1adap was successful in obtaining the optimal solutions. On the 8 instances, both HH2 and HH2adap perform similarly. In the overall, HH2adap is better than HH2.

TABLE 6.4 Average performance comparison of HH2 and HH2adap and HH1adap, where a bold entry indicates that an algorithm obtained the optimal result in all runs on the relevant instance

	HH2			vs.	HH2adap			HH2adap vs.
Instance	**avr.**	**st.dev.**	**avr. t.**		**avr.**	**st. dev.**	**avr. t.**	**HH1adap**
BCV1.8.1	254.8	2.8	30	≤	252.0	3.4	30	>
BCV1.8.2	867.5	7.0	30	<	853.0	7.0	30	>
BCV1.8.3	234.8	3.5	57	<	232.0	3.5	57	≥
BCV1.8.4	248.6	7.3	40	≤	248.0	7.5	40	≥
BCV2.46.1	1592.0	19.2	44	~	1592.0	19.2	44	<
BCV3.46.1	3385.0	25.0	44	≤	3380.0	23.0	44	<
BCV3.46.2	896.4	2.0	60	<	894.0	2.0	60	>
BCV4.13.1	10.2	0.4	35	~	10.2	0.5	35	≥
BCV4.13.2	10.2	0.5	30	~	10.2	0.7	30	≤
BCV5.4.1	**48.0**	**0.0**	**20**	~	**48.0**	**0.0**	**20**	~
BCV6.13.1	784.3	4.6	46	~	784.3	8.6	46	<
BCV6.13.2	392.0	0.0	46	~	392.0	5.0	46	~
BCV7.10.1	386.8	7.0	31	≤	386.0	8.0	31	≥
BCV.8.13.1	**148.0**	**0.0**	**33**	~	**148.0**	**0.0**	**33**	~
BCV8.13.2	**148.0**	**0.0**	**32**	~	**148.0**	**0.0**	**32**	~
BCVA12.1	2062.0	279.0	55	≤	1975.0	269.0	55	≤
BCVA12.2	2615.3	255.7	1m55	≤	2529.0	246.4	1m55	≤
ORTEC01	407.7	34.7	42	≤	397.0	24.7	42	≤
ORTEC02	472.6	54.9	67	<	442.0	56.8	67	≤
GPost	12.0	2.0	30	~	12.0	7.6	30	≥
GPost-B	12.6	2.3	34	≥	13.0	3.2	34	<
QMC-1	26.1	3.5	50	<	24.0	3.0	50	>
QMC-2	32.4	1.5	43	≤	32.0	1.4	43	≤
Ikegami2d1	2.0	0.0	1m4	~	2.0	0.0	1m4	>
Ikegami3d1	**2.0**	**0.0**	**55**	~	**2.0**	**0.0**	**55**	>
Ikegami3d1.1	34.1	3.9	1m29	<	30.0	10.1	1m29	≥
Ikegami3d1.2	34.6	4.4	1m12	<	31.6	7.2	1m12	≥
Millar2s1	**0.0**	**0.0**	**8**	~	**0.0**	**0.0**	**8**	~
Millar2s1.1	**0.0**	**0.0**	**4**	~	**0.0**	**0.0**	**4**	~
Valouxis	197.1	39.2	34	<	177.0	28.3	34	<
WHPP	2002.0	1.4	37	<	1990.0	14.2	37	≥
LLR	301.3	0.6	35	<	301.0	0.6	35	≥
Musa	**175.0**	**0.0**	**25**	~	**175.0**	**0.0**	**25**	~
Ozkarahan	**0.0**	**0.0**	**3**	~	**0.0**	**0.0**	**3**	~
Azaiez	0.8	0.8	29	≤	0.6	0.2	29	>
SINTEF	1.9	0.7	47	<	1.2	0.7	47	>
CHILD-A2	1184.0	161.0	3m16	<	1100.0	156.0	3m16	>
ERMGH-A	724.8	74.0	1m25	≤	714.1	73.6	1m25	>
ERMGH-B	1354.1	24.0	43	≤	1354.1	34.6	43	>
ERRVH-A	2196.6	67.6	5m47	≤	2196.0	67.6	5m47	≥
ERRVH-B	3347.4	80.9	6m11	≤	3317.0	78.2	6m11	>
MER-A	9643.0	126.7	30m	~	9643.0	106.7	10m	>
QMC-A	28.0	1.1	34	<	27.0	1.1	34	≥

Both hyper-heuristic strategies HH1adap and HH2adap provide good results. In some cases, the HH1adap approach is better than the HH2adap version and vice versa. Based on the results in the last column of Table 6.4, it is observed that HH2adap is statistically significantly better than HH1adap on 13 instances: BCV1.8.1, BCV1.8.2, BCV3.46.2, QMC-1, Ikegami2d1, Ikegami3d1, Azaeiz, SINTEF, CHILD-A2, ERMGH-A, ERMGH-B, ERRVH-B, MER-A. HH1adap is statistically significantly better than HH2adap on 5 instances: BCV2.46.1, BCV3.46.1, BCV6.13.1, GPost-B, Valouxis. HH2adap provide slightly better results than HH1adap for 11 instances. HH1adap delivers a slightly better performance when compared to HH2adap on 6 instances. For 8 instances, there is no average performance difference between both methods. The times taken to obtain the results are also quite similar. It appears that HH2adap is better all-around to solve the nurse rostering problems. HH2adap is successful in solving highly constrained problems such as the Ikegami instances and MER-A and delivers a better performance than HH1adap. However, considering the overall performance of HH1adap, it has been observed that it still provides good results which are comparable to the solutions obtained by problem specific algorithms.

The proposed hyper-heuristics have two successive stages: local search (Greedy-LS: lines 1-3 of the pseudo-codes for HH1 and HH2) and the *rest*. Each stage operates as an independent yet successive any-time algorithm. Another factor that can be looked into is the average percentage improvement in the quality of solutions after each stage for each hyper-heuristic along with the time spent during a stage as provided in Table 6.5. The results show that the second stage of a hyper-heuristic is more time consuming than the first stage for all problems.

The second stage of the algorithm takes between 61% to 64% of the overall execution time on average across all instances. Although the second stage takes more time, the percentage of improvement obtained at the end of the second stage is in all cases smaller when compared to the percentage improvement obtained at the end of the first stage. The first stage of the algorithms improves the initial solution from 41 to 43%, whereas the second stage yields an improvement of 23, 31 and 32%. These percentage of

improvements and time spend by stage do not enable to establish which approach is best on nurse rostering problems.

TABLE 6.5 Average percentage of improvement at each stage of the hyper-heuristics and average percentage of time taken at each stage by the hyper-heuristics across each problem domain.

	%-improv		%-time	
	Greedy-LS	Rest	Greedy-LS	Rest
HH1	41%	31%	39%	61%
HH1adap	43%	32%	37%	63%
HH2	41%	23%	39%	61%
HH2adap	42%	23%	36%	64%

For all instances, both hyper-heuristics provide good results, either the optimal or best known result, better than the best known result or close to these. It is important to note that for instance, WHPP, both hyper-heuristics provide acceptable results as the modelling of this instance includes weights of 1000 or 1 for each constraint.

6.4 Best performance comparison of hyper-heuristics

In this section, a comparison of the best of run results from the algorithms to the best known solutions is done for each benchmark instance and a discussion for each instance is provided. The results are summarised in Table 6.6. The adaptive and non-adaptive versions of each hyper-heuristic strategy provide generally the same results. A few exceptions are noted for 12 instances, where better objective function values are obtained by both HH1adap and HH2adap. HH1adap outperforms HH1 on 3 instances. HH1 delivers a better solution than HH1adap for one instance. When compared to the previous best known solution, *BKN*, HH1adap finds the *best* solution for 11 instances and HH2adap finds the *best* solution for 13 instances. HH1adap obtains the new best known solution or the optimal solution 20 times over all instances. HH2adap provides the new best known solution or the optimal solutions 27 times.

TABLE 6.6 Comparison of hyper-heuristics to the previously proposed approaches based on best performance. The entries marked in bold are the optimal results obtained by the associated algorithm. The italic entries are the results which are better than best known solution (BKN), where an underline indicates the new best for the given instance obtained by the associated algorithm. The success count (s.c.) denoting the number of instances for which the associated algorithm improved on BKN or obtained the optimal result and the count of new best (c.n.b.) results obtained by an algorithm are also provided as a summary.

Instance	HH1	HH2	HH1adap	HH2adap	BKN
BCV1.8.1	*220*	*210*	*220*	*210*	252
BCV1.8.2	*830*	*820*	*830*	*820*	853
BCV1.8.3	*200*	*220*	*200*	*220*	232
BCV1.8.4	*230*	*236*	*230*	*236*	291
BCV2.46.1	*1526*	*1560*	*1526*	*1560*	1572
BCV3.46.1	3290	3355	3290	3355	3280
BCV3.46.2	**894**	**894**	**894**	**894**	**894**
BCV4.13.1	**10**	**10**	**10**	**10**	**10**
BCV4.13.2	**10**	**10**	**10**	**10**	**10**
BCV5.4.1	**48**	**48**	**48**	**48**	**48**
BCV6.13.1	*760*	*750*	*760*	*750*	768
BCV6.13.2	*382*	*382*	*382*	*382*	392
BCV7.10.1	**381**	**381**	**381**	**381**	**381**
BCV.8.13.1	**148**	**148**	**148**	**148**	**148**
BCV8.13.2	**148**	**148**	**148**	**148**	**148**
BCVA12.1	1997	1965	1497	1965	1294
BCVA12.2	1953	1975	*1853*	1953	1953
ORTEC01	380	380	380	380	**270**
ORTEC02	390	400	390	400	**270**
GPost	12	12	12	12	**5**
GPost-B	8	8	8	8	**3**
QMC-1	20	18	18	18	14
QMC-2	**29**	**29**	**29**	**29**	**29**
Ikegami2d1	4	2	4	**0**	**0**
Ikegami3d1	25	**2**	25	**2**	**2**
Ikegami3d1.1	27	13	24	11	**3**
Ikegami3d1.2	22	12	22	10	**3**
Millar2s1	**0**	**0**	**0**	**0**	**0**
Millar2s1.1	**0**	**0**	**0**	**0**	**0**
Valouxis	120	160	100	160	**20**
WHPP	1900	1870	1800	1870	**5**
LLR	**301**	**301**	**301**	**301**	**301**
Musa	**175**	**175**	**175**	**175**	**175**
Ozkarahan	**0**	**0**	**0**	**0**	**0**
Azaiez	**0**	**0**	**0**	**0**	**0**
SINTEF	5	**0**	5	**0**	**0**
CHILD-A2	1110	*990*	1116	*990*	1095
ERMGH-A	*745*	*700*	*745*	*700*	795
ERMGH-B	*1300*	*1249*	*1300*	*1249*	1459
ERRVH-A	2179	2146	*2069*	*2116*	2142
ERRVH-B	3325	3325	3225	*3117*	3121
MER-A	9760	9560	9760	9017	9017
QMC-A	*24*	*25*	*24*	*25*	27
s.c./c.n.b	24/5	27/7	26/6	30/9	

The following observations can be made: all four hyper-heuristics obtain a new best objective value for all instances of the BCV problems, ERMGH-A, ERMGH-B, QMC-A except for instances BCV3.46.1 and BCVA12.1, however HH1adap finds the new best solution for BCVA12.2. The optimal results are found by all four hyper-heuristics for 14 instances. HH2adap finds the optimal solution for the highly constrained Ikegami2d1 and Ikegami3d1 problems. HH2adap obtains the best new solution for BCV1.8.1, BCV1.8.2, BCV6.13.1, BCV6.13.2, CHILD-A2, ERMGH-A, ERMGH-B, ERRVH-A, ERRVH-B. HH1adap also provides the new best solution for BCV1.8.3, BCV1.8.4, BCV2.46.1, BCV6.13.2 and QMC-A.

The comparisons made between the average results and the best results of the adaptive and non-adaptive versions of the hyper-heuristics and presented in Tables 6.3, 6.4 and 6.6 respectively establish that the adaptive versions of the hyper-heuristics outperform the non-adaptive version of the problem. Furthermore, HH2adap obtains best known solutions and delivers a better average performance on more instances than HH1adap.

Figure 6.2 provides an illustration of how the quality of the candidate solution in hand changes in time during the execution of the first and second stages of HH1adap and HH2adap for a given sample instance based on a sample run when an algorithm obtains the best solution. The Ikegami 2 shifts (Ikegami2d1), Ikegami 3 shifts (Ikegami3d1) and the MER-A instances are chosen as examples for which both algorithms deliver a better performance than previously known algorithms. The first stage of the algorithm is the same for HH1adap and HH2adap. HH2adap outperforms HH1adap in the second stage of the algorithm.

FIGURE 6.2 The propagation of the objective value of the current solution with respect to time for Ikegami2d1 during the (a) first and (b) second stages, Ikegami3d1 (c) first and (d) second stages and MER-A (e) first and (f) second stages.

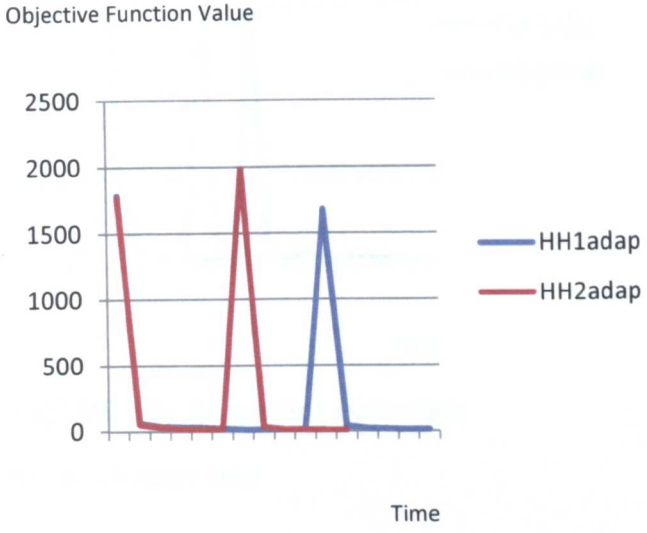


FIGURE 6.2a Ikegami2d1 First stage

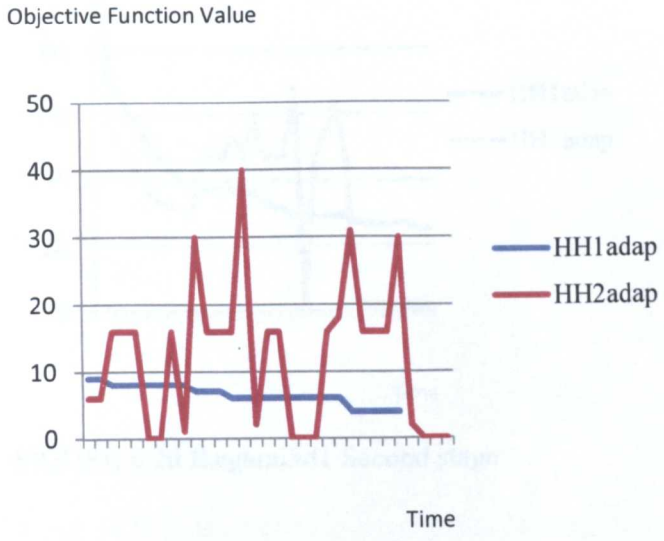


FIGURE 6.2b Ikegami2d1 Second stage

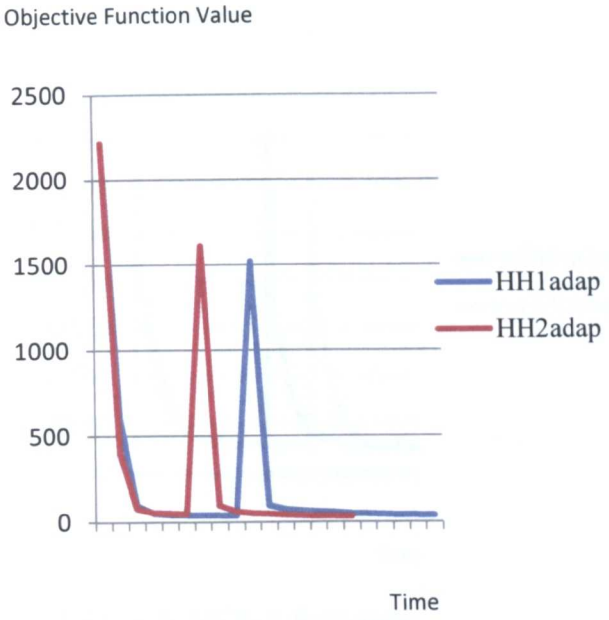


FIGURE 6.2c Ikegami3d1 First stage

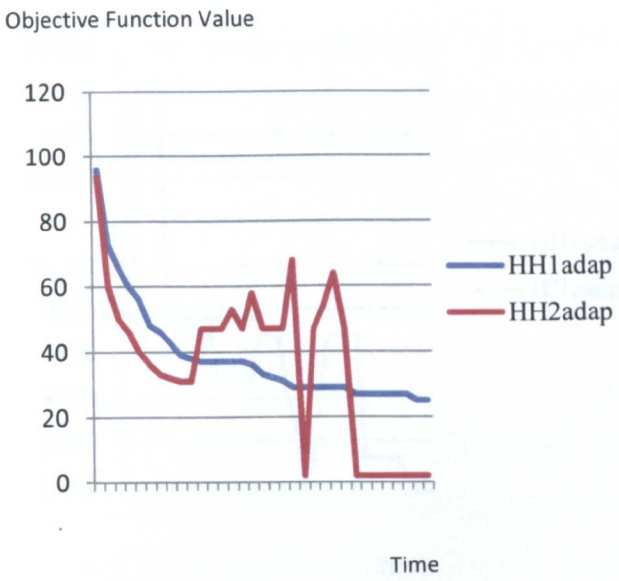


FIGURE 6.2d Ikegami3d1 Second stage

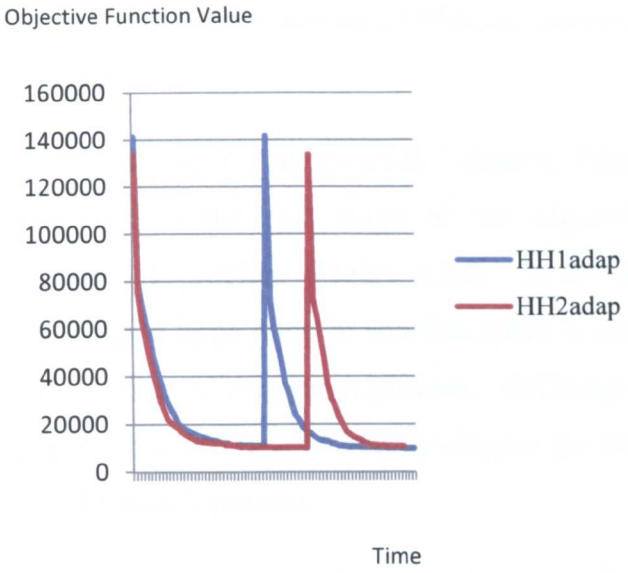


FIGURE 6.2e MER-A First stage

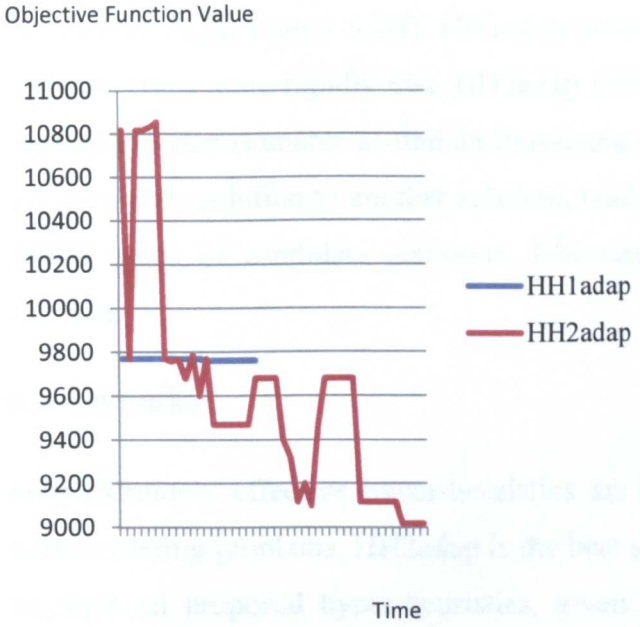


FIGURE 6.2f MER-A second stage

Figure 6.2(a) shows that both methods start the search process with an initial random solution of similar quality, in this case, 1786 for HH1adap and 1772 for HH2adap for the Ikegami2d1 instance. The final candidate solutions from the two initial solutions are similar, for HH1adap, these values are 9 and 9, for HH2adap, they are 9 and 6. The first stage is the stage that provides the most improvement. Illustrated in Figure 6.2(b) HH2adap starts with a better initial solution in the example provided (i.e. 6) and by accepting worse solutions allows an exploration of a wider search space of potential solutions, the best candidate solution, 0, for this example, is obtained very quickly but as the

stopping criterion is not met, HH2adap continues to explore different possible solutions.

When solving the Ikegami3d1 instance, Figures 6.2(c) illustrates a similar behaviour in the first stage of the algorithms to the first stage of the Ikegami2d1 problem (Figure 6.2a). In the second stage (Figure 6.2d), for HH1adap, a large jump in solution space is noticeable, at the beginning of the second stage of the search process. HH2adap obtains improving results more quickly than HH1adap but only obtains the best candidate solution two thirds into the search process.

For the MER-A example, as is shown in Figure 6.2(e), both HH1adap and HH2adap exhibit a similar behaviour in the first stage of the algorithms. For the second stage, Figure 6.2(f), HH1adap starts with a better solution (9770) and converges more rapidly than HH2adap towards the final solution. This is because H1adap is unable to find an improving solution. HH2adap jumps from one candidate solution to another solution, enabling a larger exploration of the search space of candidate solutions. Ultimately, this allows a better final solution.

