Construction-based Metaheuristics for Personnel Scheduling Problems

by

Melissa D. Goodman

Thesis submitted to Cardiff University
in candidature for the degree of
Doctor of Philosophy

School of Mathematics
Cardiff University

November 2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgements

I would like to extend my thanks to the many people without whom this thesis never would have been completed. Firstly, to my two supervisors Dr. Jonathan Thompson and Dr. Kathryn Dowsland, who have made this research possible, and whose unerring support has been invaluable throughout the process. I would also like to thank the staff at the Cardiff University School of Mathematics for their invaluable assistance and the Engineering and Physical Sciences Research Council (EPSRC) who provided the funding for this research.

Finally, I would like to offer my appreciation for the constant support and encouragement provided by my family and friends.

Summary

This thesis investigates the idea of balancing different constraints in order to find optimal solutions to two personnel scheduling problems, within the framework of constructive metaheuristic approaches. The two problems considered are a nurse scheduling problem, for which finding feasible solutions is known to be difficult and for which the hard and soft constraints are in direct conflict, and a medical student scheduling problem for which there is little relevant literature; this second problem also has conflicting hard and soft constraints, but presents further conflict between the different soft constraints. The methods used to solve these problems are focused on two constructive metaheuristics in particular: Greedy Randomised Adaptive Search Procedures (GRASP) and Ant Colony Optimisation (ACO) and for each approach several construction heuristics are introduced and compared. Using GRASP, a number of local search neighbourhoods are established for each problem, while for ACO the suitability of three trail definitions are compared. In order to further explore the balance which may obtained between the different constraints and objectives for the two problems, hybrid constructions are investigated, incorporating exact methods which take advantage of the underlying structures of each problem with regards to feasibility. For medical student scheduling, this exact method was developed into a new type of construction mechanism providing much improved results over a standard heuristic approach. Further enhancements investigated include the use of problem-specific feedback for nurse scheduling and the use of an intelligent memory procedure for the medical student scheduling problem. For the nurse scheduling problem, the final algorithm developed was able to rival the best in the literature so far and produce optimal solutions for all available datasets. For the medical student scheduling problem, optimal solutions are not known, but the results obtained are very promising and provide a good basis for further study of the problem.

Contents

List of figures

Chapter 2 – Introduction to problems

Chapter 3 – Literature review

Chapter 4 – Nurse scheduling with GRASP

Chapter 5 – Nurse scheduling with ACO

Chapter 6 – Medical student scheduling with GRASP

List of tables

Chapter 4 – Nurse scheduling with GRASP

Chapter 5 – Nurse scheduling with ACO

Chapter 6 – Medical student scheduling with GRASP

Appendix A – Nurse scheduling datasets

Chapter 1

Introduction

In today's society, a multitude of large and complex problems arise which require high-quality solutions. Usually, there are a number of different constraints on the problem which must be satisfied and, further, these can be split into hard and soft constraints depending on whether they are essential or merely desirable, respectively. Problems with both hard and soft constraints have, essentially, two separate objectives: firstly, all the hard constraints must be satisfied in order for the solution to be feasible and secondly, the soft constraints must be satisfied as far as possible. Where there are several soft constraints, this raises a further issue of which of these constraints are the most important and, from an algorithmic point of view, how to deal with the problem of setting appropriate weights. Often, different constraints will be in direct conflict with one another, and so a trade-off is necessary in order to find optimal solutions. Additional difficulties occur when the satisfaction of the hard constraints is non-trivial and this raises a further issue of how much bias can be given to the feasibility aspects of the problem without adversely affecting the final solution quality. Clearly, if the bias is too much in favour of the soft constraints, resulting

solutions will not be feasible, but by focusing solely on the satisfaction of the feasibility constraints, optimal solutions may be missed.

With very small problems, all possible solutions to the problem may be enumerated and finding an optimal solution is reduced to the task of merely choosing the solution with the best cost. With larger problems this is not feasible, but, for certain problems, linear and mathematical programming models will be able to find the optimal solution in a reasonable amount of time. However, for NP-hard problems, the amount of time required to solve such a model grows exponentially with problem size and so, for this type of problem, inexact, heuristic methods may be used to search for good solutions. Although optimal solutions cannot be guaranteed, these types of methods have been shown to be capable of producing high-quality solutions within a much shorter time-frame.

The focus of this thesis is the careful balance required between the different constraints in these types of large NP-hard problems, involving hierarchies of objectives and conflicting constraints, and the resulting trade-offs necessary to produce high-quality solutions. In particular attention is given to the solution of two multiple-choice scheduling problems, a nurse scheduling problem and a medical student scheduling problem, within a constructive metaheuristic framework. Both problems consist of assigning a number of tasks to each individual such that certain constraints are met. For both problems there are conflicting hard and soft constraints and these must be successfully balanced in order for optimal solutions to be produced. Both have been chosen as they are suitable to be solved using an iterative constructive technique and the solution methods applied to each problem will be discussed in the next section.

The nurse scheduling problem consists of assigning each nurse in a ward with a complete set of shifts for a one-week planning period such that the staffing requirements of the ward are met for each of 14 shifts and 3 skill or 'grade' levels. For each available nurse, a single cost relating to the soft constraints is known for each complete set of potential shifts, or 'shift patterns'. These costs are based on the nurses' individual preferences; by keeping these costs low, staff satisfaction is increased. The problem has been solved previously using a variety of techniques and

obtaining this balance between the two aspects of the problem has proved very difficult. It is interesting to consider here because a great deal about the problem structure is known and so different ways of balancing the hard and soft constraints can be investigated with prior knowledge of the difficulty of each problem aspect. Further, since optimal solutions have been obtained using an IP approach, often requiring an excessive amount of computation time, the results of solution approaches in this thesis can be compared directly with the optimal costs, as well as with those of other metaheuristic techniques.

The medical student scheduling problem is interesting for study for a different set of reasons. As well as providing conflict between the hard and soft constraints, further conflict exists between different soft constraints. It consists of assigning each student to one location or 'firm' in each of 5 timeslots in which to study the 5 necessary specialities such that each speciality is studied exactly once and the capacities of the firms not exceeded in any timeslot. The soft constraints are incorporated to give each student as much experience as possible, by penalising situations where a student visits the same hospital, is taught by the same consultant, or is placed in the same firm as any other given student, on more than one occasion. While the nurse scheduling problem has had the soft constraint violations combined into a linear cost function, so that one cost exists for each potential nursing assignment, the objective function of the medical student scheduling problem presents both quadratic and quartic terms, so the cost of each assignment depends very much on every other. Not only does this mean that the objective function is more complicated, but it means that an IP formulation of the problem is not practical due to the number of binary variables which would be required in order to linearise it; thus optimal solutions are not known. Further, this problem has not been well studied and so the structure of this problem and the particular difficulties associated with achieving good solutions have not been identified. However, the similarities between the two problems allow an exchange of ideas between them about the construction methods. The aim is first to find successful methods for balancing the constraints of the nurse scheduling problem and then to apply these techniques as far as possible to the medical student scheduling problem. The aim of this thesis is to successfully identify methods of balancing different and conflicting constraints and to evaluate the methods successful on one problem on another. By developing an understanding of the delicate balance required for the nurse

scheduling problem, known to be difficult in this respect, and further, by applying similar techniques to another, relatively unknown problem with a similar requirement for such a balance, but a very different problem structure, a greater understanding of the difficulties of finding a suitable balance and the successful techniques for overcoming these difficulties can be provided.