6.5 Remarks

Four extremely effective hyper-heuristics are presented to solve real world nurse rostering problems. HH2adap is the best selection hyper-heuristic variant amongst all proposed hyper-heuristics, given the fact that HH2adap allows partial restarts through the use of an acceptance method which allows worsening solutions to be accepted. On the other hand, HH1 and HH1adap, which is strengthened by the use of a hyper-heuristic performing pure hill climbing with multiple solutions at the first stage, act as local search algorithms in the overall during the second stage, increasing the chances that they get stuck at a local optimum. Yet, the experimental results indicate the success of the HH1, HH2, HH1adap and HH2adap hyper-heuristics which obtained new best and optimal solutions in the overall on 19, 23, 20 and 26 benchmark instances, respectively. Two key properties of the proposed hyper-heuristics is that they have less number of parameters than the previously

proposed approaches for nurse rostering making them easy to implement and the generality level that they achieved. They do not require any modification given an unseen instance as long as the problem is defined in a standard format. Using multiple solutions at the start of the search process and employing a pure hill climbing stage turned out to be an effective technique as a part of the overall approach for nurse rostering.

The results for the nurse rostering domain show that selection hyper-heuristics are sufficiently general approaches that can automatically produce high quality nurse rosters, even if the characteristics of the problem instances vary extremely. Furthermore, although not expected, the proposed hyper-heuristics generated high quality nurse rosters which are comparable to and in some cases, even better than the rosters obtained from a method, specifically tailored to the relevant problems.

In the next chapter, the four proposed hyper-heuristics will be applied to problems in different domains.

Chapter 7

Selection Hyper-heuristics for Cross-domain Heuristic Search

As mentioned previously, this research's main goal is to establish a more general method to solve different healthcare problems. Two hyper-heuristic approaches are developed and applied to 43 different real world nurse rostering problems, the results are provided in chapter 6. In order to further evaluate the level of generality of these approaches and review the methodologies, the four hyper-heuristics developed are also applied to a variety of different problems in different domains. This chapter will focus on the problem domains included in the CHeSC competition.

In this chapter, a first comparison is made between both hyper-heuristic approaches developed and the 8 hyper-heuristics provided as examples for the CHeSC competition. Secondly, a set of initial experiments are performed to compare both variants of two hyper-heuristics on five instances from each CHeSC problem domain. The top two hyper-heuristics are kept and evaluated against the algorithms submitted to the CHeSC competition. As mentioned previously both hyper-heuristic approaches are different. Each strategy will be reviewed to see which approach is the best in the overall and/or in which cases an approach is better than the other one depending on the problem structure. Section 7.1 evaluates the performance of the four hyper-heuristics proposed in this thesis against the 8 hyper-heuristics provided as examples by the CHeSC organisers. In section 7.2 the experimental methodology is defined. Section 7.3 compares the performance of the original hyper-heuristics HH1 and HH2 with their adaptive versions HH1adap and HH2adap. Section 7.4 evaluates the hyper-heuristics against the best algorithms submitted to the CHeSC competition. In section 7.5 the results will be discussed and section 7.6 include concluding remarks.

7.1 Comparison of New Hyper-heuristics to Eight Examples of Hyper-heuristics for the Mock Competition and Parameter tuning

7.1.1 Experimental Methodology

Prior to evaluating the four hyper-heuristics against the instances included in the CHeSC competition. Both hyper-heuristics HH1 and HH2 are evaluated against 8 examples of hyper-heuristics provided by the CHeSC competition. Both HH1 and HH2 were run on each instance of each problem domain 30 times. The experiments were performed on Intel(R) Core(TM)2Duo CPU E8500 @3.16GHz. The 8 hyper-heuristic examples are run for 10 minutes. HH1 and HH2 are not run for as long, the first stage of the algorithms terminates when no further improvement can be made to the candidate solution through local search heuristics. The second stage of the algorithm stops after a pre-set time limit.

The comparison is based on the average objective function values over 30 runs on all instances used by the 8 examples of hyper-heuristics.

7.1.2 Performance Comparison of HH1 and HH2 with 8 Hyper-heuristics

In this section, the results obtained after having applied both original hyper-heuristics (HH1 and HH2) to 10 public instances, across three problem domains are reviewed. The public instances were used to evaluate the 8 hyper-heuristics provided as examples by CHeSC 2011.

Table 7.1 to 7.3 summarise the results for each problem domain. The first column contains the benchmark instances; the second to the ninth columns represent the average results found over thirty runs for the benchmark hyper-heuristics, the tenth column shows the average time taken over the thirty runs for HH1. The eleventh and the thirteenth columns present the average time taken in seconds over the thirty runs to obtain results with HH1. The twelfth column includes the average result found for the thirty runs for HH2. Table 7.1 provides the results for the maximum satisfiability problem. Table 7.2 includes results for the permutation flow shop problem and Table 7.3 shows the results obtained for the one dimensional bin packing problem. The times are all in seconds unless stated otherwise.

TABLE 7.1 Max-SAT Problems Results

	H1	H2	H3	H4	H5	H6	H7	H8	HH1	Time	HH2	Time
I0	47	35	23	38	125	14	51	46	136	23	136	22
I1	35	31	38	51	109	30	37	57	105	48	105	39
I2	32	24	26	44	115	27	29	54	102	35	102	25
I3	19	13	31	15	54	17	18	25	24	4	24	5
I4	11	8	39	32	56	33	10	42	46	17	46	17
I5	25	17	56	46	110	50	23	54	45	17	45	18
I6	7	6	12	12	16	10	6	18	64	13	64	13
I7	6	6	11	12	16	11	7	15	73	13	73	13
I8	9	8	13	17	26	14	10	21	57	15	57	16
I9	213	211	216	235	263	219	215	233	65	14	65	15

The results obtained by HH1 and HH2 are quite close for the Max-SAT, for the first four instances HH1 and HH2 obtain results close to example hyper-heuristic H5, which is the worst performing hyper-heuristic. For the next three instances I3, I4 and I5, the average results are middle-ranking. For I6, I7 and I8, HH1 and HH2 are the worst performing algorithms. For instance I9, HH1 and HH2 provide by far the best average results, this is due to the first stage of the algorithm.

TABLE 7.2 Permutation Flow Shop Problem Results

	H1	H2	H3	H4	H5	H6	H7	H8	HH1	T	HH2	T
I0	6381	6383	6368	6326	6387	6312	6391	6315	6412	3	6437	2
I1	6329	6336	6339	6263	6315	6271	6334	6265	6367	4	6367	2
I2	6404	6404	6398	6362	6407	6344	6404	6351	6435	4	6481	2
I3	6390	6389	6369	6366	6392	6350	6385	6366	6418	4	6481	2
I4	6481	6468	6439	6407	6468	6398	6480	6419	6509	4	6604	2
I5	10547	10549	10544	10503	10549	10500	10542	10522	10581	3	10636	2
I6	10965	10965	10965	10923	10965	10922	10968	10957	11034	3	11047	2
I7	26440	26440	26487	26382	26476	26424	26450	26406	26536	13	26536	13
I8	26984	26958	26998	26864	26974	26896	26928	26939	27031	10	27001	21
I9	26779	26754	26818	26721	26756	26764	26767	26726	26778	16	26748	14

The results for HH1 and HH2 are comparable to all results obtained by the 8 examples of hyper-heuristics, though, in significantly less computational time. The time allocated to running the hyper-heuristics examples provided was 10 minutes.

TABLE 7.3 One Dimensional Bin Packing Problem Results

	H1	H2	H3	H4	H5	H6	H7	H8	HH1	T	HH2	T
I0	0.0170	0.0170	0.0060	0.0117	0.0510	0.0157	0.0219	0.0717	0.0850	3	0.0906	1
I1	0.0164	0.0170	0.0070	0.0117	0.0500	0.0118	0.0211	0.0679	0.0080	3	0.0080	1
I2	0.0234	0.0235	0.0240	0.0230	0.0280	0.0230	0.0240	0.0310	0.1120	3	0.1090	1
I3	0.0248	0.0246	0.0260	0.0246	0.0320	0.0240	0.0260	0.0330	0.0863	3	0.0880	1
I4	0.0060	0.0070	0.0003	0.0045	0.0151	0.0068	0.0069	0.0220	0.0422	3	0.0435	1
I5	.00428	0.0085	0.0034	0.0037	0.0179	0.0084	0.0090	0.0024	0.0399	2	0.0441	1
I6	0.1148	0.0990	0.0110	0.0221	0.1720	0.0484	0.1388	0.1850	0.1750	3	0.1834	1
I7	0.1360	0.1360	0.0190	0.0640	0.1820	0.0840	0.1510	0.1841	0.1814	2	0.1897	1
I8	0.0550	0.0544	0.0580	0.0920	0.0930	0.0610	0.0560	0.1260	0.1620	2	0.1622	2
I9	0.0124	0.0113	0.0160	0.0267	0.0350	0.0170	0.0150	0.0429	0.0560	2	0.0560	2

The same observations can be made for the one dimensional bin packing problems as for the permutation flow shop problems.

7.1.3 Parameter Tuning

For HH1 and HH1adap, a ruin and recreate low level heuristic is applied with a probability p of 0.3. Different values were evaluated for p. Experiments were run on each instance included in this chapter. The experiments were done on probabilities ranging from 0.1 to 0.9. The results obtained indicated that 0.3 was the best choice for p. Table 7.4 provides the average objective function value for probability values of 0.1, 0.3, 0.5, 0.7 and 0.9 for one instance in each four problem domains.

Table 7.4 Average objective function value for each probability for one instance in each problem domain included in CHeSC 2011, where BKN includes the best known solution for the instance evaluated. The best average objective function values are in bold.

	Probability					
Instances	0.1	0.3	0.5	0.7	0.9	BKN
Max-SAT	35	**29**	**29**	37	34	1
FS	**6800**	**6800**	7230	7280	7650	6214
BP	0.1390	0.1390	**0.1370**	0.1380	0.1384	0.1083
PS	410	**380**	410	420	430	280

7.2 Experimental Set-up

This sub-section provides the experimental set-up for all tests conducted in this chapter. All four hyper-heuristics were run on each instance of each problem domain 31 times. The experiments were performed on Intel(R) Core(TM)2Duo

CPU E8500 @3.16GHz. The proposed hyper-heuristics were tested on the CHeSC 2011 benchmark. As mentioned previously, the first part of both algorithms uses a common structure. There is no time limit on the iterative local search, the search ceases when no further improvements to the candidate solution are found. The second part for both algorithms has a pre-set time limit. Each algorithm was run for 10 nominal minutes which was adjusted based on the benchmarking tool provided on the CHeSC website. The same methodology was adopted to run the modified versions of both hyper-heuristic approaches.

First, HH1 is compared with HH1adap. The comparison is performed based on the average objective function values over 31 runs on all competition instances. The same comparison is done between HH2 and HH2adap. The results are provided in section 7.3. In section 7.4, the top two hyper-heuristics from the previous set of experiments are evaluated against the algorithms submitted for the competition. The proposed hyper-heuristics are evaluated against the competition algorithms based on the F1 scoring system.

7.3 Performance Comparison of the Proposed Hyper-heuristics

In this section, the results obtained after having applied both original hyper-heuristics (HH1 and HH2) and their modified versions (HH1adap and HH2adap) to all competition instances across six problem domains are reviewed. Table 7.5 summarises the results for each problem domain. The student's t-test is performed between the original and modified hyper-heuristics using the results from 31 runs for each problem instance. The following notation is used: A > B indicates that the algorithm A performs better than the algorithm B and this is statistically significant within 95% confidence while < denotes vice versa. A $\geq$ ($\leq$ or $\approx$) B indicates that the algorithm A performs slightly better (worse or no different) than the algorithm B. The average objective function value over 31 runs is provided for each hyper-heuristic for a given instance. The best values are identified in bold.

TABLE 7.5 Average objective values achieved by the proposed hyper-heuristics for CHeSC instances

		HH1adap	vs	HH1	HH2adap	vs	HH2
Max-SAT	inst0	9	≥	11	**3**	>	8
	inst1	12	>	19	**2**	>	13
	inst2	15	>	25	**2**	>	17
	inst3	6	>	9	**4**	>	8
	inst4	9	>	32	**2**	>	24
FS	inst0	6228	≥	6329	**6217**	>	6369
	inst1	26931	≤	26730	**26262**	>	26851
	inst2	**6341**	≥	6375	6439	≥	6463
	inst3	11477	≤	11468	**11552**	≥	11557
	inst4	26830	≥	26901	**26654**	>	27061
BP	inst0	**0.0179**	>	0.0883	0.0186	>	0.0948
	inst1	**0.0079**	>	0.0519	0.0086	>	0.0551
	inst2	0.0063	>	0.0763	**0.0055**	>	0.0848
	inst3	**0.1188**	>	0.1729	0.1298	>	0.1875
	inst4	**0.0085**	>	0.1258	0.0097	>	0.1277
PS	inst0	19	>	25	**17**	>	22
	inst1	9676	>	9963	**9563**	>	9949
	inst2	3242	≥	3335	**3210**	≥	3306
	inst3	1565	≥	1678	**1555**	>	1925
	inst4	**293**	>	323	325	>	425
TSP	inst0	48104.9	>	49043.7	**45883.1**	>	56124.2
	inst1	**21143872.5**	≥	21143873.0	21356911.0	≈	21356911.0
	inst2	**6821.7**	>	7000.7	6898.0	≥	7077.7
	inst3	**66838.0**	>	68740.0	67131.0	>	69860.2
	inst4	54058.8	≥	54068.9	**52943.2**	>	56124.2
VRP	inst0	74960.1	>	134940.1	**72438.2**	>	131834.2
	inst1	13375.8	>	18276.8	**12396.9**	>	16787.8
	inst2	158717.7	>	271587.2	**146896.6**	>	253391.3
	inst3	23158.1	>	25155.9	**20668.1**	>	22728.4
	inst4	160264.7	>	201279.7	**146372.3**	>	200987.7

The t-test between HH1 versus HH1adap and HH2 versus HH2adap shows that the differences in the average performance of the original and modified versions of the hyper-heuristics are statistically significant for almost all instances. For inst1 and inst3 of the permutation flow shop problem domain, HH1 performs slightly better than HH1adap. The average performances of the modified hyper-heuristics are in general better than their original versions almost across all problem domains. HH1adap performs significantly better than HH1 on 20 out of 30 instances, while HH2adap is significantly better than HH2 on 25 out of 30 instances. HH2adap delivers the best performance over all problem instances for the Maximum Satisfiability and the VRP domains. In the personnel scheduling and permutation flow shop problem domains,

HH2adap performs better than HH1adap in 4 out of 5 instances based on average objective values, while the situation is reversed in the one dimensional bin packing problem domain.

7.4 Performance Comparison of the Proposed Modified Hyper-heuristics to the CHeSC Competitors

In this section, the median and best performance of the top two hyper-heuristics; namely, HH1adap and HH2adap are compared to that of the best performing hyper-heuristic for each instance from the CHeSC 2011 benchmark. Table 7.6 includes the median and best objective values across 31 runs for a given competition instance.

TABLE 7.6 Comparison of the best performing hyper-heuristic among CHeSC competitiors denoted as Best-HH and proposed hyper-heuristics on all problem domains using objective function values from 31 runs for each instance.

		Median of runs			Best of runs		
		Best-HH	HH1adap	HH2adap	Best-HH	HH1adap	HH2adap
Max-SAT	inst0	3	12	**2**	**0**	4	**0**
	inst1	3	10	**2**	**1**	5	**0**
	inst2	**2**	16	**2**	**0**	6	**0**
	inst3	3	6	4	**1**	4	**1**
	inst4	7	8	**2**	7	6	**1**
FS	inst0	6240	6230	**6228**	**6214**	6218	**6214**
	inst1	26800	26901	**26202**	26722	26898	**26198**
	inst2	**6323**	6325	6339	**6290**	6323	6298
	inst3	**11359**	11468	11539	11318	11368	**11310**
	inst4	**26602**	26730	26666	**26535**	26598	**26535**
BP	inst0	**0.0161**	0.0179	0.0185	**0.0131**	0.0172	0.0183
	inst1	**0.0032**	0.0076	0.0088	**0.0028**	**0.0028**	0.0083
	inst2	**0.0036**	0.0054	0.0055	**0.0004**	0.0051	0.0051
	inst3	**0.1083**	0.0054	0.1278	**0.1083**	0.1149	0.1277
	inst4	**0.0035**	0.0088	0.0099	**0.0031**	0.0083	0.0094
PS	inst0	18	18	**17**	11	11	**9**
	inst1	9625	9615	**9565**	**9325**	9605	9400
	inst2	3223	3213	**3206**	3124	3204	**3110**
	inst3	1558	1545	**1525**	1350	1530	**1300**
	inst4	315	295	**305**	280	280	**270**
TSP	inst0	48194.9	48194.9	**45879.1**	48194.9	48032.2	**45473.6**
	inst1	**20822145.7**	21143872.5	21356711.0	**20747367.7**	21076843.1	21330170.0
	inst2	**6810.5**	6811.7	6816.0	6796.0	6756.2	**6316.6**
	inst3	**66756.2**	66840.0	67127.0	**65958.6**	66058.5	66860.0
	inst4	**52925.3**	54068.8	52938.2	**52053.4**	53444.1	52126.5
VRP	inst0	**60608.2**	74940.1	72418.2	**58052.1**	66411.1	64089.0
	inst1	**12290.0**	13276.8	12394.9	**12263.0**	12276.3	12278.0
	inst2	**145333.5**	158727.2	146696.6	**142517.0**	157649.7	145433.4
	inst3	20650.8	23155.9	**20568.1**	20650.8	21884.6	**19865.2**
	inst4	147124.6	160279.7	**146227.3**	144269.4	158279.8	**141527.7**

The results show that HH2adap outperforms HH1adap achieving 5 draws and 12 wins out of 30 considering the best performance of the best hyper-heuristic from CHeSC for each instance. HH1adap generates only a single draw considering the best performance of the CHeSC competitors. Based on the median performances of these hyper-heuristics, HH2adap again beats HH1adap with 13 wins and one draw out of 30, while HH1adap has no wins.

HH1adap and HH2adap are put into competition with all CHeSC 2011 hyper-heuristics and their performance is evaluated based on the F1 scoring system. HH2adap is an efficient and effective hyper-heuristic which outperforms the best ranking algorithm in the competition; namely AdapHH, Misir et al. (2012) with an overall score of 203 points. HH1adap, although being inferior to HH2adap, ranked the third generating an overall score of 134 points. Figure 7.1 illustrates the performance of HH1adap, HH2adap and top three ranking hyper-heuristics from the competition for each problem domain based on F1 scores. HH2adap is the best hyper-heuristic in the maximum satisfiability and the vehicle routing problem domains based on the F1 scores. HH1adap and HH2adap have an identical total F1 score and rank first for the personnel scheduling domain. For the one dimensional bin packing problem, the score of HH1adap is slightly better than HH2adap. HH1adap and HH2adap rank the 5^{th} and 6^{th}, respectively among all other CHeSC 2011 hyper-heuristics which joined the competition for this domain. For the permutation flow shop problem, HH2adap scores slightly higher than HH1adap ranking 4^{th}, while HH1adap ranks 6^{th} among all competing hyper-heuristics. For the travelling salesman problem, both hyper-heuristics have an identical total F1 score ranking in 3^{rd} position in the competition.

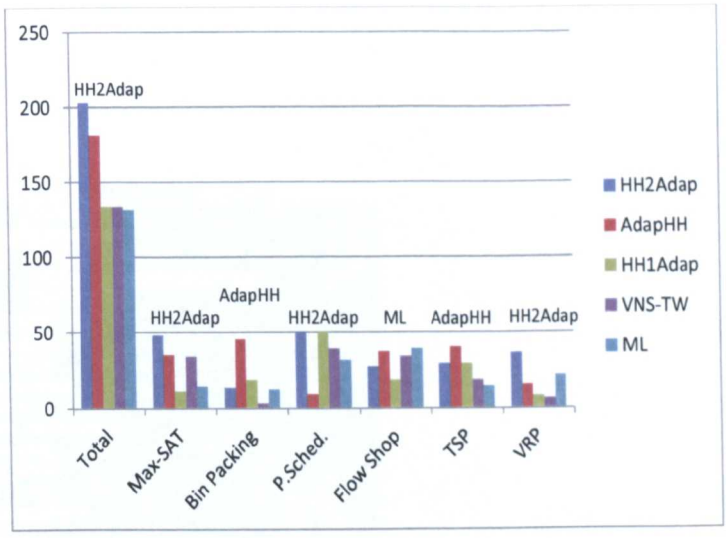
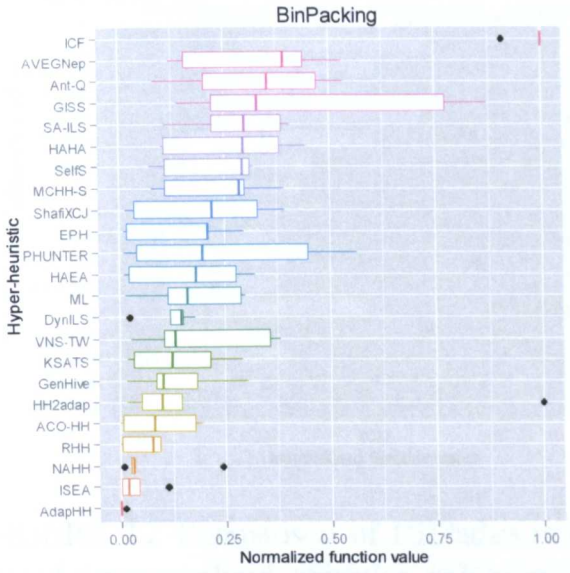
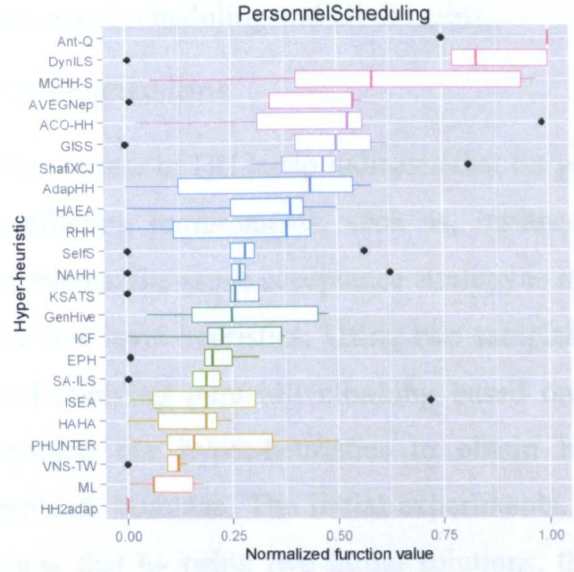


FIGURE 7.1 Overall and domain based score of the proposed and the top 3 hyper-heuristics from CHeSC.