There are two types of metaheuristics: those which are initialised with a random solution, such as Tabu Search and Simulated Annealing, and those for which the construction of each solution is important, such as Greedy Randomised Adaptive Search Procedures (GRASP), Ant Colony Optimisation (ACO), and Genetic Algorithms (GAs) where the solution components are not encoded directly. Manpower scheduling problems naturally lend themselves to a solution approach of the second type since, when assigning tasks to personnel, it is possible to bias the choice of each assignment such that solution quality is improved. As mentioned, this thesis is concerned with creating solutions within a constructive metaheuristic framework and the two problems chosen are suitable for such methods to be applied. The two metaheuristic approaches applied in this thesis are GRASP and ACO. Both of these fit into the second metaheuristic category, using a constructive approach as a major part of the solution process; GRASP creates one construction and then uses a local search approach to improve this constructed solution, applying this two-part process repeatedly in what has been likened to a repetitive sampling technique; ACO can be viewed as a population based approach and uses information from the solutions in previous cycles to feed back information about which assignments were successful, in order to aid the construction of better solutions in future. GRASP and ACO have been widely employed in the literature and yet the role of the construction has been very little studied, although much work has been done on enhancing the basic algorithms as will be discussed further in the literature review in Chapter 3.

1.1 Overview of research presented in this thesis

As discussed, this thesis presents two problems: nurse scheduling and medical student scheduling. This variant of the nurse scheduling problem has been the subject of much investigation. This research is motivated by the work of Aickelin (1999), who presented a series of heuristics to construct a solution which balance the bias between

feasibility and optimality, within the context of a GA approach. The quality of the results obtained was variable, but the research showed that the idea of balancing conflicting constraints within a constructive framework was worth further research. The two metaheuristics introduced to solve the problems are GRASP and ACO, as mentioned previously. These two approaches use a similar type of construction and they are chosen for study because ideas from one can easily be carried over into the other; a single GRASP construction with certain parameters is equivalent to an ACO algorithm with population size 1 and with no feedback from previous solutions.

One of the important features of both these scheduling problems is that each has an underlying structure pertaining to the feasibility which may be exploited in order to improve the chances of creating feasible solutions. It shall be shown how, for the nurse scheduling problem, a knapsack model may be used to ensure that any constructed solutions may be made feasible through the use of local search and how, for the medical student scheduling problem, the hard constraints may be modelled as a network flow problem and thus solved exactly. For the medical student scheduling problem, then, the use of this network flow model in conjunction with a heuristic construction is able to guarantee that the constructed solutions will be feasible. The knapsack model, relating to the feasibility of the nurse scheduling problem, is based only on a relaxation of the hard constraints, however, and so is not able to guarantee feasibility, but only make feasibility more achievable. The hybridisation of techniques to balance the different constraints will prove very important for both problems and will be discussed more fully in the appropriate results sections.

The original intention was to apply both GRASP and ACO to both nurse scheduling and medical student scheduling and both of these approaches have been applied to the nurse scheduling problem. However, after a further investigation of the medical student scheduling problem structure in Chapter 2, and considering the information gained by applying ACO to the nurse scheduling problem in Chapter 5 and the poor quality of the GRASP constructions for medical student scheduling obtained in Chapter 6, it was decided that an ACO application to the medical student scheduling problem would not be worthwhile. Chapter 2 goes into some of the reasons why the medical student scheduling problem does not lend itself well to the successful incorporation of feedback from one solution to the next.

1.2 Aims

This thesis has three main aims, all of which are related to the exploitation of a high-quality constructive approach. These aims are:

1. To investigate the role of the construction within a metaheuristic approach.
2. To investigate how different constraints may be balanced within a construction.
3. To investigate how exploiting problem structure with regards to feasibility may improve constraint balance within the construction.

A secondary aim is of course to try to produce high-quality solutions for the problems studied. A fourth aim is therefore:

4. To produce a robust method for each problem, capable of producing high-quality solutions for all problem instances.

Each of these aims is now discussed in further detail.

1. Investigating the role of the construction within a metaheuristic framework

By investigating both GRASP and ACO approaches to the problem, the roles of the construction and improvement phases in a solution approach can be determined as well as the relative importance of each. As has been discussed, the GRASP and ACO constructions are very similar and it will be interesting to investigate how much of the information necessary to successfully balance the different constraints can be built into the constructions and whether it is possible to maintain high solution quality only using very simple neighbourhoods. The question which will need to be answered is whether the construction can provide sufficient balance to enable satisfaction of all constraints or whether some aspects must necessarily be left to the local search. For example, it shall be investigated whether or not results can be improved by allowing the construction to concentrate only on one aspect, such as feasibility, rather than finding a balance. This type of approach provides a new type of balance to be investigated, occurring between the construction and local search, a comparison which does not often appear in the literature. The ACO algorithm often incorporates

an element of local search to improve solution quality, even though it is not part of the standard algorithm, and it will be interesting to note whether the feedback mechanism will be sufficient to build high-quality solutions, or whether local search will still be necessary.

In terms of the nurse scheduling problem, the most successful approach in the literature is a tabu search method, which is initialised with a random solution and focuses on improving using very complex and problem-specific neighbourhoods. It will be interesting, therefore, to see whether good solutions can be obtained for this problem with a method which exploits an aggressive construction rather than local search.

2. Investigating how different constraints may be balanced within a construction

The problems presented in this thesis provide a variety of conflicting constraints and in order to find good solutions a balance must be found, such that all hard constraints are satisfied, but not at the expense of unnecessary soft constraint violations. As mentioned previously, while the nurse scheduling problem offers conflict between the hard and soft constraints, the medical student scheduling problem offers further conflict between a number of different soft constraints. In order to successfully construct high-quality schedules, it is essential that an appropriate score function can be found which will be able to balance the weights associated with each type of constraint.

3. Investigating how exploiting problem structure with regards to feasibility may improve constraint balance within the construction.

As mentioned, each problem has an underlying structure pertaining to the feasibility which allows the constraints, or a relaxation of the constraints to be modelled exactly. Clearly, by incorporating these types of exact approaches, the construction is biased towards feasibility. This thesis investigates the need for the use of such methods in order to determine whether they are a necessary addition or whether the bias in favour of feasibility is too great and becomes an obstacle to the satisfaction of the soft constraints. On the other hand, incorporating the feasibility constraints in this way

may enable the heuristic score functions to focus more strongly on the remaining constraints enabling solution quality to be improved overall. It will need to be determined whether this type of exploitation is advantageous and the effect it has on the balance required by the heuristic used alongside it.

4. Producing a robust method for each problem, capable of producing high-quality solutions for all problem instances.

Secondary to the main thrust of the research, it will be interesting to determine whether, using the construction-based techniques explored in the fulfilment of aims 1 – 3, the methods produced will be able to produce high-quality solutions. In terms of the nurse scheduling problem, the optimal solutions are known and the best heuristic approach in the literature so far is a tabu search method capable of producing optimal solutions for all known datasets in a reasonable timeframe. It will therefore be interesting to see whether the methods produced in this thesis using a construction-based approach will be able to rival the improvement-based approach of the tabu search. Note that more details of the methods applied to the nurse scheduling problem will be given in Chapters 2 and 3. For medical student scheduling, optimal solutions are not known and there is no research in the literature with which results may be compared. However, it will be interesting to see whether a robust, construction-based approach is capable of being found.

The next section gives a breakdown of the thesis structure and explains the contents of the remaining chapters.

1.3 Thesis structure

The following chapter gives an overview of the two problems investigated in this thesis, giving full details of each problem, including its mathematical formulation, and discussing the particular difficulties of finding good-quality solutions and interesting features of the problem structure, such as the ability to model some or all of the hard constraints so that they may be solved exactly. The chapter serves to introduce the problems and discuss them on an equal footing before the literature review in Chapter 3; there is little relevant literature for the medical student scheduling problem, while

the nurse scheduling problem presented here has been much more widely solved. Further, Chapter 2 provides sufficient detail of each problem such that the literature review may be tailored to include the most relevant details from the literature and such that details of the different methods applied may easily be given in a context suitable for the remainder of this thesis.

As mentioned previously, there is little available literature on the medical student scheduling problem and, while problems similar to medical student scheduling are discussed in Chapter 2, the literature review in Chapter 3 must necessarily be focused on the literature relating to the nurse scheduling problem. This literature review provides details of the variants of the nurse scheduling problem which arise in different hospitals around the world and the differences in the types of constraints which arise as well as other, more fundamental differences. A history of the GRASP and ACO metaheuristics is provided as well as a history of the approaches which have been previously applied to nurse scheduling. Finally, a complete overview of the methods previously applied to this variant of the problem is presented along with the particular difficulties relating to balancing the different constraints which arose from each approach.