To provide another comparison method of the algorithms, Figure 7.2 includes a normalised version Di Gaspero and Urli (2012) of the median objectives values of HH2adap (omitting HH1adap) and the algorithms submitted to CHeSC 2011. A normalised value is a mapping of a median objective value to a value between 0 and 1. Figure 7.2 illustrates the results for the bin packing domain, the personnel scheduling domain and the overall results. These domains are chosen because with the F1 scoring system HH2adap obtains its worst results on the bin packing domain and its best results on the personnel scheduling problems. With this other evaluation method, HH2adap is still the best hyper-heuristic overall, performing better than the other competing algorithms on all personnel scheduling instances included in CHeSC 2011. HH2adap's performance is worse on the bin packing domain, ranking sixth amongst all algorithms.


BinPacking
ICF
AVEGNep
Ant-Q
GISS
SA-ILS
HAHA
SelfS
MCHH-S
ShafiXCJ
EPH
PHUNTER
HAEA
ML
DynILS
VNS-TW
KSATS
GenHive
HH2adap
ACO-HH
RHH
NAHH
ISEA
AdapHH
Hyper-heuristic
0.00
0.25
0.50
0.75
1.00
Normalized function value


PersonnelScheduling
Ant-Q
DynILS
MCHH-S
AVEGNep
ACO-HH
GISS
ShafiXCJ
AdapHH
HAEA
RHH
SelfS
NAHH
KSATS
GenHive
ICF
EPH
SA-ILS
ISEA
HAHA
PHUNTER
VNS-TW
ML
HH2adap
Hyper-heuristic
0.00
0.25
0.50
0.75
1.00
Normalized function value

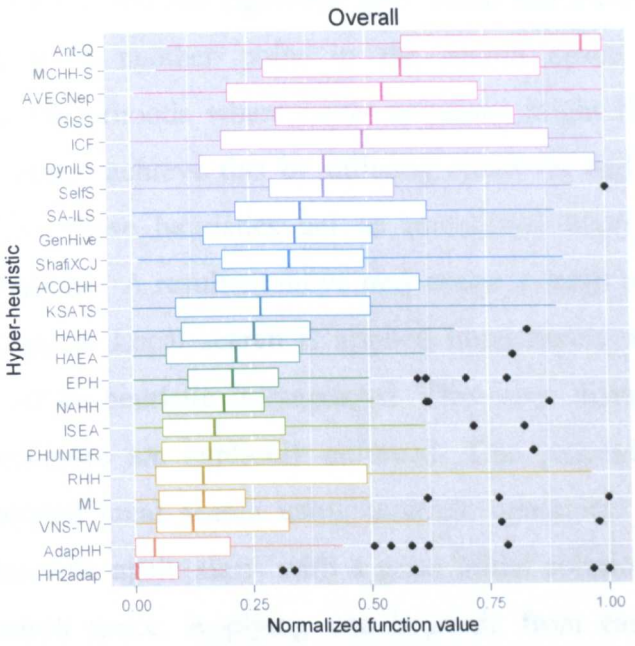


FIGURE 7.2 Comparison of HH2adap to the other CHeSC hyper-heuristics based on normalised objective values across all instances on bin packing, personnel scheduling and all domains.

7.5 Discussions

The success of HH2adap indicates that for problems with many constraints and conflicting requirements, such as, instances from the personnel scheduling problems, the move acceptance strategy is more important than the selection of the low level heuristics. Using two samples from the search space at the start and applying pure hill climbing based on local search low level heuristics enabled our hyper-heuristics to obtain better quality solutions across all problem domains. The initial experiments, which were reported in chapter 4, show that by using two initial solutions, the improvements in the results are between 12 and 20% better than using one initial solution for the six problem domains. When starting the search process using more than two initial solutions only marginally better results were obtained, for example, for the nurse rostering domain the improvement on the final solution between using 2 and 3 solutions is of 0.3%. This is not a worthwhile trade-off between computational time and results and so two initial solutions are used during the first stage. Given the time restriction, it makes sense to perform hill climbing as long as possible and relying on the internal diversification mechanisms within the local search heuristics if there is any. When the search process

stagnates and the algorithm gets stuck, and therefore it is essential to make a jump to another point in the search space to explore other potential neighbourhoods where better solutions might lie. The first type of hyper-heuristic achieve this by utilising crossover and ruin and recreate heuristics, since these heuristics act as mutational heuristics in almost all problem domains. A resultant move may create a small or a large variation in the new solution. Local search is applied immediately after a crossover or ruin and recreate heuristic is employed. This way, diversification and intensification processes are explicitly enforced. This process is a generalised version of iterated local search using multiple operators. The second type of hyper-heuristic again starts with a good initial solution i.e. a local optimum in the search space. Applying one heuristic from each category and keeping the resulting solution helps to explore a different part of the search space. Diversification and intensification is achieved via not only application of mutational and local search heuristics, but also accepting a move which produces the greater change in the quality of a solution.

TABLE 7.7 Average percentage of improvement at each stage of the hyper-heuristics and average percentage of time taken at each stage by the hyper-heuristics across each problem domain.

		%-improv		%-time	
		RP-LS	rest	RP-LS	rest
Max-SAT	HH1	31	31	37	63
	HH1adap	42	32	38	62
	HH2	32	23	39	61
	HH2adap	41	23	37	63
FS	HH1	31	31	37	63
	HH1adap	42	32	38	62
	HH2	32	23	39	61
	HH2adap	23	23	37	63
BP	HH1	33	24	35	65
	HH1adap	43	24	35	65
	HH2	35	25	37	64
	HH2adap	43	27	28	72
PS	HH1	41	31	39	61
	HH1adap	43	32	37	63
	HH2	41	23	39	61
	HH2adap	42	23	36	64
TSP	HH1	35	24	35	65
	HH1adap	44	25	35	65
	HH2	29	25	35	65
	HH2adap	41	26	36	64
VRP	HH1	32	23	42	58
	HH1adap	44	25	38	62
	HH2	29	24	41	59
	HH2adap	41	26	40	60

The proposed hyper-heuristics have two successive stages: local search (RP-LS: lines 1-3 of the pseudo-codes for HH1 and HH2) and the *rest*. We looked into the average percentage improvement in the quality of solutions after each stage for each hyper-heuristic along with the time spent during a stage as provided in Table 7.7. The second stage of a hyper-heuristic is more time consuming than the first stage for all problem domains. The second stage of the algorithm takes 63% of the overall execution time on average across all instances. Although the second stage takes more time, the percentage improvement obtained at the end of the second stage is in all cases smaller when compared to the percentage improvement obtained at the end of the first stage. The first stage of the algorithms improves the initial solution from 30-40%, whereas the second stage yields an improvement of 20-30%.

The proposed modification in both hyper-heuristics improves the efficiency of the local search stage. At first glance, considering their average performances,

it looks like as if both hyper-heuristics perform similarly. For the maximum satisfiability problem and the vehicle routing problems it is found that HH2adap provided noticeably better results than HH1adap. For both problem domains, the results using HH2adap were the best or the second best on five instances with the exception is the VRP *inst0* instance where the score is 0. The total F1 score for HH2adap for both these problem domains is the highest when compared to the other CHeSC hyper-heuristics. HH2adap algorithm produced better results on highly constrained personnel scheduling problems. The maximum satisfiability problem is also generally highly constrained, once the local search stage is completed, allowing worsening solutions introduces the diversity required and yields good results eventually.

In order to further analyse our approaches, a final set of experiments has been conducted across all instances. Each hyper-heuristic is run for 10 minutes on a problem instance and the objective function value of the current solution is recorded every 600 milliseconds until the time limit is reached. Figure 7.3 illustrates a sample run on *inst4* from the maximum satisfiability domain. The first stage of the modified versions of the hyper-heuristics can find better solutions in less time when compared to the regular versions. The modified hyper-heuristics spend more time in the second stage attempting to improve a solution in hand further through diversification and intensification steps. A similar behaviour is observed for the rest of the Max-Sat instances along with most of the other instances from the other domains. A *good* solution is found before the time limit ends for most of the problems. There are as can be seen from Table 7.6 a few exceptions. Both approaches do not yield as good results for the bin packing problem except for *inst1* where the first approach obtains the best result amongst all CHeSC competitors. For the vehicle routing problem domain, the second approach provides the best results for *inst3* and *inst4* but does not yield very good results for the other three instances.

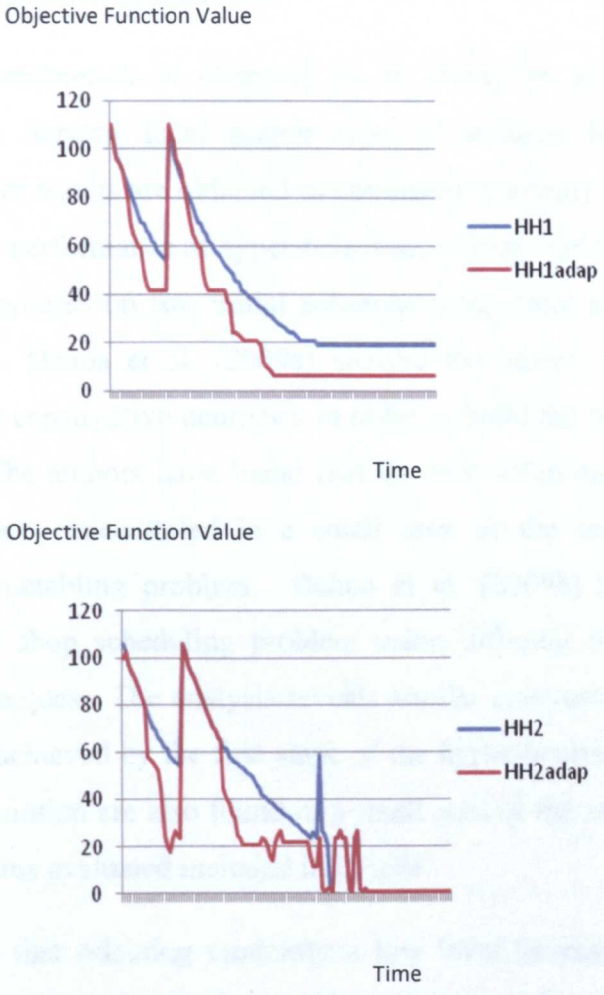


FIGURE 7.3 Objective function value of the current solution versus time plot from a sample run using each proposed hyper-heuristic on *inst4* from the Max-SAT domain.

For the personnel scheduling problems the best score on all instances are found by both hyper-heuristics, HH1adap and HH2adap. This is mainly due to the first improvement stage of the algorithms which make use of all local search algorithms. The local search algorithms included in HyFlex for this domain are very powerful. The improvements are made by identifying the key violations that the scheduler should remove; for example, if there is a night shift followed by an early shift as an unsatisfactory workload, the local search heuristic will identify this violation and attempt to rectify it by swapping between the same schedule or another person's schedule. As the operators are efficient for this domain, it is observed that a highly improved candidate

solution is obtained even before entering into the second stage of the hyper-heuristics as illustrated in Table 7.7.

The same phenomenon is observed as in Burke et al. (2010a) that by employing an iterated local search type of strategy based on multiple heuristics, better results are obtained as compared to simply using all low level heuristics. The performance of hyper-heuristics is even improved further by the hill climbing process on two initial solutions using local search heuristics in the first stage. Ochoa et al. (2009a) analyse the search space of a hyper-heuristic using constructive heuristics in order to build the best hyper-heuristic framework. The authors have found that the best solutions to an instance of the problem are concentrated in a small area of the search space on an educational timetabling problem. Ochoa et al. (2009b) studied the search space of flow shop scheduling problem using different representations and sampling techniques. The analysis reveals similar conclusions. Based on the improvement achieved by the first stage of the hyper-heuristics, it can be seen that the best solution are also found in a small area of the search space for the problem domains evaluated included in CHeSC.

It is observed that selecting randomly a low level heuristic from a specific category of low level heuristic provides similar results to preselecting and fixing a specific low level heuristic within a category. As the goal is to have a general framework and a heuristic from a category is expected to behave similarly, a random choice is preferred and proved to be successful at the end.

7.6 Remarks

In this study, four selection hyper-heuristics are described which are implemented as extensions to HyFlex. The best examples from previous work are used and analyses on the HyFlex problem domain implementations using the public instances as a guidance to design the proposed hyper-heuristics. The previous studies indicate the importance of local search as an intensification component and mutation as a diversification component Burke et al. (2009a, 2010a, 2009c, 2009d), Cowling et al. (2002). The balance between intensification and diversification is extremely important for a search methodology. Especially, if there is lack of guidance due to unfamiliar search

landscape, a managed trade-off between the intensification and the diversification of the search process considering the time restriction proves to be indispensable Özcan et al. (2006). That is the reason why two types of hyper-heuristics were designed, one of which explicitly enforces diversification and intensification under an iterated search like framework and another one automatically does that via a move acceptance method which enables large moves in the search space. Online learning mechanisms are certainly valuable for guiding the search process Burke et al. (2009d). In the thesis, the use of frameworks which are successful with strong empirical evidence in search and optimisation were preferred, emphasizing diversification and intensification either explicitly or implicitly within heuristic selection or move acceptance Özcan et al. (2006), Burke et al. (2009d). Given that a general time contract selection hyper-heuristics design is the goal and the problem domains provide effective local search operators, diversification is delayed and the search process is started with intensification only. It has been also observed that starting with the intensification process using two initial solutions instead of a single one, improves the quality of final solutions between 12 to 20%. The empirical results on the CHeSC 2011 benchmark show that the proposed hyper-heuristics are effective and efficient general search methodologies. Particularly, HH2adap outperformed the best hyper-heuristics from the CHeSC 2011 competition and to the best of the author's knowledge, there is no hyper-heuristic performing better than HH2adap in the literature, currently. After the intensification stage, the greedy hyper-heuristic component of HH2adap allows acceptance of a non-improving solution i.e. solution generating the largest worsening, enables a quick escape from one neighbourhood to another region of the search space that could potentially contain better solutions. HH1adap ranks third on the CHeSC 2011 benchmark.

Chapter 8

Surgery Admission Planning Using Selection Hyper-heuristics

The main goal of this part of the study is to design a framework to solve another healthcare problem for surgery scheduling. Considering the success of the proposed hyper-heuristics across a range of problem domains, including nurse rostering, and more importantly, observing that the selection hyper-heuristics do achieve a certain level of generality, a hyper-heuristic framework is designed and implemented for solving the surgery admission planning problem. This allows the testing of the four successful hyper-heuristics on this domain as well as comparison of their performance to a previously proposed approach. As mentioned previously, there is currently no consensus on the definition of surgery scheduling within the research community. This is mainly due to the fact that the surgery scheduling problem has not been studied extensively, for example as the nurse rostering problem.

The surgery admission planning problem can informally be defined as assigning known surgeries a day and relevant resources within the planning horizon subject to constraints. The surgeries can be scheduled within a prescribed delay from the patient's referral date. The surgeon must also be available and can only perform one surgery at a time. Once a surgery has been scheduled for a specific day, it is necessary to assign the time of the day for the surgery and the operating theatre. The model and datasets used have been defined in Chapter 2 and are real-world problems.

Section 8.1 describes the new hyper-heuristic software framework designed. Section 8.2 provides the experimental methodology and evaluation method. Section 8.3 focuses on the results obtained by the four hyper-heuristics. Section 8.4 evaluates the benefits of using multiple initial solutions. Section 8.5 provides concluding remarks.

8.1 A Hyper-heuristic Software Framework for Surgery Admission Planning

The software framework used for this study has previously been defined in Chapter 2. This subsection will describe the low level heuristics developed to address the surgery admission planning problems. The four hyper-heuristic strategies used for this problem domain are the same as the ones described in Chapter 5. In order to keep the same hyper-heuristic strategy for this domain the same types of low-level heuristics as in the benchmark software needed to be created.

The first step consists of assigning the surgeries that need to be scheduled to a specific day in the scheduling period. To solve the surgery admission planning problems, like Riise and Burke (2011), the initial schedules created include the surgeries that have been referred over the last 14 days and any older outstanding surgeries. The algorithm that creates the initial solution is detailed below.

The first surgeries assigned are the ones that have been referred for the longest period i.e. the surgeries are ordered by referral dates and waiting time, the algorithm will assign a day to a surgery by looking at the referral date. In cases where the referral dates are the same the urgency of the operations are considered. As an example if patient A and patient B have the same referral date, the deadline for the patient's operation is the element that decides which operation will be placed in the schedule first. The surgery with the longest referral date is placed in the first slot of the schedule for the planning period, the second in the second available slot and so on. It is important to note that when creating the initial schedule the duration time of the surgery is also considered as well the surgeon's availability. So the initial solution is a feasible solution. The schedule will include only days in which the maximum number of hours are permitted for the operating theatre. All surgeries included in the referrals are scheduled as long as the operating theatre is available which means the algorithm permits overtime for each surgeon of up to 3hours and 20 minutes. This amount of overtime is allocated for each operating theatre. Only surgeries that cannot fit in the schedule without

breaking the overtime of the operating rooms are left out. The surgeries that are left out are kept in a list of unassigned surgeries. Although the initial surgery schedule is feasible it may have a very costly objection function value as the overtime of surgeons is allowed.

Four types of low level heuristics: mutational, local search, crossover and ruin and create heuristics. These heuristics are used as low level heuristics under the control of the selection hyper-heuristics.

The software framework for the surgery admission planning problem includes three mutational, four local search, two crossover and three ruin and create heuristics.

Mutational Heuristics

The first mutational heuristic performs a swap between two days of surgery for one surgeon, the surgeries are selected randomly. The second swap is done between the longest and the shortest days for a selected surgeon. The third mutational heuristic consists of assigning and/or deleting surgeries from a schedule.

Local Search Heuristics

Three types of swaps between surgeries are proposed as move operators forming the basis for three local search heuristics. A move is then are accepted only when the overall quality of a schedule is improved.

The first algorithm sums each surgeon's total surgery time for each day of the planning period. The surgery time always includes the preparation time and recovery and cleaning time. The algorithm then selects the surgeon with the total longest duration time out of all days of the planning period and swaps surgeries between days to view if an improvement in the objective function can be made. More specifically, when the longest total duration time for all days is selected the algorithm then takes the longest surgery for that day for that surgeon and sees with which surgery it can be swapped in another day. Each possible swap is evaluated based on total duration of surgery for the surgeon for that day and the patient's deadline. The best swap is accepted and a new

candidate solution is obtained i.e. a new surgery schedule. Then the total surgery time per day per surgeon is calculated again and the swaps are done in the same way until no further improvements can be made to the schedule. This is an algorithm that swaps surgeries for the same surgeon over two days but looks at all possible swaps for each surgeon.

The second local search algorithm will calculate the longest days for each surgeon and the surgeon with the total longest day is selected. The swaps are done with the unassigned surgeries that are pre-assigned to the surgeon. The best swaps i.e. that provide the most improvement to the objective function are accepted.

The third local search heuristic operates on a similar principle although it enables a quicker solution time. Again for each day and each surgeon the total duration time of surgery is summed. The longest day for any surgeon is identified. For this same surgeon the shortest day of the planning period is also identified. Swaps are made between both days to provide the best duration. The best total duration is the one that provides a total day that enables an even distribution between both days. As an example surgeon A has 2 days of surgery identified, the longest and the shortest. One day one the surgeon has two surgeries one of 6 hours and one of 3 hours. On the other day the surgeon has a surgery of 3.5 and a surgery of 1.5 hours only. The best possible swap to respect the surgeon's overtime assuming that all swaps are possible between surgeries i.e. patient deadlines will still be met is to swap the 6 hour surgery with the 3.5 hours surgery, these will mean one day of 6.5 hours and the other day of 4.5 hours. The 4.5 hours day can then be possibly be assigned a surgery that is currently unassigned in the planning period.

The fourth local search heuristic selects randomly a day within the planning period and evaluates possible swaps between the selected day and the next work day for each surgeon. Only improving swaps are accepted.

Crossover Heuristics

Cross-over heuristics take two initial solutions and combine them. The first cross-over heuristic finds the best surgeries in each schedule. The best

surgeries are the one for which when the surgery is removed from the schedule it causes the largest increase in the objective function value. It is the patient waiting time or the overtime of the surgeon that has been increased. The best surgeries/day from each schedule will be combined to create the new schedule. Of course this means that the algorithm looks at the total surgery time per day and the patient's waiting time. The surgeries that remain will be assigned to fill the available periods, until the best combination is found. The best combination is found by evaluating the total waiting time of the patient and the duration time of the surgery.

The second cross-over heuristic will take the surgeries that are common to both initial solutions. The other surgeries are un-assigned and are used to create the complete schedule by the algorithm that builds a solution i.e. adding each surgery to a spot that creates the less increase in the objective function.

Ruin and recreate Heuristics

The heuristics randomly remove 3 surgeries from one, two or three surgeons' schedule and then rebuilds the schedule considering the patient's waiting time and the preferred window of operations.

The low level heuristics described above are used by the four hyper-heuristics that will be presented in chapter 5. The 4 hyper-heuristics developed for the surgery admission planning problem remain the same as the ones created for the other problem domains included in this research. The hyper-heuristics initially find a schedule in which the surgeries have been assigned i.e. each surgery is assigned to a day in the planning period, this is done while ensuring that no surgeon is allocated to one surgery at a time. However, as the surgery admission planning problem is in fact a two stage problem, the first stage solution is fulfilled by the hyper-heuristic by using the low level heuristics defined previously. The second stage of the problem which consists of assigning a time slot and an operating theatre for the scheduled surgeries for that day will be addressed by the following heuristics.

The algorithms developed for the assignment problem use the same logic as the heuristics used at the planning stage i.e. when assigning the surgeries to

each day. The algorithm starts with an initial solution in which surgeries assigned for that day are ordered by decreasing duration time. This means the longest surgery will be first on the list and the shortest will be last. The first 'n' surgeries i.e. the longest 'n' surgeries will be scheduled each in a different operating room. For the test data the number of operating rooms is 4 therefore the four longest surgeries are each assigned to one operating room. The next four surgeries are scheduled, this continues until all surgeries that are planned for that day have been scheduled. The second step of the algorithm consists of removing surgeries that would not respect the constraint relating to surgeon's being able to perform one surgery at a time. Various new positions are tried randomly for each of the surgeries that have been removed and the first slot tried that respects the constraint is accepted.

Once the initial solution is found, the total surgery time of each operating theatre is calculated and the surgeries in the operating theatre that has the longest hours for the day will be evaluated. First these surgeries will be ordered by decreasing order. The longest duration in the list will have the first position; this surgery will be swapped with another surgery in another operating theatre. The other operating theatre selected is the one with the shortest total duration. A swap is tried between the longest surgery and the shortest surgery from both operating rooms. A swap is only accepted if it does not increase the total objective function and the surgeon's schedule does not clash with this change.

8.2 Experimental Methodology and Evaluation Method

This sub-section provides the experimental set-up for all tests conducted in this chapter and defines the evaluation method. All four hyper-heuristics were run on each instance of each problem domain 31 times. The experiments were performed on Intel(R) Core(TM)2Duo CPU E8500 @3.16GHz. The proposed hyper-heuristics were tested on the surgery admission planning benchmark problems provided by SINTEF. As mentioned previously, the first part of both algorithms uses a common structure. There is no time limit on the iterative local search, the search ceases when no further improvements to the candidate solution are found. The second part for both algorithms has a pre-set time

limit. For the third stage of the algorithm which was created uniquely for the surgery admission planning problem, the algorithm stops when no further improvements can be made to the candidate solution.

As outlined previously, the hyper-heuristics need to provide good results in an acceptable computational time. The surgery admission problem datasets studied in this thesis, were generated and solved by Riise and Burke (2011); no other researchers have yet provided results using this data. Their results provide a benchmark to evaluate both algorithmic approaches provided in the thesis.

8.3 Performance Comparison of the Proposed Hyper-heuristics

In this section, the results obtained after having applied both original hyper-heuristics (HH1 and HH2) and their modified versions (HH1adap and HH2adap) to all surgery admission planning are reviewed. Table 8.1 and table 8.2 summarise the results for HH1 and HH2 respectively. Tables 8.3 and 8.4 provide the results obtained using HH1adap and HH2adap.

The first column includes the dataset, the second column represents the average objective value, calculated over the 31 runs and the third column contains the average time taken over the 31 runs to obtain the final objective value. The fourth column shows the best objective value obtained over the 31 runs. Column five includes the number of times the best solution is obtained by the hyper-heuristic evaluated over the 31 runs. The new best known solutions are highlighted in bold. The standard deviation is given in column six and the best known solution is provided in column seven.

TABLE 8.1 Results Predetermined sequence non-worsening hyper-heuristic (HH1)

Instances	HH1	TimeHH1	Best	Frequency	stddev	BKN
WOT1	0.5340	49	0.5320	18	0.020	0.5315
WOT2	0.5040	42	**0.5030**	15	0.011	0.5276
WOT3	0.3200	41	**0.3140**	26	0.003	0.3149
WOT4	0.2450	32	**0.2450**	17	0.600	0.2489
WOT5	0.3433	36	**0.3431**	8	0.002	0.3433
WOT6	0.4200	43	0.4010	11	0.016	0.4002
WOT7	0.4540	42	**0.4530**	23	0.121	0.4681
WOT8	0.4417	47	**0.4400**	25	0.111	0.4407
WOT9	0.4350	46	**0.4281**	19	0.120	0.4290
WOT10	0.2950	52	0.2948	26	0.003	0.2947

As can be seen from Table 8.1, the results obtained are close or better than the best known results. For seven instances, HH1 provides the new best known solution. For the other 3 instances i.e. WOT1, WOT6 and WOT10, HH1 obtains results close to the best known solution.

TABLE 8.2 Results Greedy absolute largest change hyper-heuristic (HH2)

Instances	HH2	TimeHH2	Best	Frequency	stddev	BKN
WOT1	0.5320	47	0.5320	21	0.003	0.5315
WOT2	0.5100	42	**0.5030**	17	0.026	0.5276
WOT3	0.3320	52	**0.3130**	28	0.014	0.3149
WOT4	0.2450	47	**0.2440**	27	0.118	0.2489
WOT5	0.3433	39	**0.3432**	24	0.159	0.3433
WOT6	0.4190	51	0.4110	18	0.083	0.4002
WOT7	0.4620	44	**0.4530**	19	0.012	0.4681
WOT8	0.4427	56	0.4410	21	0.224	0.4407
WOT9	0.4630	46	**0.4281**	20	0.017	0.4290
WOT10	0.2975	52	0.2948	17	0.246	0.2947

HH2 provides on average results close or better in some cases than the best known solution. HH2 provides the new best known solution for 6 instances out of the ten. The results for the other instances are close to the best known solution.

TABLE 8.3 Results Adaptive Predetermined sequence non-worsening hyper-heuristic (HH1adap)

Instances	HH1adap	TimeHH1adap	Best	Frequency	std dev	BKN
WOT1	0.5310	47	**0.5300**	19	0.020	0.5315
WOT2	0.5030	42	**0.5030**	15	0.010	0.5276
WOT3	0.3100	42	**0.3110**	28	0.001	0.3149
WOT4	0.2200	47	**0.2180**	22	0.014	0.2489
WOT5	0.3330	39	**0.3320**	18	0.120	0.3433
WOT6	0.4000	51	**0.3900**	19	0.014	0.4002
WOT7	0.4400	44	**0.4300**	27	0.124	0.4681
WOT8	0.4407	56	**0.4400**	26	0.006	0.4407
WOT9	0.4299	46	**0.4281**	21	0.024	0.4290
WOT10	0.2747	52	**0.2678**	26	0.002	0.2947

HH1adap provides the best known results for all 10 instances. On average the results obtain are better or close to the best known solution.

TABLE 8.4 Results Adaptive Greedy absolute largest change hyper-heuristic (HH2adap)

Instances	HH2adap	TimeHH2adap	Best	Frequency	std dev	BKN
WOT1	0.5300	53	**0.5200**	23	0.021	0.5315
WOT2	0.4890	41	**0.4820**	19	0.011	0.5276
WOT3	0.3300	51	**0.3100**	28	0.002	0.3149
WOT4	0.2386	49	**0.2180**	27	0.011	0.2489
WOT5	0.3563	42	**0.3320**	25	0.002	0.3433
WOT6	0.4003	56	**0.3900**	23	0.020	0.4002
WOT7	0.4235	47	**0.4214**	22	0.011	0.4681
WOT8	0.4417	56	0.4410	21	0.115	0.4407
WOT9	0.4156	49	**0.4152**	21	0.164	0.4290
WOT10	0.2478	58	**0.2472**	18	0.002	0.2947

HH2adap obtains the best known result for 9 instances, the instance WOT8 is close to the best known solution.

Both hyper-heuristic approaches provide generally good results. As it has been done for the other problem domains studied, a comparison is done between the four hyper-heuristics. The student's t test is performed between HH1 and HH1adap and another test is done between HH2 and HH2adap, using the results from the 31 runs for each benchmark instance. Table 35 outlines the results. The following notation is used: A > B indicates that the algorithm A

performs better than the algorithm B and this is statistically significant within 95% confidence while < denotes vice versa. A ≥ (≤ or ~) B indicates that the algorithm A performs slightly better (worse or no different) than the algorithm B. The average objective function value over 31 runs is provided for each hyper-heuristic for a given instance. Table 8.6 provides a comparison between HH1adap and HH2adap using the same test and notation.

TABLE 8.5 Comparison between algorithmic approaches i.e. between HH1 and HH1adap and HH2 and HH2adap

Instances	HH1	vs	HH1adap	HH2	vs	HH2adap
WOT1	0.5340	≤	0.5310	0.5320	≤	0.5300
WOT2	0.5040	≤	0.5030	0.5100	<	0.4890
WOT3	0.3200	<	0.3100	0.3320	≤	0.3300
WOT4	0.2450	≤	0.2200	0.2450	≤	0.2386
WOT5	0.3433	≤	0.3330	0.3433	≥	0.3563
WOT6	0.4200	<	0.4000	0.4190	≤	0.4003
WOT7	0.4540	≤	0.4400	0.4620	<	0.4235
WOT8	0.4417	≤	0.4407	0.4427	≤	0.4417
WOT9	0.4350	≤	0.4299	0.4630	≤	0.4156
WOT10	0.2950	<	0.2747	0.2975	≤	0.2478

The comparison indicates only 3 instances where HH1 is statistically significantly better than HH1adap and 2 instances where HH2adap is statistically significantly better than HH2. From these results alone, it is not possible to conclude that the adaptive version of the algorithm is better than the non-adaptive version. However, the adaptive version either provides the new best known solution or obtains it more frequently over the 31 runs. The best results between HH1 and HH1adap are found by HH1adap for 8 instances. For the other 2 instances (*WOT2 and WOT9*) the best results are the same and the average objective values for both instances are better for HH1adap. HH2adap finds better best results for 9 instances out of 10 when compared to HH2, for the other instance (*WOT8*) HH2adap provides a better average objective value. These results demonstrate that the added intensification and diversification step of the algorithms provides better results.

As per the previous chapters, a comparison will be made between the HH1adap and the HH2adap approaches. One method of comparison is again the student's t test. The results will be included in Table 8.6, where the same methodology and notation is used as for the previous comparisons done in Table 8.5.

TABLE 8.6 Comparison between algorithmic approaches i.e. between HH1adap and HH2adap

Instances	HH1adap	vs	HH2adap
WOT1	0.5310	≤	0.5300
WOT2	0.5030	<	0.4890
WOT3	0.3100	>	0.3300
WOT4	0.2200	>	0.2386
WOT5	0.3330	≥	0.3563
WOT6	0.4000	≥	0.4003
WOT7	0.4400	≤	0.4235
WOT8	0.4407	≥	0.4417
WOT9	0.4299	≤	0.4156
WOT10	0.2747	<	0.2478

For four instances it can be established that there is method that is statistically significantly better within 95% confidence. For instances *WOT3* and *WOT4*, HH1adap is better, and for *WOT2* and *WOT10* HH2adap, is better. HH1adap and HH2adap are both slightly better on 3 instances each. With these results, it is necessary to look at other indicators of performance. HH2adap obtains the best known solution for 6 instances of the surgery admission problem i.e. instances *WOT1, WOT2, WOT3, WOT7, WOT9 and WOT10*, whereas HH1adap provides the best known results for one instance of the problem i.e. *WOT8*. For the other instances, both methods obtain the same result, which is the best known solution. When the best solution is found both by HH1adap and HH2adap, HH2adap finds the best value more often over the 31 runs than HH1adap. This is the case for instances *WOT4, WOT5 and WOT6*. Though, it takes HH2adap more time to achieve these results for these four instances. Table 8.7 contains the average time in percentage taken at the first two stage of each algorithm to obtain the final solution and the average percentage of improvement in the quality of the solution after each stage for each hyper-heuristic. Note the third stage of the algorithm is not provided as this stage is

common to all four hyper-heuristics. This is compiled over 1000 iterations of 31 runs. The second stage for all four hyper-heuristics is more time consuming than the first stage. The second stage of the algorithm takes 62.75% of the overall execution time on average across all instances. Although the second stage takes more time, the percentage improvement obtained at the end of the second stage is in all cases smaller when compared to the percentage improvement obtained at the end of the first stage. The first stage of the algorithms improves the initial solution from 45-65%, whereas the second stage yields an improvement of 22-37%. From table 8.7, it can be observed that the added step of intensification and diversification, in the first stage of the algorithms, provides a larger improvement on solution quality in less time than the local search without this step. This is another indicator that the adaptive versions are better than the non-adaptive versions of the algorithms. The results are similar to the ones obtained for the other domains included in this study. It is worth noting that the first stage of HH1adap and HH2adap provides a larger percentage of improvement on the candidate solution than for any other problem domains.

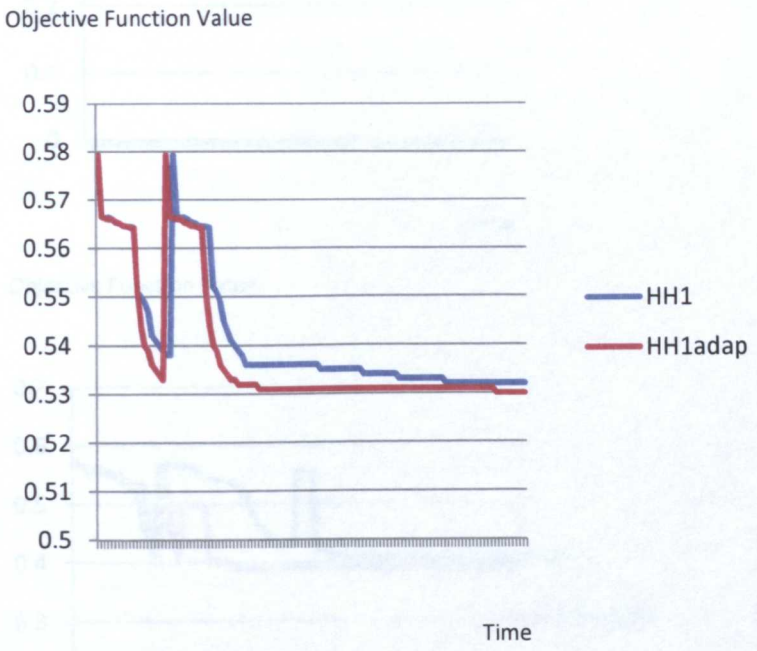
TABLE 8.7 Average percentage of improvement and time taken at stages 1 and 2 of the four hyper-heuristics for the surgery admission planning domain (SAP). The percentage of improvement and time taken for each stage for the domains included in CHeSC 2011 are also included for comparison purposes.

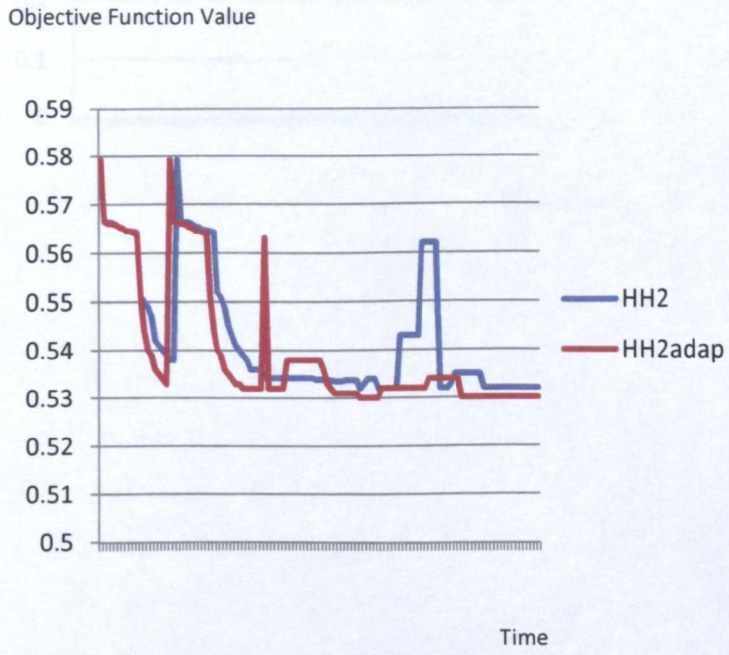
		%-improv		%-time	
		RP-LS	rest	RP-LS	rest
SAP	HH1	45	36	44	66
	HH1adap	65	22	33	67
	HH2	45	37	47	53
	HH2adap	64	31	35	65
Max-SAT	HH1	31	31	37	63
	HH1adap	42	32	38	62
	HH2	32	23	39	61
	HH2adap	41	23	37	63
FS	HH1	31	31	37	63
	HH1adap	42	32	38	62
	HH2	32	23	39	61
	HH2adap	23	23	37	63
BP	HH1	33	24	35	65
	HH1adap	43	24	35	65
	HH2	35	25	37	64
	HH2adap	43	27	28	72
PS	HH1	41	31	39	61
	HH1adap	43	32	37	63
	HH2	41	23	39	61
	HH2adap	42	23	36	64
TSP	HH1	35	24	35	65
	HH1adap	44	25	35	65
	HH2	29	25	35	65
	HH2adap	41	26	36	64
VRP	HH1	32	23	42	58
	HH1adap	44	25	38	62
	HH2	29	24	41	59
	HH2adap	41	26	40	60

In order to further analyse the hyper-heuristic approaches, an additional set of experiments across all ten instances, is done. Each hyper-heuristic runs for 10 minutes on a problem instance and the objective function value of the current solution is recorded every 600 milliseconds until the time limit is reached. The stopping criteria of the second stage of the algorithms are relaxed to enable further improvements. As mentioned previously, the first stage of the algorithms stops when no further improvements can be made to either of the two random initial solutions, the stopping criteria remains the same. Figure 10 illustrates as an example a sample run on *WOT1*, *WOT6* and *WOT10*. These instances are chosen because of their size, *WOT1* is a smaller instance with 152 surgeries, *WOT6* can be considered a medium size instance with 166 surgeries

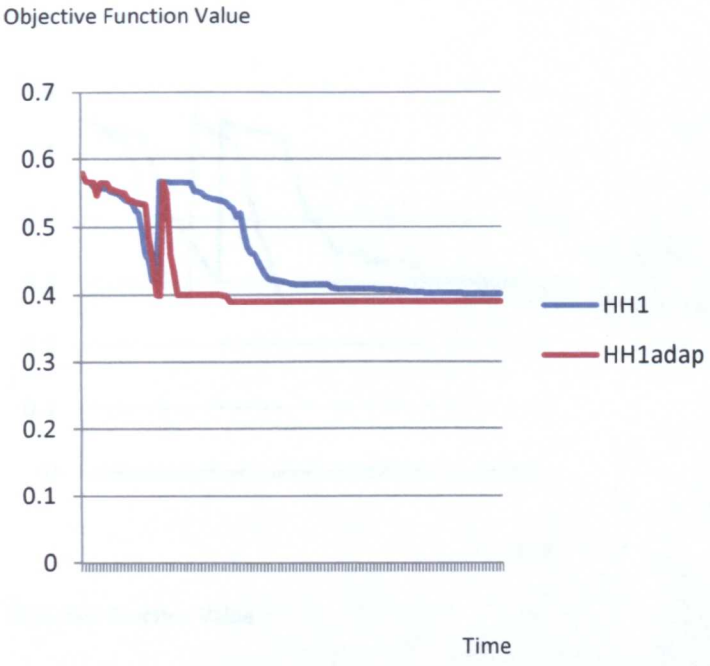
to schedule and the highest size is *WOT10* with 186 surgeries. The first stage of the modified versions of the hyper-heuristics can find better solutions in less time when compared to the regular versions. The modified hyper-heuristics spend more time in the second stage attempting to improve a solution in hand further through diversification and intensification steps. A similar behaviour is observed for the rest of the surgery admission planning instances. A *good* solution is found before the time limit ends for all the problems.

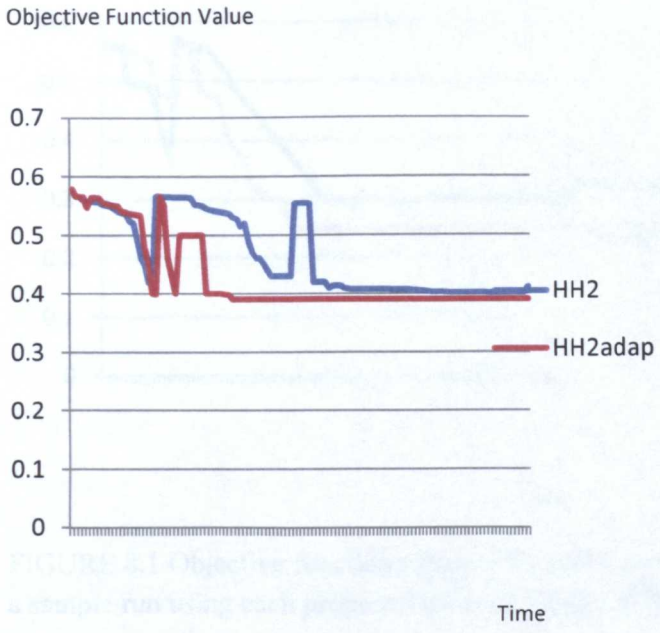
WOT1



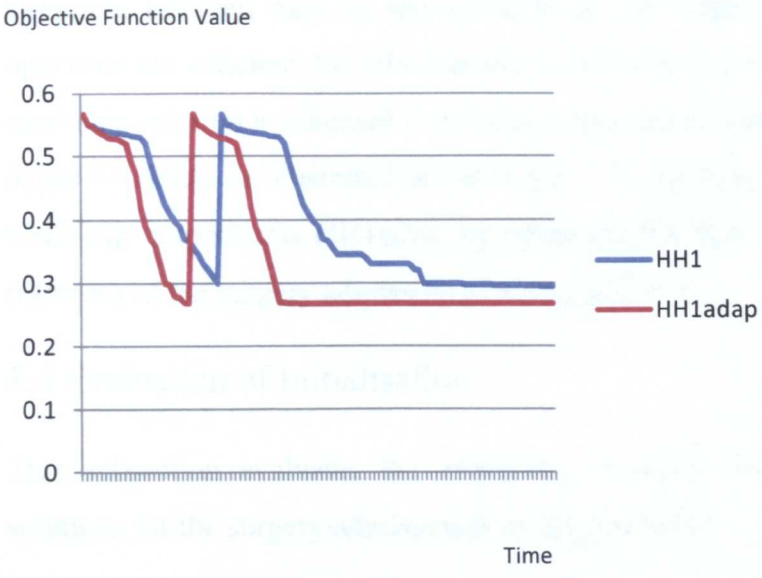


WOT6


Objective Function Value
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
HH1
HH1adap
Time


Objective Function Value
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
HH2
HH2adap
Time

WOT10



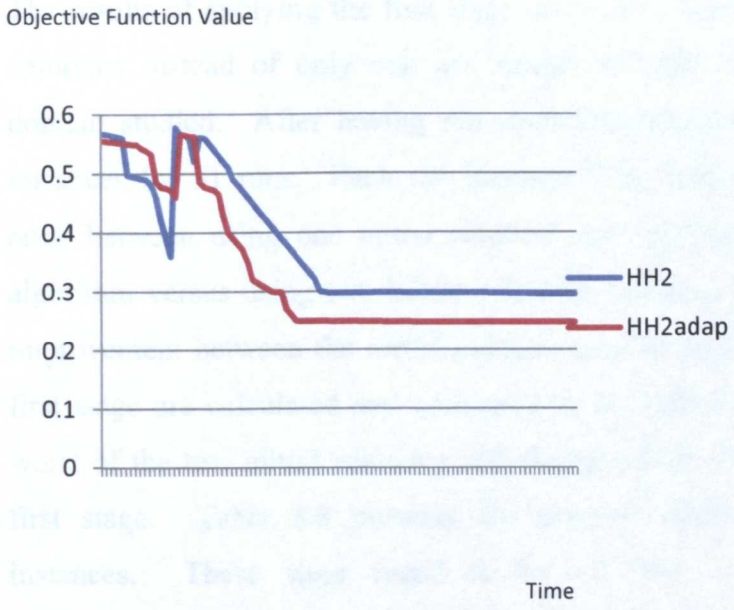


FIGURE 8.1 Objective function value of the current solution versus time graphic from a sample run using each proposed hyper-heuristic on *WOT1, WOT6 and WOT1.*

For all instances of the surgery admission planning problem, the first stage of the algorithm provides the most improvement to the random initial solutions. This is illustrated by the examples in figure 8.1, for all four hyper-heuristics the Random-permutation Local search stage provides the best improvement. When starting the second stage for all four hyper-heuristics, the improvement to the candidate solution is minimal. This is due to the efficiency of the local search algorithms for this domain. The improvements are made by identifying the key violations that the scheduler should remove; for example, if there is a day which requires overtime for a surgeon, making an unsatisfactory workload,

the local search heuristic will identify this violation and attempt to rectify it by swapping between days in the schedule or un-assigned surgeries. As the operators are efficient for this domain, it is observed that a highly improved candidate solution is obtained even before entering into the second stage of the hyper-heuristics as illustrated in Table 8.7. To find the best known solution, HH2adap outperforms HH1adap, by obtaining the best known solution for 6 instances of the surgery admission planning problem.

8.4 Evaluation of Initialisation

This subsection evaluates the efficiency of using multiple random initial solutions for the surgery admission planning problem.

The results of applying the first stage of all four hyper-heuristics on 2 initial solutions instead of only one are similar for this domain to the previous domain studied. After having run each hyper-heuristic on the 10 problem instances for 31 runs. Each run includes 1000 iterations. A comparison is done between using one initial solution and applying the first stage of the algorithm versus using two initial solutions followed by the first stage. The improvement between the initial solution and the candidate solution after the first stage are calculated and compared to the improvement provided by the worst of the two initial solutions and the candidate solution at the end of the first stage. Table 8.8 presents the average improvement across all ten instances. These were found to be of 18%. The total increase in computational time when using two initial solutions on which the step RP-LS is applied to each of these is of 19%. Conducting the same experiments when using 3 random initial solutions provides only an improvement of 20% on the candidate solution but the computational time was increased by 40%. As can be seen from the results in Table 8.8, using more than two initial solutions only provides marginally better results but increases the computational time consumed. This is not a worthwhile trade-off between computational time and results and so it was preferable to start with two initial solutions during the first stage. These results are similar to the ones obtained for the previous problem domain studied.

TABLE 8.8 Percentage of improvement and percentage of time used.

	%-improv		%-time	
	Two	Three	Two	Three
RP-LS	18	19	20	40

8.5 Remarks

The success of HH2adap for the surgery admission planning problem indicates that for highly constrained problems, the move acceptance strategy is more important than the selection of the low level heuristics. Using two samples from the search space at the start and applying pure hill climbing based on local search low level heuristics enabled the hyper-heuristics to obtain better quality solutions across all instances. Using two initial solutions, the improvements in the results are on average 18% better than using one initial solution for the surgery admission planning domain. These same conclusions were made for the previous problem domain included in the thesis. Performing hill climbing as long as possible and relying on the internal diversification mechanisms within the local search heuristics is efficient. Both HH1adap and HH2adap provide good results. However, HH2adap's strategy of diversification and intensification through not only the application of mutational and local search heuristics, but also by accepting a move which produces the greater change in the quality of a solution proves to be a better approach for the surgery admission planning problem.

Chapter 9

Conclusion and Future Work

In this study, four selection hyper-heuristics are developed to solve problems in seven different problem domains. All hyper-heuristics are designed as time contract algorithms which terminate when the given time limit is exceeded, as well as anytime algorithms which run until there is no improvement in the solution quality. Each hyper-heuristic is designed based on one of the two algorithmic approaches. One approach uses a predetermined template grouping each type of low level heuristics together. A heuristic is randomly selected and applied from a group at each stage of the search process in which the order of stages, hence groups is prefixed, enforcing diversification and intensification explicitly. The other approach employs a greedy strategy combined with a novel acceptance criterion. The second approach was established as the best strategy in the overall. Specifically, HH2adap proves to be a more effective method in finding high quality solutions across all seven problem domains by allowing acceptance of worsening solutions.

The balance between intensification and diversification is extremely important for a search methodology. Especially, if there is lack of guidance due to unfamiliar search landscape, a managed trade-off between the intensification and the diversification of the search process considering the time restriction proves to be indispensable. That is the reason behind the design of two types of hyper-heuristics, one of which explicitly enforces diversification and intensification under an iterated search like framework and another one automatically does that via a move acceptance method which enables large moves in the search space. It has been also observed that starting with the intensification process using two initial solutions instead of a single one, improves the quality of final solutions between 12 to 20%, depending on the problem domain.

The empirical results on the CHeSC 2011 benchmark show that the proposed hyper-heuristics are effective and efficient general search methodologies. Particularly, HH2adap outperformed the best hyper-heuristics from the CHeSC 2011 competition and there is currently no hyper-heuristic performing better than HH2adap in the literature. After the intensification stage, the greedy hyper-heuristic component of HH2adap allows acceptance of a non-improving solution i.e. solution generating the largest worsening, enables a quick escape from one neighbourhood to another region of the search space that could potentially contain better solutions.

The results on the nurse rostering problems also indicate that HH2adap is a powerful hyper-heuristic that obtained results close to the best known solutions for the 43 nurse rostering benchmark instances and in some cases provided the new best known results for 13 benchmark instances. The same phenomenon was observed for the surgery admission planning problem for 9 instances.

There was a need to develop heuristics to apply specifically to the surgery admission planning problem. Although this domain has some similarities with the nurse rostering problem it also has many differences.

The surgery planning problem can be represented as two sub-problems. The first sub-problem is to assign a surgery to a day within the planning horizon. The second sub-problem is to schedule a time and assign an operating theatre to the surgery for the specific day. There are many similarities between nurse rostering and surgery planning considering the first sub-problem. An appropriate day is sought for a given surgery and the surgeon has already been assigned who will perform the surgery. The hospital/work contracts and coverage constraints are similar in both nurse rostering and surgery planning problems. For example, nurses cannot work more than a determined number of hours or shifts in one day, and similarly, there is a limitation on the number of hours a surgeon may work during a day. For the surgery admission planning problem, the Norwegian hospitals typically wish to minimise overtime work done by the surgeons to minimise the cost and ensure a certain patient quality of care. The assignment of surgeries to the planning period also needs to

consider that a surgeon can only perform one surgery at a time just like the nurse can only be assigned one shift at a time.

However, the second sub-problem of assigning a surgery to each day a time and operating theatre is different to the nurse rostering problem. In fact, the problem of assigning the surgeries of each day to the operating room resembles that of assigning a job to a machine. For the permutation flow shop problem the goal is to minimise the total makespan time i.e. the total duration of processing all jobs on all machines. Precisely, this is done by finding the order that would considering a known duration time to process all jobs minimise the time it takes to process all jobs. For the surgery admission planning problem the problem consists of minimising the total duration time of surgeries this will ensure that we are minimising surgeon's overtime. The wish is to create the best sequence of operations for each operating room to fulfil all surgery commitments i.e. ensure all surgeries scheduled for that day are done while minimising the surgeon's time.

To the research question implied in section 1.2: Is it possible to develop a more general method to solve different instances of a problem and to solve a variety of different problems within different problem domains using the same general strategy? The four hyper-heuristics developed and particularly the HH2adap algorithm demonstrates that a more general method can be successfully applied to a variety of problem domains. The four hyper-heuristics were evaluated against problem solving scenarios over seven problem domains, even on some unseen instances for each domain.

9.1 Future Work

As future work, it would be of interest to explore the search landscape of different problem domains with the goal of understanding the proposed hyper-heuristics better. Moreover, these studies can enable us to form a hyper-heuristic portfolio indicating which type of hyper-heuristic is better suited to which type of problem.

All four hyper-heuristics developed for this thesis use the same low level heuristics from 4 categories i.e. ruin and recreate, local search, mutational and

cross-over heuristics. The surgery admission software planning domain is also built to include heuristics from these four heuristic types. In chapter 4, it was shown that the heuristics within all these categories behaved either as hill-climbing or mutational heuristics. It would be of interest to evaluate the hyper-heuristics in a different environment where such categories do not exist. This would have implication on their level of generality and this level would need to be evaluated in this new context.

In this study, only perturbative low-level heuristics were used. Another curiosity would be designing a selection hyper-heuristic which can control a mix of constructive and perturbative low level heuristics. Could the structure of the hyper-heuristics proposed manage such a mix? The proposed hyper-heuristics cannot handle such systems. Then what type of modification or minimal change in the design is necessary? These possibilities tend to bring us towards another level of generalisation; is it possible to generalise the performance of a hyper-heuristic even under different type of hyper-heuristic frameworks?

The four hyper-heuristics developed use parameters that remain static during the search phase. Would the value attributed to the parameters provide good results if applied to other problem domains? To increase the level of generality of the approaches, it would be useful to apply a learning approach to parameter tuning during the search process. An online learning method should also be added to the hyper-heuristic strategies developed for the selection of the low level heuristics.

The hyper-heuristic HH2adap proved to be efficient, this is due to the acceptance criteria. Currently some hyper-heuristics use a non-deterministic move acceptance criteria. These are based on the comparison with a parameter. Looking into the possibility of modifying the move acceptance criteria during the search process based on the results i.e. making it adaptive to the search space explored would be interesting and could enable the hyper-heuristics to solve problems from other domains, that have not been explored in this study, without adjustments to the hyper-heuristics.

For this study, four hyper-heuristics are applied separately to each problem instance. Could the level of generality be increased even more by providing a set of hyper-heuristic strategies that could be called in different circumstances depending on the structure of the problem?

The surgery admission planning problem modelled for this research included three objectives; minimising the surgeons overtime, minimising the patients waiting time and scheduling children early on the day of the surgery. The problem evaluated contained a static duration for the operation. This information was provided by the hospitals and depended on the surgery. When planning surgeries many difficulties arise, one major difficulty is the real duration of the surgery. In order to reflect the real-world problem more accurately, it is essential to add a stochastic element to the model. As a future work, the same problem will be addressed but it will include real surgery duration, that is variable. Two options are being explored, obtaining information on surgery duration from large Canadian hospitals and analysing how the hospital schedules change during the course of the day when surgeries have overrun.

For both the surgery admission planning and the nurse rostering problems, it would of interest to add components to the model studied for this research. For the surgery admission planning problem the integration of the nursing and other staff members would be of interest as would including the recovery stage to the model. The nurse rostering model, the idea of including float nurses might be useful to generalise the problem even further.

Appendix A

Analyses of Heuristics

In chapter 4, three sets of experiments where done on each low-level heuristic included in four problem domains in HyFlex. The first set of experiments consisted of applying iteratively each low level heuristic on an initial random solution over 31 runs. This was done for the one dimensional bin packing, nurse rostering, Max-SAT and permutation flow shop problems. Tables A.1 to A.4 present the results by domain for the first set of experiments. The value of each cell is the improvement between the current candidate solution and the previous candidate solution.

The second set of experiments consisted of applying iteratively each low level heuristic on an initial random solution for 30 seconds. This was done for the one dimensional bin packing, nurse rostering, Max-SAT and permutation flow shop problems. Tables A.5 to A.8 present the results by domain for the second set of experiments. The value of each cell is the improvement between the current candidate solution and the previous candidate solution.

The third set of experiments consisted of applying iteratively each low level heuristic on an improved initial solution over 31 runs. The improved initial solution is the solution of the previous experiment i.e. the final candidate solution after 30 seconds. This was done for the one dimensional bin packing, nurse rostering, Max-SAT and permutation flow shop problems. Tables A.9 to A.12 present the results by domain for the third set of experiments. The value of each cell is the improvement between the current candidate solution and the previous candidate solution.

TABLE A.1 Results over 31 runs of 1000 iterations for each low level heuristic for one instance, of the one dimensional bin packing domain.

LS4	LS6	MU0	MU3	MU5	RR1	RR2	CO7
0.1	0.12	0.02	0.4	0.47	-0.11	0.09	0.01
0.1	0.4	0.03	0.08	0.66	0.88	0.41	0.02
0.02	0.62	0.16	0.9	0.34	0.2	0.11	0.3
0.3	0.26	0.2	0.4	0.46	0.01	0.16	0.1
0.4	0.33	0.21	0.6	0.25	0.23	0.6	0.05
0.011	0.24	0.4	0.056	0.78	-0.1	0.5	0.16
0.1	0.48	0.08	0.23	0.41	0.23	0.4	0.2
0.12	0.11	0.9	0.3	0.11	0.24	0.23	0.2
0.23	0.21	0.4	0.4	0.29	0.14	0.41	0.2
0.23	0.14	0.6	0.4	0.31	0.26	0.55	0.2
0.23	0.76	0.056	0.28	0.46	0.3	0.6	0.2
0.23	0.1	0.23	0.28	0.42	0.14	0.1	0.2
0.23	0.1	0.3	0.28	0.42	0.17	0.2	0.2
0.23	0.02	0.4	0.28	0.74	0.7	0.01	0.2
0.23	0.3	0.4	0.28	0.41	0.41	0.23	0.2
0.23	0.4	0.28	0.28	0.43	0.31	-0.1	0.2
0.23	0.011	0.28	0.28	0.44	0.46	0.23	0.2
0.23	0.1	0.28	0.7	0.76	0.25	0.24	0.2
0.23	0.12	0.28	0.66	0.89	0.78	0.14	0.2
0.36	0.23	0.28	0.34	0.22	0.41	0.26	0.2
0.36	0.23	0.28	0.46	0.74	0.11	0.3	0.2
0.36	0.23	0.28	0.25	0.86	0.29	0.14	0.2
0.36	0.23	0.28	0.78	0.9	0.31	0.17	0.16
0.36	0.36	0.28	0.41	0.86	0.46	0.7	0.34
0.4	0.42	0.28	0.11	0.86	0.32	0.41	0.31
0.28	0.57	0.32	0.29	0.86	0.31	0.31	0.09
0.63	0.89	0.33	0.31	0.86	0.11	0.86	0.17
0.76	0.94	0.45	0.46	0.86	0.21	0.31	0.26
0.12	0.22	0.36	0.42	0.73	0.22	0.29	0.24
0.32	0.76	0.42	0.42	0.73	0.24	0.21	0.27
0.12	0.48	0.4	0.28	0.7	0.14	0.29	0.17

TABLE A.2 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the nurse rostering problem.

LS0	LS1	LS2	LS3	LS4	CO8	CO9	CO10	MU11	RR5	RR6	RR7
21373	18030	4540	4270	21307	20	96	12	300	140	140	1600
20700	3347	16807	20961	21307	40	40	88	140	120	120	1200
20700	3347	16807	20961	21307	14	12	76	12	0	0	0
20700	3347	16807	20961	21307	14	4	0	10	-120	-120	180
20700	3347	16807	20961	21307	220	210	140	0	-60	-60	-60
20700	3347	16807	20961	21307	220	450	450	0	180	180	-120
20700	3347	16807	20961	21307	220	380	12	0	450	450	560
20700	18030	16807	20961	21307	220	120	100	120	-320	3200	6400
20700	3776	16807	20961	21307	220	120	120	0	780	780	-700
21300	3776	16807	20961	21307	220	120	160	36	0	0	0
21300	3776	16807	20961	21307	220	120	180	0	0	0	100
21300	3776	16807	20961	21307	220	120	190	40	-360	-360	-360
21300	3776	16807	20961	21307	220	120	110	260	0	0	0
21300	3776	16807	20961	21307	220	120	120	0	-600	-600	-1600
21300	3776	17450	20961	21307	220	120	420	0	-1220	1600	1800
21300	3776	16450	20961	21307	220	120	460	0	0	0	100
21300	3776	14520	20312	21307	220	120	120	0	160	160	460
21300	3776	14520	20312	20790	260	120	114	0	2400	1800	1200
21300	3776	14520	20312	16540	324	120	380	0	0	0	120
21300	18030	14520	20312	14756	480	120	380	0	-220	-220	160
21300	18030	14520	20312	18942	410	120	380	0	-240	-240	160
21300	18030	14520	20312	18942	200	120	380	0	-620	200	-300
21300	18030	14520	20312	18942	210	120	380	0	1200	1200	-1100
21300	18030	14520	20312	18942	220	120	460	120	1400	1400	1680
21300	18030	14520	20312	18942	224	120	390	180	-1600	100	200
3800	18030	14520	20312	18942	220	110	410	240	-160	120	60
3700	18030	14520	20312	18942	220	80	140	10	-20	-20	-20
3600	18030	14520	20312	18942	220	170	180	80	0	120	220
18000	18030	14520	20312	18942	220	180	320	0	10	100	-80
18882	17376	14526	18960	16540	180	240	310	120	120	160	116
10750	12220	16080	1480	600	240	220	150	80	120	76	20

TABLE A.3 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the Max-SAT problem.

LS7	LS8	MU0	MU1	MU2	MU3	MU4	MU5	RR6	CO9	CO10
322	125	-100	-225	-100	-100	100	7	9	10	0
120	121	100	-120	100	100	70	-21	19	0	1
324	126	-200	240	-200	-200	-99	17	-9	0	0
120	122	99	99	99	99	120	14	-19	0	0
122	325	100	101	100	100	110	12	39	0	0
126	225	70	76	70	70	331	-45	46	30	0
126	300	-99	131	-99	-99	334	11	-110	-10	0
122	124	120	-111	120	120	113	-16	-110	10	128
122	122	110	110	110	110	114	144	-110	200	333
120	422	331	331	331	331	-116	126	-110	200	254
110	125	334	336	334	334	114	334	-110	100	226
327	125	113	113	113	113	56	122	-78	0	10
125	624	114	114	118	118	66	-111	19	0	0
424	125	-116	-116	11	11	17	-121	36	0	0
122	125	-99	-99	119	76	-22	100	72	0	0
124	125	113	-77	-133	131	-48	70	-189	0	0
122	125	114	114	6	-111	-100	-99	-128	0	30
426	625	177	-28	6	110	100	120	19	0	-10
122	125	7	7	6	331	-200	110	188	0	10
422	125	231	231	232	336	99	331	-142	0	200
122	125	-19	-19	-71	113	100	334	-76	256	200
122	125	7	7	88	114	70	-65	-72	457	-458
422	126	-21	-21	-26	-116	-99	-110	-19	-520	-120
122	122	17	17	117	-99	120	125	-9	-600	220
122	522	14	14	114	-77	110	-422	-26	320	-320
426	224	12	12	120	7	331	226	19	420	-160
122	226	5	-45	8	3	334	12	-50	-623	-122
522	525	1	11	1	79	113	6	9	-120	0
122	125	-18	-16	14	111	118	-8	45	110	0
426	224	113	99	133	113	331	144	126	100	128

TABLE A.4 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the permutation flow shop problem.

LS7	LS8	LS9	LS10	CO11	CO12	CO13	CO14	MU0	MU1	MU2	MU3	MU4	RR5	RR6
132	430	88	120	170	125	110	224	-20	140	280	250	386	-350	550
245	20	430	420	170	288	288	224	220	88	280	386	386	-350	420
222	65	220	320	-25	270	170	120	160	176	280	386	386	-350	380
222	24	430	222	170	170	170	170	180	114	280	386	386	-350	570
222	430	220	280	200	310	71	76	200	142	280	386	386	-350	480
222	270	430	280	200	31	152	44	220	147	280	386	386	-350	290
222	270	290	288	140	220	420	71	240	136	280	386	386	342	170
222	270	270	288	288	188	120	74	260	-12	154	386	386	342	290
222	270	270	128	-70	188	145	79	140	135	156	386	386	342	480
222	270	270	280	390	188	160	165	-80	187	157	386	164	342	380
222	270	488	420	-132	188	170	142	226	210	164	386	162	342	342
222	270	488	88	135	188	220	130	189	113	162	386	163	163	165
222	270	488	45	134	188	176	152	220	93	163	386	280	189	250
222	270	280	45	-45	188	125	74	189	280	280	386	289	280	280
222	270	280	45	300	188	125	120	226	280	289	250	299	280	244
222	270	620	45	253	267	288	147	220	280	299	250	321	280	127
222	88	320	45	-194	244	270	420	160	280	321	140	280	280	410
222	164	640	488	208	274	170	120	240	280	342	270	280	280	380
222	145	640	140	220	146	310	145	189	280	314	470	280	280	228
222	222	640	280	-15	124	31	160	189	280	245	425	290	280	162
222	280	280	100	120	227	220	170	-85	290	245	469	290	280	470
222	280	428	242	324	410	188	220	240	290	135	386	480	280	580
222	288	210	288	267	382	188	176	189	480	187	-124	260	280	120
222	288	100	421	276	374	188	125	189	-52	210	-12	240	410	390
222	128	200	136	233	374	220	125	260	480	113	389	220	440	270
222	280	210	270	178	362	220	288	240	480	93	386	-112	460	222
222	120	122	270	135	342	220	270	220	136	187	386	-362	380	430
450	100	120	270	220	310	220	170	189	147	178	386	117	420	440
240	100	120	480	228	170	220	224	220	136	139	389	387	560	220
240	120	110	320	228	170	220	224	220	147	342	469	386	280	162

TABLE A.5 Results obtained after 30 seconds for each low level heuristic for the one dimensional bin packing problem, one instance.

LS1	LS2	M1	M2	M3	R1	R2	CO1
0.1	0.12	0.02	0.12	0.47	-0.11	0.09	0.01
0.02	0.4	0.03	0.08	0.66	-0.15	0.09	0.02
0.01	0.41	0.16	0.1	0.34	0.01	0.11	0.3
0.03	0.03	0.2	0.3	0.46	0.01	0.16	0.1
0.12	0.33	0.21	0.14	0.25	0.1	0.2	0.05
0.11	0.24	0.4	0.06	0.78	-0.1	0.03	0.16
0.1	0.48	0.08	0.23	0.41	0.13	0.2	0.16
0.12	0.05	0.02	0.17	0.11	0.04	0.23	0.16
0.12	0.11	0.01	0.01	0.29	0.06	0.29	0.16
0.12	0.14	0.3	0.2	0.31	0.01	0.15	0.16
0.12	0.25	0.06	0.28	0.46	0.01	0.16	0.02
0.12	0.1	0.23	0.28	0.42	0.14	0.1	0.03
0.12	0.1	0.1	0.01	0.42	0.14	0.2	0.03
0.12	0.2	0.4	0.28	0.06	0.01	0.01	0.03
0.12	0.3	0.3	0.28	0.41	0.02	0.23	0.03
0.12	0.4	0.28	0.28	0.43	0.01	-0.1	0.03
0.04	0.011	0.28	0.28	0.44	0.06	0.23	0.03
0.03	0.1	0.01	0.1	0.7	0.05	0.24	0.03
0.02	0.11	0.28	0.07	0.11	0.08	0.14	0.03
0.06	0.23	0.2	0.14	0.22	0.1	0.26	0.03
0.07	0.23	0.09	0.01	0.06	0.11	0.3	0.03
0.1	0.23	0.3	0.25	0.16	0.09	0.14	0.03
0.1	0.23	0.28	0.28	0.12	0.03	0.17	0.16
0.1	0.036	0.28	0.22	0.14	0.06	0.7	0.14
0.12	0.42	0.28	0.11	0.16	0.1	0.11	0.11
0.12	0.4	0.32	0.17	0.3	0.1	0.11	0.09
0.12	0.1	0.33	0.05	0.3	0.01	0.16	0.14
0.12	0.16	0.01	0.16	0.4	0.02	0.13	0.06
0.1	0.48	0.02	0.11	0.04	0.02	0.29	0.04
0.1	0.03	0.32	0.12	0.13	0.03	0.21	0.07

TABLE A.6 Results obtained after 30 seconds for each low level heuristic for the nurse rostering problem, one instance.

LS1	LS2	LS3	LS4	LS5	CO1	CO2	CO3	M1	RR1	RR2	RR3
4200	12220	4540	1480	600	20	96	12	80	14	14	14
1055	3347	12080	1420	240	40	40	88	17	120	76	16
1055	3347	16080	1120	29	14	12	76	11	0	0	0
1055	3347	4440	1000	32	14	4	0	2	-120	74	12
1055	3347	3800	380	32	220	210	140	0	-60	-60	-1
1055	3347	3700	290	32	220	65	150	0	18	18	2
1055	3347	2400	29	32	220	38	12	0	45	45	3
1055	3660	2200	32	32	220	120	100	3	-87	-87	20
1055	3776	2200	32	32	220	120	120	0	78	62	12
1055	3776	2200	32	32	220	120	150	6	0	0	0
1055	3776	2200	32	4	220	120	150	0	0	0	0
1055	3776	2200	32	12	220	100	90	1	46	64	1
1055	3776	2200	32	320	220	120	10	4	0	0	0
1055	3776	2200	32	240	220	90	20	0	60	60	1
1055	3776	2200	4	260	220	120	42	0	28	28	8
245	3776	600	8	180	220	120	46	0	0	0	0
620	3776	380	1200	160	220	120	12	0	-16	-16	6
620	3776	380	1200	120	140	120	12	0	2	2	2
620	3776	380	1200	100	240	120	38	0	0	0	0
620	10040	380	1200	80	220	120	66	0	-10	-10	-4
3600	4200	380	600	120	80	120	75	0	-11	-11	-11
2400	4200	380	120	170	200	120	74	0	-22	-22	14
3600	4200	380	240	340	210	120	76	0	12	12	12
9988	4200	380	32	420	220	120	88	12	14	14	14
10750	4200	380	18	560	224	120	39	11	-11	-11	-11
3800	4200	220	22	480	220	110	43	24	-13	-13	-13
3700	4200	120	124	320	220	80	110	80	-20	-20	-20
3600	4200	420	120	180	220	220	120	-12	0	0	0
3600	4200	540	360	18	220	18	146	13	10	10	10
3600	4200	540	360	26	180	24	34	11	20	20	20

TABLE A.7 Results obtained after 30 seconds for each low level heuristic for the Max-SAT, one instance.

LS1	LS2	M1	M2	M3	M4	M5	M6	R1	CO1	CO2
2	2	2	2	2	1	1	1	1	2	0
4	4	4	4	2	1	1	1	1	0	1
3	3	3	3	2	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	2	1	0	0
1	1	1	1	1	1	1	2	1	3	0
61	16	11	7	1	1	0	2	1	-1	0
20	20	2	-2	-2	1	1	2	1	6	3
16	16	11	-4	-4	1	1	2	1	0	-1
2	2	2	2	2	1	-2	2	1	2	-1
13	13	12	-1	-1	1	1	2	1	3	0
78	14	12	1	1	1	1	2	1	0	0
62	33	12	1	1	1	-2	2	1	0	0
14	14	12	1	1	1	1	2	1	0	0
22	22	12	1	1	1	1	2	-10	0	0
22	22	12	0	0	1	1	2	1	0	2
22	26	12	-1	-1	-4	-4	2	1	0	0
22	22	1	1	1	-3	-3	2	-3	0	0
22	22	4	-7	1	0	0	2	1	0	0
22	22	5	-1	-1	1	1	2	1	0	0
13	13	3	-19	2	1	1	2	-2	1	0
11	11	11	7	1	1	1	1	1	4	0
21	21	11	-2	-2	-1	-1	-1	1	5	0
14	14	9	8	0	-6	-6	-6	1	2	0
18	18	8	4	0	1	1	1	1	2	0
32	32	2	2	2	1	1	1	2	3	0
66	42	6	5	1	1	0	0	1	4	0
1	1	1	1	1	1	1	1	2	5	0
1	1	1	1	1	1	0	0	2	6	3
2	2	2	2	2	1	1	1	-4	6	3

TABLE A.8 Results obtained after 30 seconds for each low level heuristic for the permutation flow shop, one instance.

LS 1	LS 2	LS 3	LS 4	CO 1	CO 2	CO 3	CO 4	M1	M2	M3	M4	M5	R1	R2
122	100	88	120	170	125	110	15	-20	-20	14	1	1	1	100
120	20	30	120	170	140	120	25	15	15	11	1	1	1	90
10	65	100	120	-25	27	27	32	12	12	12	1	1	1	80
12	24	27	120	170	17	27	40	14	14	14	1	1	1	55
28	30	26	120	180	31	31	76	26	26	6	1	1	1	48
43	100	26	120	180	31	31	44	44	44	4	1	1	6	29
27	27	26	120	140	46	46	71	11	11	11	1	1	1	17
12	26	26	120	10	88	88	74	14	14	14	3	1	1	29
10	26	26	118	-70	88	88	79	19	19	9	3	1	4	48
10	26	26	28	39	88	48	90	9	9	9	3	1	1	48
10	26	26	42	-13	88	88	12	12	12	12	3	1	1	48
10	26	26	88	13	88	32	14	14	14	14	3	1	1	48
10	26	26	45	134	88	88	15	15	15	5	4	1	1	48
120	26	88	45	-45	88	88	7	7	7	7	5	1	7	48
27	26	16	45	30	88	88	12	12	12	12	8	1	1	48
110	26	14	45	24	47	47	14	-4	-4	2	10	1	1	48
111	88	22	45	-194	47	47	42	-6	-6	1	1	1	1	48
122	16	28	85	20	74	74	12	-1	-1	3	3	5	5	48
100	14	32	75	170	12	12	14	1	1	1	3	3	3	48
90	22	32	36	-15	12	12	16	1	1	11	2	-4	-4	48
88	28	32	100	120	12	64	17	1	1	1	2	2	2	48
77	32	32	112	124	12	12	22	1	1	10	-4	4	4	11
1	32	32	28	67	12	12	6	1	1	1	2	3	3	12
17	32	32	42	74	12	12	6	1	1	1	2	5	5	39
22	32	224	116	33	12	36	6	1	1	1	2	-1	-1	6
120	32	110	27	18	12	12	6	1	1	1	2	-1	-1	3
14	32	80	24	12	8	8	6	1	1	1	2	-1	-1	1
16	32	110	22	110	7	7	6	1	1	1	2	-1	-1	1
122	100	120	48	120	6	6	56	-2	-2	12	2	1	1	1
122	90	110	32	80	2	2	64	-3	-3	13	2	1	1	1

TABLE A.9 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the one dimensional bin packing problem, using a better initial solution.

LS1	LS2	M1	M2	M3	R1	R2	CO1
0.1	0.12	0.02	0.4	0.47	-0.11	0.09	0.01
0.1	0.4	0.03	0.08	0.66	0.88	0.41	0.02
0.02	0.62	0.16	0.9	0.34	0.2	0.11	0.3
0.3	0.26	0.2	0.4	0.46	0.01	0.16	0.1
0.4	0.33	0.21	0.6	0.25	0.23	0.6	0.05
0.011	0.24	0.4	0.056	0.78	-0.1	0.5	0.16
0.1	0.48	0.08	0.23	0.41	0.23	0.4	0.2
0.12	0.11	0.9	0.3	0.11	0.24	0.23	0.2
0.23	0.21	0.4	0.4	0.29	0.14	0.41	0.2
0.23	0.14	0.6	0.4	0.31	0.26	0.55	0.2
0.23	0.76	0.056	0.28	0.46	0.3	0.6	0.2
0.23	0.1	0.23	0.28	0.42	0.14	0.1	0.2
0.23	0.1	0.3	0.28	0.42	0.17	0.2	0.2
0.23	0.02	0.4	0.28	0.74	0.7	0.01	0.2
0.23	0.3	0.4	0.28	0.41	0.41	0.23	0.2
0.23	0.4	0.28	0.28	0.43	0.31	-0.1	0.2
0.23	0.011	0.28	0.28	0.44	0.46	0.23	0.2
0.23	0.1	0.28	0.7	0.76	0.25	0.24	0.2
0.23	0.12	0.28	0.66	0.89	0.78	0.14	0.2
0.36	0.23	0.28	0.34	0.22	0.41	0.26	0.2
0.36	0.23	0.28	0.46	0.74	0.11	0.3	0.2
0.36	0.23	0.28	0.25	0.86	0.29	0.14	0.2
0.36	0.23	0.28	0.78	0.9	0.31	0.17	0.16
0.36	0.36	0.28	0.41	0.86	0.46	0.7	0.34
0.4	0.42	0.28	0.11	0.86	0.32	0.41	0.31
0.28	0.57	0.32	0.29	0.86	0.31	0.31	0.09
0.63	0.89	0.33	0.31	0.86	0.11	0.86	0.17
0.76	0.94	0.45	0.46	0.86	0.21	0.31	0.26
0.12	0.22	0.36	0.42	0.73	0.22	0.29	0.24
0.32	0.76	0.42	0.42	0.73	0.24	0.21	0.27

TABLE A.10 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the nurse rostering problem, using a better initial solution.

LS1	LS2	LS3	LS4	LS5	CO1	CO2	CO3	M1	RR1	RR2	RR3
7000	10036	13250	12060	11307	20	96	12	170	140	145	21
6500	10036	12890	13450	10037	40	40	88	40	120	76	20
7060	10036	13240	1100	10090	144	12	76	12	0	0	0
7060	10036	13220	1000	9970	41	4	41	10	-120	-20	74
7060	10036	12880	2200	9970	220	6	140	0	-60	-60	40
7060	10036	12740	2200	9970	170	450	450	0	180	90	12
7060	10036	12980	2200	9970	220	180	12	0	150	50	-56
7060	10036	12989	9961	9970	190	120	70	12	120	-100	64
7060	10012	12456	10961	9970	220	90	120	0	90	110	-70
7060	9990	12450	10961	9970	190	90	60	36	50	0	0
7060	9890	11800	10961	10020	220	90	80	0	0	0	80
7060	9896	11800	10961	10060	220	90	90	40	-160	110	-52
7060	10036	11800	10961	9450	120	90	80	41	0	0	0
7060	11025	11800	10961	6100	80	90	70	0	-76	120	-16
7060	10041	11800	10961	7000	110	90	120	0	-120	-80	80
7060	10740	11800	10961	2200	220	90	160	0	0	0	100
7060	9987	11800	10961	2200	80	90	120	0	140	140	460
7060	9874	11800	10960	9960	260	90	114	0	120	120	1200
7060	9654	11800	11000	9961	324	90	180	0	0	0	120
7060	9541	11800	9160	10961	180	90	180	0	-120	-120	50
7060	9756	12520	9160	10961	210	90	220	0	80	-60	-100
7060	9890	12540	9160	10961	60	90	220	0	0	180	110
6400	9897	12570	9160	10942	210	120	220	0	120	150	0
6200	9889	13010	9160	10942	70	60	220	14	110	120	0
6700	10032	13020	9160	10942	224	120	190	45	120	90	110
6700	10034	12740	9160	10942	220	70	110	17	90	50	0
6800	10022	12880	10312	10942	120	80	140	12	20	0	84
6800	9752	12520	9980	9960	80	170	180	65	0	-160	-80
6800	9890	12570	9970	9980	90	180	120	0	10	90	0
6800	9940	11940	9980	9970	180	140	90	23	110	70	140

TABLE A.11 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the Max-SAT problem, using a better initial solution.

LS1	LS2	M1	M2	M3	M4	M5	M6	R1	CO1	CO2
78	42	12	8	2	1	1	2	1	2	0
120	42	12	8	1	-3	4	1	2	4	1
84	42	12	8	2	1	5	-1	5	3	0
80	42	12	8	1	1	3	3	4	4	3
86	42	12	8	-4	-2	10	4	-2	3	3
30	42	12	8	-4	1	-13	1	-3	2	3
26	42	12	8	-3	1	-14	-3	-5	2	3
72	42	-10	8	2	0	11	-4	1	-1	1
62	42	-10	8	0	0	10	-2	1	-3	2
86	42	8	8	0	0	1	-1	1	2	1
30	42	5	8	1	0	2	1	1	3	1
26	42	7	8	2	0	-1	1	1	0	0
72	42	8	8	2	0	-5	1	1	0	1
62	42	2	8	2	0	-4	1	1	6	2
62	42	1	8	2	0	0	-2	1	0	2
62	42	1	8	2	0	-2	1	1	0	2
62	42	1	6	2	0	3	-2	1	3	2
62	42	1	2	2	0	1	-2	1	0	-1
62	42	1	-1	2	0	1	1	1	0	2
22	42	1	-1	2	1	1	-2	1	2	3
74	42	1	-2	2	1	1	-2	1	6	3
26	42	1	3	2	1	3	-2	1	5	3
32	26	5	4	2	1	1	-2	1	3	2
46	28	3	6	2	1	1	1	1	6	3
50	34	2	2	2	1	1	1	-2	-1	3
42	36	2	-2	2	1	1	-2	-3	2	2
38	38	1	-4	2	1	1	-2	-4	4	3
36	42	1	2	2	-3	1	-3	2	3	2
32	42	-1	1	2	-2	1	-8	3	2	1
32	42	-2	6	2	1	1	144	2	1	3

TABLE A.12 Results over 31 runs of 1000 iterations for each low level heuristic for one instance of the permutation flow shop problem, using a better initial solution.

LS 1	LS 2	LS 3	LS 4	CO 1	CO 2	CO 3	CO 4	M1	M2	M3	M4	M5	R1	R2
122	100	88	80	110	110	110	88	-20	23	14	10	5	-9	55
112	100	224	75	110	110	110	84	16	21	14	10	4	-16	-20
116	100	220	112	120	120	120	82	12	17	14	10	6	-20	38
118	100	120	112	120	120	120	88	10	11	14	10	3	-35	57
120	100	130	80	120	120	120	76	11	14	14	10	2	-36	48
108	100	90	66	180	140	104	44	19	17	14	10	1	-80	29
90	100	110	77	160	140	109	71	240	16	14	10	1	5	17
114	100	100	74	176	96	96	74	-8	13	11	10	1	5	9
113	100	210	73	-90	80	80	79	14	15	9	10	1	5	8
110	100	142	72	170	170	110	75	5	18	11	10	1	5	20
110	65	143	57	110	110	110	34	17	21	6	7	1	5	11
106	42	188	67	135	125	95	30	20	23	2	10	1	5	12
122	88	82	66	134	124	92	52	21	23	1	10	1	5	14
84	74	94	63	-45	75	75	74	11	17	8	10	1	5	28
76	66	75	78	140	140	87	88	13	4	9	10	1	4	24
64	66	110	74	133	111	46	90	9	3	8	10	1	4	100
62	88	114	25	92	92	67	90	10	18	6	10	1	6	41
56	100	113	45	74	73	68	39	22	6	9	10	5	4	14
122	90	124	26	88	89	39	47	25	10	8	8	6	4	28
122	98	220	28	25	25	39	76	19	12	5	6	3	7	12
122	78	216	22	-14	12	25	79	15	15	11	6	4	3	16
70	45	218	110	-23	32	12	82	24	19	9	8	2	5	12
88	60	224	116	67	76	76	82	19	21	13	6	2	3	10
90	80	222	120	76	68	68	84	26	23	14	-12	2	3	39
92	88	223	90	49	49	49	71	21	22	5	3	2	3	27
58	90	210	90	85	85	85	82	20	18	12	4	2	3	12
48	100	180	74	57	79	79	82	20	15	13	6	1	2	9
86	100	110	38	46	81	81	82	18	17	14	-4	-8	7	5
88	100	80	48	180	120	120	90	19	18	9	10	2	8	14
122	90	70	110	156	106	106	29	26	18	10	8	1	7	62

Appendix B

Healthcare Scheduling and Methodologies

Appendix B includes a more extensive review of work done on healthcare scheduling i.e. nurse rostering and surgery scheduling problems. Also included is a review of general methodologies. Some publications mentioned in chapter 2 or 3 are also included in Appendix B as more detailed information is provided.

It is worthwhile mentioning in this research the problem of tour scheduling. Tour scheduling is a personnel scheduling problem in which each day is broken into small time units and each unit is assigned a task, these tasks will eventually be allocated to an individual. The time units include productive tasks and rest tasks. This publication is relevant as hospitals often do not actually work in a shift based system but need more flexibility to allow for nurses to start and end their day at different times.

Alfares (2004) presents briefly the main models used to express tour scheduling problems and methods used to solve the problem. The tour scheduling problem consists of assigning a schedule or tour for a given period to an employee. The tour consists of the starting time of work, includes off periods, off days until ending time of work for the planning period. Tour scheduling differs from shifts as the timeframe is for the planning period and the start and end times of work may vary from person to person and day to day. Generally the mathematical models aim to minimise cost of the workforce or to minimise the workforce while ensuring personnel requirements are met. The author classifies the papers based on the methodology used to solve the problem.

This paper is interesting as it provides a brief overview of the studies done on tour scheduling and classifies these according to the methodology used to

solve the problem; comparisons are also made between the different methods used

The following publications have been grouped by the general methodology or representation used to solve the nurse rostering problem. Section B.1.1 encompasses research done using single point search methods. Section B.1.2 contains publication that decompose the complex problem into subcomponents and section B.1.3 includes models that define the problem differently. Section B.1.4 includes general methods used on non-healthcare scheduling problems.

B.1.1 Single Point Search Algorithms

Berrada et al. (1996) represent the nurse rostering as a multi-objective model where each constraint is an objective/goal to be minimised. The authors propose three different solution methods, these affect the model representation. The sequential technique which solves all constraints, the weights technique which focuses on a linear objective function and a tabu search which will be included only on two hard constraints and the other constraints will be presented as non-linear objective functions. All three methods are tested using real data for three wards of a large hospital. The computational time are also compared. All three solution methods/models yield a feasible solution and respect soft constraints. The computational time is less for the weights and sequential techniques than the tabu search.

This research tests the models/techniques on real data; the multi-objective model is flexible as it enables adding, removing and changing soft constraints to reflect time period or requirements of different wards/hospitals. The model is also easily understandable for a head nurse. The initial model is limited as it only uses three shifts per day and each nurse is scheduled to work the same shift throughout the planning period. A model that would allow flexibility with shift starting time and length of shift as well as allowing different shifts for each nurse through the period would represent the problem even more realistically.

Schaerf and Meisels (1999) review a general employee scheduling model. The problem considered is to assign an employee to a task in a shift. Shifts can

start and end at any time in the day. The model considers work requirements, personnel availability, qualifications, workload restrictions i.e. a minimum and maximum number of tasks an employee can perform. The objective is to minimise the violation of constraints. The model covers a week period. An initial solution is found by inserting employees in a schedule that minimises constraint violations. The initial solution respects work requirements. A hill climbing algorithm is used to improve the solution. The moves considered are moves that improve or have no impact on the objective function. The moves chosen are to replace, insert or delete employees in a schedule. The methodology permits the creation of partial assignments. The method was tested on two theoretical employee scheduling problems: a nurse rostering problem and a scheduling on a production line in a factory.

The model used permits flexible start and end times for shifts. The hill climbing method explores a large solution space by allowing moves that provide a better solution or sideway moves. The method also allows partial schedules which can be useful in real world scheduling problems. However the method was not tested on real data; it would be interesting to apply the model and methodology to a real world problem. The method requires adjustment for each problem studied.

Li and Aickelin (2004) propose a different way of representing and solving the nurse scheduling problem. The idea is to build the nurses' schedules based on four rules that are used by human schedulers, the rules introduce randomness, consider only cost, consider coverage and grade requirements in the scheduling process. The authors present the model used in previous work in a graph where each node represents a nurse and the rule that is used to schedule the nurse. Each nurse has one node for each rule. Initially the edges represent the number of times a path is used randomly to construct a schedule in a number of runs. The probability of using each specific path is calculated and new paths are selected according to their fitness and added to the set of possible paths for the scheduling. The authors also apply a second algorithm, an adapted classifier system, to solve the same instances of the nurse scheduling problem. This algorithm uses the same modelling of the problem. Initially

each node is assigned randomly a score, which the authors call strength. New schedules are built according to these strengths i.e. the nodes chosen have the highest scores. The new schedules are evaluated and if they are better than the previous schedules they are rewarded, if they are not they are penalized. The reward/penalty is attributed evenly between the nodes of the solution. At each iteration the solution is kept.

This paper is relevant, it presents the nurse rostering problem differently and the algorithms used are tested on real data.

Bard and Purnomo (2005) model the nurse rostering problem as an integer programming problem and present an algorithm to solve the issue using various size problems i.e. variable number of nurses and planning period. The model takes into account floating nurses and the cost of the extra resources. The method proposed also enables the planner to apply a degree of severity for not respecting soft constraints which will be taken in account when solving the problem. From an initial solution over coverage and under coverage periods are found for each nurse and swapping is done from shifts that are over covered to under covered to create new schedules. The heuristic checks for feasibility of the schedules, ensures that there is no duplicity of schedule and calculates cost.

This paper provides a more complete model of the nurse rostering problem considering different skill sets, previous schedules, attributing penalties/cost evenly between nurses and including floating personnel in the model. The results are also analysed to demonstrate which parameters render the problem difficult for example when the planning horizon is increased.

Aickelin and Li (2007) reflect and solve the nurse scheduling problem in a new way. The idea is to build the nurses' schedules based on four rules that are used by human schedulers, the rules introduce randomness, consider only cost, consider coverage and grade requirements in the scheduling process. The authors present the model used in previous work Aickelin and Dowsland (2000) in a graph where each node represents a nurse and the rule that is used to schedule the nurse. Each nurse has one node for each rule. Initially the edges represent the number of times a path is used randomly to construct a

schedule in a number of runs. The probability of using each specific path is calculated and new paths are selected according to their fitness and added to the set of possible paths for the scheduling.

An important contribution of this paper is to reflect the nurse scheduling problem in a directed graph which permits the incorporation of rules used by human scheduler to construct a schedule. This methodology is flexible as it is possible to add rules to create schedules. This also was tested on real data.

B.1.2 Problem Decomposition

As the nurse rostering problem is complex, some publications decompose the problem into more manageable sub-problems. A category for this type of research is made to highlight this modelling possibility.

Azaiez and Al Sharif (2005) propose a goal programming model to reflect the nurse rostering problem. The authors have conducted a study of nurses' preferences by a survey and have incorporated hospital/ward requirements in the goal programming model. The model is first expressed in hard and soft constraints and is then converted so that each goal represents a constraint; weights are added according to the importance of the goal. To solve the problem, the authors divided the nurses in subgroups and developed a program to obtain optimal schedules. The study found that overtime cost was reduced and a second survey showed that nurses' satisfaction was good.

This paper is helpful; the study of a real hospital environment was done and the model takes into consideration nurses' preferences. The program is also being used on a trial basis by the hospital. The program is user friendly and enables head nurses to do some modifications to the system to respect some preferences, it allows flexibility. The method proposed could have been presented in more details.

Aickelin et al. (2009) present an algorithm to solve 2 scheduling problems. The algorithm used is a Squeaky Wheel Optimisation method with added elements. The proposed algorithm Evolutionary Squeaky Wheel Optimisation uses an initial solution that it divides into components. These components are

then analysed i.e. evaluation of each component and attribution of a fitness (score) based on parameters of the problem at hand (such as coverage). Based on their fitness score components are then compared to a random number to evaluate if they will remain in the solution or be discarded. From the remaining components a further random small number will also be discarded. The discarded shifts (components) are evaluated (based on the previous analysis) and ordered. The hardest shift to schedule is the first in the ordered list. The last step consists of assigning the discarded shifts to cover all personnel requirements. The method was defined and tested on a driver scheduling problem and a nurse rostering problem. In both cases the new method provides results comparable or better to previously used algorithms.

This paper is interesting because by dividing the initial solution into subsets (components), evaluating these and setting priorities, the problematic shifts are highlighted and can be dealt with at the beginning of solution construction. The method is applied to two different problems. The method is general although it does not have the level of generality of hyper-heuristics because the parameters are taken into account and dealt with directly but nonetheless the method was adapted easily to each problem.

Burke and al. (2009f) use a combination of integer programming and a variable neighbourhood search algorithms to solve a nurse rostering problem. The authors define a model to solve nurse rostering as an integer programming problem and divide the problem into sub problems. The first sub problem is used to model the hard constraints and the important soft constraints. This sub problem is solved by an Integer Programming algorithm but not to optimality. The other sub problem contains the soft constraints not dealt with in the first sub problem. The variable neighbourhood search is employed to solve this sub problem. The results are comparable or better than other meta-heuristics used to solve the same problem.

This paper is relevant; it decomposes the complex problem of nurse rostering and models it in all its complexities. The integer programming algorithm is used intelligently i.e. not to find optimality as it is too time consuming but in obtaining a feasible solution. The combination of heuristic and exact

algorithm is an important contribution and represents a new way of defining and solving the complex nurse rostering problem.

Brucker et al. (2010) decompose a nurse rostering problem into subsets which will be solved separately. The second objective of the paper is to present and use new real data on nurse rostering. The subsets are a shift sequence problem, a schedule problem and a roster problem. Constraints are attributed to each subset. The shift sequencing consists of defining for each nurse of each qualification a set of all possible sequence of shifts that respect the hard constraints and have the lowest penalties in regards to soft constraints. These shift sequences are kept and ordered for each nurse. The second problem consists of assigning a schedule to each nurse using the best shift sequences previously defined. The nurses that have the most schedule restrictions will have their schedule created first. The third step consists of gathering the schedules to create a roster; the roster is evaluated to ensure respects of requirements. If roster constraints are not met, the roster is modified by swapping nurse schedules. A further evaluation is done and more changes are made to the roster to ensure respect of constraints or obtain better solutions.

An important contribution of this research is the presentation and use of new data sets. These can be used to evaluate other solution methods for the nurse rostering problem. The other important contribution is the problem formulation into 3 subsets each with their own constraints. Most research is concerned with the solution method, the different representation of the problem is as important to gain a better solution. Although the methods used to solve the problems are not hyper-heuristics, the methods are general.

B.1.3 Unique Modelling Approaches

This section groups publications that define the nurse rostering problem differently such as including float nurses in the model or address an issue that is part of the reality of nurse scheduling in hospitals such re-rostering due absenteeism.

Trivedi and Warner (1976) represent the problematic of assigning float nurses on a daily basis to units in a hospital. The assignments must be made based on

information of the day such as current available staff for each unit, the patients' requirements in all units considered and available float nurses. The authors use the knowledge of the head nurses of each unit to evaluate at the beginning of each shift the requirements to have float nurses assigned, the authors call this a "severity index". The authors build a model whose objective is to minimise the severity index i.e. to assign available float nurses to each unit in such a way as to reduce inconveniences during the shift. The model considers interdependence between available nurse's skill sets and the severity index. The model is solved using a branch and bound algorithm while ensuring equitable assignments of float nurses between units. The results were compared with the assignments of float nurses made by the hospital decision maker. The comparison was made for a four week period. The authors find that the model assigns float nurses similarly to the assignments made by the decision maker.

This methodology is flexible; the problem is a short term planning issue that varies from shift to shift. In order to evaluate the situation human knowledge is required; this is provided by the model using a severity index. As the model and method was tested on real data this could be used in a hospital to help for short term planning. The methodology could be further developed to generate the severity index based on history of assignments. The problematic has not been studied much although this is an important issue that arises in a hospital environment.

Brusco and Jacobs (1995) present and solve two models for personnel scheduling in a 24 hour seven days a week environment. The authors present a model where no shifts may overlap over a 24 hour period and a model that permits overlapping enabling shifts to start at different times of the day. The authors show the first model decreases the number of possible shift patterns yet the labour costs are higher. The authors also include in the model the possibility of using part time and full time workers. The resolution method proposed to obtain an acceptable computational time is a local search heuristic.

An initial feasible solution is found randomly, a set of employee schedules is taken away and rescheduled to improve solution.

This paper demonstrates that although the complexity of a schedule that does not permit overlapping over a 24 hour period is reduced as they are less possible patterns the cost of labour is higher. The second model that permits overlap provides more flexibility and the use of part time and full time workforce reflects better organizational demands.

Meyer auf'm Hofe (1997) defines the nurse rostering problem more completely i.e. where the constraints are defined and evaluated in terms of importance. The author defines every soft constraint and allocates to each an importance. The constraints are then placed in a hierarchy. The hard constraint is placed at the top of the hierarchy as it must be satisfied. This is done to ensure that the solution takes into account the importance of the constraints and satisfies the important constraints. This cannot be done by attributing weights only to the requirements. The model is further defined by attributing in each constraint a weight to each variable. The objective of the weight is to measure the penalty (cost) of not respecting this soft constraint for each nurse. The algorithm used to solve the model is a modified branch and bound algorithm. An initial solution is found. The schedules for each nurse are evaluated and the ones that do not respect most important constraints are further modified to find a better feasible solution.

The important contribution of the paper is the model that takes into account the importance of each requirement and builds a hierarchy based on this importance. This implies that a solution will not only satisfy requirements that are less important but rather first the most important constraints will be respected. This represents better the reality of a nurse rostering problem. When evaluating the schedules, the modifications made are logical and are easy steps. In the example provided 2 nurses are scheduled for one shift and another shift is empty. The algorithm simply assigns one shift to one of these nurses.

Meyer auf'm Hofe (2001) explains how the nurse rostering problem can be defined as a constraint optimization problem. The problematic consists of

assigning shifts to nurses while respecting several constraints. To define the problem the authors have created different hierarchal constraint levels i.e. all constraints in a level will reflect the importance of these constraints where zero is the most important level. As in each hierarchal level of constraints some constraints may be more important than others, each constraint within the level is assigned a weight, the higher the weight the more important the constraint. The methodology allows partial satisfaction of some constraints; this will depend on the requirements of the hospital. The model also keeps a tab on the nurses' past shift assignments to allow for future compensation i.e. extra days off or other shifts in future. The software described uses a branch and bound algorithm coupled with an iterative search algorithm to find good rosters. The software enables the scheduler to attribute weights and change the level of hierarchy of constraints to allow more flexibility in building schedule to permit consideration of nurses' preferences.

As this paper was published in 2001 it is interesting as it is the first to allow partial constraint satisfaction and reflects the reality of hard and soft constraints using hierarchy level of constraints and weight attribution for each constraint within each level. The software also enables the scheduler to define modifications to a good roster, to identify modifications that need to be made because some constraints are violated or to change priority level of constraints. These features provide flexibility for future requirements in nurse scheduling.

Moz and Pato (2004) study the re-rostering nurse problem. The problem arises when the roster has been established for a given period and one or more nurses must be absent during a period in which they were scheduled to work in the roster. The authors present two models to reflect the problem. The goal is to minimise the dissimilarities between the existing roster and the new roster. The first model presented is an acyclical network graph where each level represents the number days in the period and each node represents a shift assigned to a nurse for each day; morning, evening, night and fictitious shift (to represent days off). The path between nodes at each level represents the roster for the nurses. To minimise dissimilarity between the current schedule and the new schedule a cost is assigned to each possible pair of nodes i.e. nurses shift on day "d" to shift on day "d+1". The cost is attributed to satisfy

hard constraints where cost is high for coupling that cannot be done due hard constraints. The model does not take into account overstaffing. To reflect the reality of overstaffing an aggregated model is presented. The model aggregates node of the same level for each shift i.e. for each level 8 nodes are defined and represent the 4 shifts and a "shadow" shift of each shift. The link between a node and its shadow will ensure that staffing requirements are met for each shift of the day. This model has less variables, it is therefore less complex. A cost for linking nodes is also assigned to respect hard constraints. The authors find that the computational time is less for the aggregated model and the results are better.

This paper tackles the re-rostering problem which has not been studied much but represents a challenge in hospital staffing. The models were tested using real data of various sizes i.e. one ward of 19 nurses and one of 32 nurses. However as the authors point out only hard constraints are considered. The model would be of more interest and could be implemented in hospitals by integrating soft constraints to the re-rostering problem.

B.1.4 General Methods

Ouelhadj and Petrovic (2008) present a new hyper-heuristic methodology that is tested on flow shop problems. The authors define a hyper-heuristic that will coordinate the use of low level heuristics agents. The hyper-heuristic which the authors name hyper-heuristic agent (HHA) will select and keep the best or not worsening solutions and the low level heuristic agents that have given these results. The role of the low level heuristic agent (LLHA) is to use different low level heuristic on a starting solution to find a better solution. Each low level heuristic agents use the same low level heuristics to explore the search area of the current solution. The HHA selects which LLHA will search the solution space; the selected LLHA searches the solution space and gives the best found solution back to the HHA. The HHA decides to keep this solution or not. The HHA also decides which LLHA to use for the next iteration based on stored results. The HHA chosen were a greedy algorithm and a tabu search algorithm. The results show that the tabu search as an HHA performs better.

The research done on hyper-heuristic increases the level of generality of a solution method to resolve problems; the hyper-heuristic framework is not using any problem related data. This method however needs to be tested further on larger problems and other scheduling problems as pointed out by the authors.

Qu et al. (2009) present a new hyper-heuristic methodology. This consists of evaluating low level heuristic sequences and creating a hyper heuristic that mimics the creation of the best sequences i.e. the algorithm is adaptive. In other publications hyper heuristics are clearly defined i.e. tabu search or genetic algorithm, in this case the authors choose to create a hyper heuristic based on the evaluation of the sequence of low-level heuristics that gave the best results on the exam timetabling problem and the graph colouring problem. It is evaluated that the LWD (largest weighted degree) heuristic and the SD (saturation degree) give the best results as low level heuristics for both problems. It is also proven that the LWD is used and modified at the beginning of the creation of the solution. Using this information the authors develop an algorithm that adapts the LWD in the sequence of heuristics to construct the solution. Sequences are evaluated and if the solution needs to be improved the LWD heuristic is given a parameter to permit further hybridisation. The goal is to observe and analyse results obtained randomly using graph heuristics to create sequences of these heuristics to find feasible solutions. Once the analysis is done the authors developed an algorithm to mimic the choosing of the sequence of the heuristics and their modifications (hybridisations).

The interesting contribution of this research is the development of a method to choose a sequence of heuristics and adapt these automatically that reflects previous analysis of random sequences.

Bibliography

Acuña A., V. Parada and G. Gatica. (2011) Cross-domain Heuristic Search Challenge:GISS Algorihtm presentation.CheSC 2011.

Aickelin U., E. K. Burke and J. Li. (2009). *An Evolutionary Squeaky Wheel Optimisation Approach to Personnel Scheduling.* IEEE Transactions on Evolutionary Computation, 13: pp. 433-443, 2009.

Aickelin U., E. K. Burke and J. Li. (2007). *An Estimation of Distribution Algorithm with Intelligent Local Search for Rule-based Nurse Rostering.* Journal of the Operational Research Society, 2007. **58**(12): pp. 1574-1585.

Aickelin U. and K. Dowsland. (2000). *Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem.* Journal of Scheduling, 2000. **3**: pp. 139-153.

Aickelin U. and J. Li. (2007). *An Estimation of Distribution Algorithm for Nurse Scheduling.* Annals of Operations Research, 2007. **155**(1):pp. 289-309.

Aickelin U. and P. White. (2004). *Building Better Nurse Scheduling Algorithms.* Annals of Operations Research, 128: pp. 159-177, 2004.

Alfares K.H. (2004). *Survey, Categorization and Comparison of Recent Tour Scheduling Literature.* Annals of Operations Research, 2004. **127**: pp.145-175.

Azaiez M.N. and S.S. Al Sharif. (2005). *A 0-1 goal programming model for nurse scheduling.* Computers & Operations Research, 2005. **32**: pp. 491-507.

Bai R., E.K. Burke, G. Kendall, J. Li and B. McCollum. (2010). *A Hybrid Evolutionary Approach to the Nurse Rostering Problem.* IEEE Transactions on Evolutionary Computation, 2010. 14(4): pp. 580-590.

Bai R., E.K. Burke, M. Gendreau, G. Kendall and B. McCollum. (2007). *Memory Length in Hyper-Heuristics: An Empirical Study.* In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched2007), Hilton Hawaiian Village, Honolulu, Hawaii, USA.

Bard J. F. and H.W. Purnomo. (2005). *Preference Scheduling for Nurses using Column Generation.* Journal of Operational Research, 2005. 164: pp. 510-534.

Beddoe G., S. Petrovic and J. Li. (2009). *A Hybrid metaheuristic case-based reasoning system for nurse rostering.* Journal of Scheduling, 2009. **12**(2): pp. 99-119.

Beddoe G. and S. Petrovic. (2007). *Enhancing case-based reasoning for personnel rostering with selected tabu search concepts.* Journal of the Operational Research Society, 2007. **58**(12): pp.1586-1598.

Beliën J. and E. Demeulemeester. (2007). *Building cyclic master surgery schedules with leveled resulting bed occupancy.* European Journal of Operations Research, 2007. **176**: pp. 1185-1204.

Beliën J. and E. Demeulemeester. (2008). A Branch-and-price approach for integrating nurse and surgery scheduling. European Journal of Operational Research. 189(3), pp. 652-668, 2008. 0377-2217 DOI:10.1016/j.ejor.2006.10.060.

Berrada I., J.A. Ferland and Philippe Michelon. (1996). *A Multi-objective Approach to nurse Scheduling with both Hard and Soft Constraints.* Socio-Economic Planning Science, 1996. **30**(3): pp. 183-193.

Bilgin B., P. De Causmaecker, B. Rossie, G.V. Berghe (2012). Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, 2012, *194*(1), pp. 33-57.

Bilgin, B., P. Demeester, M. Mısır, W. Vancroonenburg, G. V. Berghe, T. Wauters (2010). A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In *The 8th international conference on the practice and theory of automated timetabling (PATAT'10)—the nurse rostering competition.*

Blake J. T., F. Dexter and J. Donald. (2002). *Operating room managers' use of integer programming for assigning block time to surgical groups: A case study.* Anesth Analg., 2002. **94**(1): pp. 143-148.

Brucker M., E. Burke, T. Curtois, R. Qu and G. Vanden Berghe. (2010). *A Shift Sequence Based Approach for Nurse Scheduling and New Benchmark Dataset.* Journal of Heuristics. 2010 (16): pp:559-573. 16:559-573, Doi: 10.1007/s10732-008-9099-6.

Brusco M. and L. W. Jacobs. (1995). *Cost Analysis of alternative formulations for personnel scheduling in continuously operating*

organizations. European Journal of Operational Research, 1995. **86**: pp. 249-261.

Burke E. K., P. Cowling, P. De Causmaecker and G.V.Berghe. (2001a). *A memetic approach to the nurse rostering problem.* Applied intelligence, 2001. **15**(3): pp. 199-214.

Burke E. K. and T. Curtois (2010). An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling PATAT 2010.*

Burke, E.K., T. Curtois, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic and J.A. Vazquez-Rodriguez. (2010a). *Iterated Local Search vs Hyper-heuristics: Towards General-Purpose Search Algorithms.* 2010 IEEE Congress on Evolutionary Computation, pp. 1-8.

Burke E.K., T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic and J.A. Vasquez-Rodriguez. (2009a). *HyFlex: A Flexible Framework for the Design and Analysis of hyper-heuristics.* In Multidisciplinary International Scheduling Conference (MISTA 2009), pp. 790-797, Dublin, Ireland, August 2009.

Burke E. K., T. Curtois, G. Post, R. Qu and B. Veltman. (2008a). *A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem.* European Journal of Operational Research, 2008. **188**(2): pp. 330-341.

Burke E. K., T. Curtois, R. Qu and G.V. Berghe. (2009b). *A scatter search for the nurse rostering problem.* Journal of the Operational Research Society, 2009.

Burke E. K., P. De Causmaecker, S. Petrovic and G.V. Berghe. (2006a). *Metaheuristics for handling time interval coverage constraints in nurse scheduling.* Applied Artificial Intelligence, 2006. **20**(9): pp.743-766.

Burke, E.K., P. De Causmaecker, G. Vanden Berghe and H. Van Landeghem. (2004a). *The State of Art of Nurse Rostering.* Journal of Scheduling, 2004. 7(6): pp. 441-499.

Burke E. K., P. De Causmaecker and G.V. Berghe. (2004b). *Novel meta-heuristic approaches to nurse rostering problems in belgian hospitals.*

Handbook of Scheduling: Algorithms, Models and Performance Analysis, J. Leung, Editor. 2004, CRC Press.

Burke E. K., P. De Causmaecker, S. Petrovic and G.V. Berghe (2004c) *Variable Neighborhood Search for Nurse Rostering Problems*, in *Metaheuristics: Computer Decision-Making*, M.G.C. Resende and J.P. de Sousa, Editors. 2004, Kluwer. pp. 153-172.

Burke E. K., P. De Causmaecker, S. Petrovic and G.V.Berghe. (2001b). *Fitness evaluation for nurse scheduling problems.* Proceedings of Congress on Evolutionary Computation, CEC2001, Seoul, IEEE Press, 2001, pp. 1139-1146.

Burke E. K., P. De Causmaecker, and G. Vanden Berghe. (1999a) A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem, in Simulated *Evolution and Learning, Selected Papers from the 2nd Asia-Pacific Conference on Simulated Evolution and Learning, SEAL 98, Springer Lecture Notes in Artificial Intelligence Volume 1585.* B. McKay, et al.,Editors. 1999: Springer. pp. 187-194.

Burke, E.K., E. Hart, G. Kendall, J. Newall, P. Ross and S. Schulendurg. (2003a). *Hyper-Heuristics: An emerging direction in modern search technology.* In Handbook of Metaheuristics, F. Glover and G. Kochenberger, Eds. Kluwer, 2003, pp. 457-474.

Burke, E.K., M. Gendreau, G. Ochoa, J. Walker. (2011a). *Adaptive Iterative Local Search for Cross-domain Optimisation.* In Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11, pp 1987-1994, New York, NY, USA, 2011. ACM.

Burke, E.K., M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu. (2009c). *A Survey of hyper-heuristics.* Technical report, School of Computer Science, University of Nottingham, 2009.

Burke, E.K., M. Hyde, G. Kendall, G. Ochoa, E. Özcan and J. Woodward. (2009d). *A Classification of Hyper-Heuristic Approaches.* In Handbook of Metaheuristics, ser. International Series in Operations Research & Management Science. Springer, 2009.

Burke E. K., M. Hyde, G. Kendall, G. Ochoa, E. Ozcan and J. Woodward. (2009e). *Exploring hyper-heuristic methodologies with genetic*

programming. In: Mumford C, Jain L (eds) Collaborative Computational Intelligence, Springer, 2009.

Burke E. K., G. Kendall and E. Soubeiga. (2003a). *A Tabu-Search Hyper-heuristic for Timetabling and Rostering.* Journal of Heuristics, 2003. **9**: pp.451-470.

Burke E. K., G. Kendall, M. Misir and E. Özcan. (2012a). Monte carlo hyper-heuristics for examination timetabling. Annals of Operations Research, 196:73-90, 2012.

Burke E. K., J. Li and R. Qu. (2009f). *A Hybrid Model of Integer Programming and Variable Neighbourhood Search for Highly-Constrained Nurse Rostering Problems.* Accepted for publication in *European Journal of Operational Research,* to appear 2010, DOI 10.1016/j.ejor.2009.07.036.

Burke E. K., J. Li and R. Qu. (2009g). *A Pareto-based search methodology for multi-objective nurse scheduling.* Annals of Operations Research, 2009, pp. 1-19.

Burke E. K., B. MacCarthy, S. Petrovic and R. Qu. (2000a). *Structured cases in case-based reasoning - re-using and adapting cases for time-tabling problems.* Knowledged-Based Systems, 2000. **13**(2-3): pp. 159-165.

Burke E. K., B. Mccollum, A. Meisels, S. Petrovic and R. Qu. (2007a). *A Graph-Based Hyper-Heuristic for Educational Timetabling Problems.* European Journal of Operational Research, 2007. **176**: pp. 177-192.

Burke E. K. and J. P. Newall. *A Multi-Stage Evolutionary Algorithm for the Timetable Problem.* (1999b). IEEE Transactions on Evolutionary Computation, 1999. 3.1: pp. 63-74.

Burke E. K., S. Petrovic and R. Qu. (2006b). *Case-Based Heuristic Selection for Timetabling Problems.* Journal of Scheduling, 2006. **9**(2): pp. 115-132.

Burke E. K. and A. Riise. (2008). Surgery Allocation and Scheduling. In Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling, 2008.

Cardoen B., E. Demeulemeester, J. Beliën (2010). Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 2010, *201*(3), pp. 921-932.

Chan C., F. Xue, W. Ip and C. Cheung. (2012). A hyper-heuristic inspired by pearl hunting. In Proceedings of LION6, LNCS 7219, pp.349-353, 2012.

Cichowicz T., M. Drozdowski, M. Frankiewicz, G. Pawlak, F Rytwinski, J. Wasilweski. (2012). Five phase and genetic hive hyper-heuristics for the cross-domain search. In Proceedings of LION 6, LNCS 7219, pp. 354-359, 2012.

Cowling, P., G. Kendall and E. Soubeiga. (2000). *A hyperheuristic approach for scheduling a sales summit.* In Selected Papers of the third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000, ser. Lecture Notes in Computer Science. Konstanz, Germany: Springer, August 2000, pp. 176-190.

Cowling, P., G. Kendall and E. Soubeiga. (2002). *Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling.* Lecture Notes in Computer Science, 2002. 2439: pp.851-860.

Curtois, T., G. Ochoa, M. Hyde, J. A. Vazquez-Rodriguez. (2010). A HyFlex Module for the Personnel Scheduling Problem, School of Computer Science, University of Nottingham, Tech. Rep.

Denton B., J. Viapiano and A. Vogl. *Optimization of surgery sequencing and scheduling decisions under uncertainty.* (2007). Health Care Management Science, 2007. **10**(1): 13-24.

Dexter F. and A. Macario. (2002). *Changing allocations of operating room time from a system based on historical utilization to one where the aim is to schedule as many surgical cases as possible.* Anesth Analg, 2002. **94**(5): pp. 1272-1279.

Di Gaspero L. and T. Urli. (2012). *Evaluation of a Family of Reinforcement Learning Cross-domain Optimization Heuristics.* In Hamadi, Y.Scoenauer, M., eds: Learning and Intelligent Optimization. Volume 0 of Lectures Notes in Computer Science. Springer Berlin/Heidelberg 92012) 384-389.

Dinh-Nguyen P. and A. Klinkert. (2008). *Surgical case scheduling as a generalized job shop scheduling problem.* European Journal of Operational Research, Volume 185, Issue 3, March 2008, pp:1011-1025.

Drake J., Özcan E. and E. Burke. (2012). *An Improved Choice Function Heuristic for Cross Domain Heuristic Search.* PPSN 2012, pp. 307-316, 2012.

Elomari J., (2012). Self-Search (Extended Abstract). CheSC 2011.

Ernst A.T., H. Jiang, M. Krishnamoorty, D. Sier. (2004). *Staff scheduling and rostering: a review of applications, methods and models.* European Journal of Operational Research, Volume 153, Issue 1, February 2004, pp:3-27.

Gendreau M., J. Ferland, B. Gendron, N. Hail, B. Jaumard,S. Lapierre, G. Pesant, and P. Soriano. (2006). *Physician Scheduling in Emergency Rooms.* In Edmund K. Burke and Hana Rudov, editors, PATAT 2006, pages 214, Faculty of Informatics, Masaryk University Brno, The Czech Republic, 2006. Masaryk University.

Glass, C.A. and R.A. Knight. (2010). *The nurse rostering problem: A critical appraisal of the problem structure.* European Journal of Operational Research, 2010. **202**(2): pp. 379-389

Guinet A. and S. Chaabane. (2003). *Operating theatre planning.* International Journal of Production Economics Planning and Control of Productive Systems, 2003. **85**(1): pp. 69-81.

Gomez J. (2011). Hybrid Adaptive Evolutionary Algorithm Hyper-Heuristic. CheSC 2011.

Hans E., G. Wullink, M. van Houdenhoven, G. Kazemier (2008). *Robust Surgery Loading.* European Journal of Operational Research, 2008. **185**: pp. 1038-1050.

Haspeslagh S., P. De Causmaecker, A. Schaerf, M. Stølevik (2010). The first international nurse rostering competition 2010. *Annals of Operations Research*, pp.1-16.

Hsu V. N., R. de Matta and C.-Y. Lee. (2003). *Scheduling Patients in an Ambulatory Surgical Center.* Naval Research Logistics, Vol. 50, pp. 218-238, 2003.

Hyde M., G. Ochoa, T. Curtois, J. A. Vazquez-Rodriguez (2010a) *A HyFlex Module for the One Dimensional Bin Packing Problem,* School of Computer Science, University of Nottingham, Tech. Rep.

Hyde M., G. Ochoa, T. Curtois, J. A. Vazquez-Rodriguez (2010b) *A HyFlex Module for the Maximum Satisfiability (MAX-SAT) Problem,* School of Computer Science, University of Nottingham, Tech. Rep.

Hsiao P.C., T.-C. Chiang and L-C Fu. (2012). *A VNS-based hyper-heuristic with adaptive computational budget of local search.* In Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC), pp.1-8, 2012.

Ikegami A. and A. Niwa. (2003). *A subproblem-centric model and approach to the nurse scheduling problem.* Mathematical Programming, 2003. **97**(3): pp. 517-541.

Jebali A., A. B. H. Alouane and P. Ladet. (2006). *Operating rooms scheduling.* International Journal of Production Economics Control and Management of Productive Systems, 2006, TY-JOUR. **99**(1-2): pp. 52-62.

Jones T. (1995). Crossover, *Macromutation, and Population-based search.* In Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, 1995, pp73-80.

Johnston M., T. Liddle, J. Miller and M. Zhang. (2011). *A Hyperheuristic Based on Dynamic Iterated Local Search.* CheSC 2011.

Kalender M., A. Kheiri, E. Özcan and E. Burke. (2012). *A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem.* In Proceedings of UKCI12, pp.1-8, 2012.

Khamassi I. (2011). *Ant-Q Hyper-heuristic Approach applied to the Cross-domain Heuristic Search Challenge problems.* CheSC 2011.

Kheiri A. and E. Özcan. (2013). *A Hyper-heuristic with a Round Robin Neighbourhood Selection.* Learning and Intelligent Optimization Conference 2013, under review.

Kubalik J. (2012). *Hyper-heuristic Based on Iterated Search Driven by Evolutionary Algorithm.* Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, 2012, Volume 7245/2012, 148-159, DOI: 10.1007/978-3-642-29124-1_13.

Larose M. (2011). *A Hyper-heuristic for the CHeSC 2011.* CheSC 2011.

Lehrbaum A. and N. Musliu. (2012). *A new hyper-heuristic algorithm for cross domain search problems.* In Proceedings of LION6, LNCS 7219, pp.437-442, 2012.

Li J., U. Aickelin and E. K. Burke. (2009). *A Component Based Heuristic Search Method with Evolutionary Eliminations for Hospital Personnel Scheduling.* INFORMS Journal on Computing. 21(3): pp 468-479, 2009.

Li J. and U. Aickelin. (2004). *The application of Bayesian Optimization and Classifier Systems in Nurse Scheduling.* Lecture Notes in Computer Science, 2004, pp.581-590.

Loudni S. (2012). *Intensification/Diversification-Driven ILS for a Graph Coloring Problem.* Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science, 2012, Volume 7245/2012, 160-171, DOI: 10.1007/978-3-642-29124-1_14.

Lourenco H. R., Martin O. C., Stutzle, T. (2003). *Iterated Local Search.* International Series in Operations Research and Management Science, 2003, Issue 57, pp. 321-354.

Lü, Z. and J. K. Hao (2012). Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 2012, *218*(3): pp. 865-876.

Mascia F. and T. Stutzle. (2012). *A non-adaptive stochastic local search algorithm for the CHeSC2011 competition.* In Proceedings of LION6, LNCS 7219, pp.101-114, 2012.

McClymont K. and E. C. Keedwell. (2011). *Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems.* In Proceedings of GECCO 2011, pp.2003-2010. ACM, 2011.

Meignan D. (2011). *An Evolutionary Programming Hyper-heuristic with Co-evolution for CHeSC 2011.*

Menana J. and S. Demassey. (2009). *Sequencing and Counting with the multi-regular Constraint.* In 6th International Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09), United States (2009). DOI: 10.1007/978-3-642-01929-6_3.

Messelis T. and P. De Causmaecker (2011). An algorithm selection approach for nurse rostering. In Proceedings of the 23rd Benelux Conference on Artificial Intelligence, 2011, pp. 160-166.

Métivier J.-P., P. Boizumault and S. Loudni. (2009). *Solving Nurse Rostering Problems Using Soft Global Constraints.* In Principals and Practice of Constraint Programming – CP 2009, I.P. Gent, Editor. 2009, Springer: pp.73-87.

Meyer auf'm Hofe H. (2001). *Solving rostering tasks as constraint optimization.* In E. K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling, Third International Conference*, Konstanz, Springer, Lecture Notes in Computer Science, vol. **2079**, 2001, pp. 191–212.

Meyer auf'm Hofe H. (1997). *ConPlan/SIEDAplan*:Personnel Assignment as a Problem of Hierarchical Constraint Satisfaction.* PACT-97 Contribution.

Misir M., K. Verbeeck, P. De Causmaecker and G. Vanden Berghe. (2012). *An Intelligent Hyper-heuristic Framework for CHeSC 2011.* The 6th Learning and Intelligent OptimizatioN Conference (LION12), Paris, France.

Moscato P. and M.G. Norman. (1992). *A memetic approach for the travelling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems.* Parallel Computing and Transputer Applications, 1992, pp. 177-186.

Moz M. and M. Vaz Pato. (2004). *Solving the Problem of Rerostering Nurse Schedules with Hard Constraints: New Multicommodity Flow Models.* Annals of Operations Research, 2004. **128**: pp. 179-197.

Nareyek A. (2003). *Choosing search heuristics by non-stationary reinforcement learning.* In: Resende MGC, de Sousa JP (eds) Metaheuristics: Computer Decision-Making, Kluwer, chap. 9, pp. 523-544, 2003.

Nawaz M., E. Enscore Jr. and I. Ham. (1983). *A Heuristic algorithm for the m-machine, n-job flow-shop sequencing problem.* OMEGA-International Journal of Management Science, 11(1):91, pp.91-95, 1983.

Nonobe K (2010). INRC2010: an approach using a general constraint optimization solver. *the website for the First International Nurse Roastering Competition.*

Núñez J. and A. Ceballos. (2011). *A general purpose Hyper-heuristic based on Ant Colony optimization.* CheSC 2011.

Ochoa G., M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, E. K. Burke. (2012). *HyFlex: A Benchmark Framework for Cross-domain Heuristic Search.* (2012). European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012), LNCS series, Vol. 7245, Springer.

Ochoa, G., R. Qu and E. K. Burke. (2009a). *Analyzing the Landscape of Graph Based hyper-heuristic for Timetabling Problems.* GECCO 09 Montreal, Canada. In Proceedings of the 11th Annual Conference on Genetic and evolutionary computation, pp: 341-348.

Ochoa, G., J. A. Vazquez-Rodriguez, S. Petrovic, E. K. Burke. (2009b). *Dispatching Rules for Production Scheduling: A Hyper-heuristic Landscape Analysis.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC-09), IEEE Press, Trondheim, Norway, 2009.

Ouelhadj D. and S. Petrovic. (2008). *A Cooperative Distributed Hyper-Heuristic Framework for Scheduling.* Proceedings of the 2008 IEEE International Conference on Systems, Man and Cybernetics, Singapore, p. 2560-2565, 2008.

Özcan E., Bilgin B. and Korkmaz E.E. (2008). *A Comprehensive Analysis of Hyper-heuristics.* Intelligent Data Analysis, IOS Press, 2008. **12**:pp.3-23.

Özcan E., Bilgin B. and Korkmaz E.E. (2006). *Hill Climbers and Mutational Heuristics in Hyperheuristics.* Parallel Problem Solving from Nature-PPSN IX. Lecture Notes in Computer Science, vol. 4193, 2006.

Özcan E. and A. Kheiri. (2011). *A Hyper-heuristic based on Random Gradient, Greedy and Dominance.* Erol Gelenbe, Ricardo Lent, Georgia

Sakellari (Eds.) Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences, pp.404-409, 2011.

Petrovic S., G. Beddoe and G. Vanden Berghe. (2003). *Storing and Adapting Repair Experiences in Employee Rostering.* In E. Burke and P. De Causmaecker (eds): PATAT2002, LNCS 2740, pp. 148-165, 2003.

Qu R. and E.K. Burke. (2009). *Hybridisations within a Graph Based Hyper-heuristic Framework for university Timetabling Problems.* Journal of the Operational Research Society, 2009. **60**(9): pp.1273-1285.

Qu R. and E. K. Burke. (2007). *Adaptive Decomposition and Construction for Examination Timetabling Problems.* Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications 2007 (MISTA 2007), 418-425, Aug. 2007, Paris, France.

Qu R., E. K. Burke and B. McCollum. (2009). *Adaptive Automated Construction of Hybrid Heuristics for Exam Timetabling and Graph Colouring Problems.* European Journal of Operational Research, 2009. **198**(2): pp. 392-404.

Riise A. and E. K. Burke. (2011). Local search for the surgery admission problem. Journal of Heuristics. 17:pp. 389-414, 2011. DOI 10.1007/s10732-010-9139-x.

Rizzato D.B., A. A. Constantino, E. L. de Melo, D. Landa-Silva, W. Romão (2010). INRC2010. Heuristic Algorithm Based on Multi-Assignment Problems For Nurse Rostering Problem. *The website for the First International Nurse Rostering Competition.*

Rohlfshagen P. and J. Bullinaria. (2007). *A genetic algorithm with exon shuffling crossover for hard bin packing problems.* In Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07), pp. 1365-1371, London, U.K., 2007.

Santibanez P., M. Begen, and D. Atkins. (2007). *Surgical block scheduling in a system of hospitals: an application to resource and wait list management in a british columbia health authority.* Health Care Management Science, 2007, TY-JOUR. **10**(3): pp.269-282.

Schaerf A. and A. Meisels. (1999). *Solving employee timetabling problems by generalised local search.* Proceedings Italian AI, 1999, pp. 493–502.

Sim K. (2011). *A Simulated Annealing Hyper-heuristic with Reinforcement Learning and Tabu Search.* CheSC 2011.

Smet P., B. Bilgin, P. De Causmaecker, G. Vanden Berghe (2012). *Modelling and evaluation issues in nurse rostering.* Annals of Operations Research (2012): pp.1-24.

Smith L. and A. Wiggins. (1977). *A Computer-Based Nurse Scheduling System.* Computers and Operations Research, 1977. 4(3): pp. 195-212.

Smith L. D. (1976). *The application of an interactive algorithm to develop cyclical rotational schedules for Nursing personnel.* INFOR, 1976. **14**: pp. 53–70.

Trivedi V. M. and M. Warner. (1976). *A branch and bound algorithm for optimum allocation of float nurses.* Management Science, 1976. **22**(9): pp. 972-981.

Valouxis C., C. Gogos, G. Goulas, P. Alefragis, E. Housos (2012). A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 2012, *219*(2): pp. 425-433.

Valouxis C. and E. Housos. (2000). *Hybrid optimisation techniques for the workshift and rest assignment of nursing personnel.* Artificial Intelligence in Medicine, 2000. **20**: pp. 155–175.

Vazquez-Rodriguez J.A., G. Ochoa, T. Curtois, M. Hyde. (2010). *A HyFlex Module for the Permutation Flow Shop Problem*, School of Computer Science, University of Nottingham. Tech. Rep.

Walker J., G. Ochoa, M. Gendreau and E. Burke. (2012). *Vehicle Routing and Adaptive Iterated Local Search within the HyFlex Hyper-heuristic.* The 6th Learning and Intelligence OptimizatioN Conference (LION12), Paris, France.

Weil, G., K. Heus, P. Francois, and M. Poujade. (1995). Constraint programming for nurse scheduling. IEEE Engineering in Medicine and Biology Magazine, 1995. 14(4): p. 417-422.