Chapters 4 and 5 relate to the investigation of solution methods for the nurse scheduling problem. Chapter 4 provides details of the investigation using a GRASP approach and Chapter 5, an ACO approach. Chapter 4 discusses the different heuristics which may be applied in conjunction with a GRASP construction as well as suitable local search neighbourhoods. The usefulness of combining the GRASP with the knapsack model is a major feature of the investigation and the GRASP is hybridised both with the knapsack model as well as with two types of feedback in order to produce the final solutions. The notion of exploring equal cost moves in the local search neighbourhoods is also put forward as a possible enhancement to final solution quality. Chapter 5 then shows the difference in quality obtained by using a GRASP approach and that of ACO, using the same constructive heuristics. After the different potential pheromone trail definitions to utilise the ACO feedback are introduced, an investigation into the many different parameters required is performed. The final ACO algorithm, whose success is compared with that of the GRASP

approach in Chapter 4 is hybridised both with the knapsack model as well as an element of local search.

Chapter 6 then details the investigation into the medical student scheduling problem using GRASP. The issue of whether or not the network flow model is necessary or helpful to the search for optimal solutions is explored and, as well as using heuristics based on those introduced in Chapter 4, heuristics based entirely on a network flow model of the problem are introduced. Although, as will be discussed further in Chapter 2, an ACO approach is not suitable for this problem, the idea of incorporating memory without the use of feedback is investigated and several ways of incorporating this memory are compared for the best of the approaches so far. Although optimal solutions are not known for this problem, the robustness of the resulting algorithms is tested by considering the number of times the best known solution is produced. Unlike the nurse scheduling problem, for which details of the 52 datasets can be found in Appendix A, the medical student scheduling problem was provided with only two sets of real-world data. Chapter 6, therefore, also provides details of how these two datasets were used to build 46 further datasets which are both realistic and interesting to study.

The final chapter of this thesis gives a summary of the findings and draws conclusions in keeping with the aims set out in this introduction. Suggestions for further research are also presented.

Note that all computation times provided in the following chapters refer to experiments carried out on a Celeron M 1.5 GHz notebook computer.

Chapter 2

Introduction to problems

2.1 Introduction

As stated in the introduction, this thesis is concerned with balancing different objectives using a metaheuristic framework relying on greedy, randomised constructive techniques. The two construction-based metaheuristics on which this work will be based are Greedy Randomised Adaptive Search Procedure (GRASP) and Ant Colony Optimisation (ACO). These will be introduced in detail in the next chapter. This chapter introduces the two problems, nurse scheduling and medical student scheduling, on which all experiments will be carried out and explains why they are both suitable and interesting for study and comparison. Further, since the medical student scheduling problem is new and therefore does not appear much in the literature, this chapter allows us to introduce these two problems on an equal footing before the literature review in Chapter 3. Finally, the nurse scheduling problem arises in many different forms and by introducing the variant we are concerned with here, we may tailor the literature review to allow more detailed information about the most relevant research. Of particular interest in this thesis is the balance between feasibility

and optimality and the two problems introduced here both possess interesting characteristics in this respect; we present a nurse scheduling problem, for which much previous research has shown that obtaining feasible solutions is not trivial and a medical student scheduling problem, for which there is little relevant literature, but the many, varied objectives provide an interesting framework for study.

The nurse scheduling problem arises in many forms and has been solved using a variety of methods. The variant presented here arises at a major UK hospital and has been solved using several approaches; finding a balance between feasible solutions and solutions with low-cost in terms of staff preferences has been found to be non-trivial. Previous research on this problem will be summarised in Section 2.2.3 and discussed further in Section 3.5 of Chapter 3. By selecting this problem for study, we are able to investigate solution approaches and compare their results with those in the literature as well as with the known optimal solutions.

However, although having known solutions available enables a direct evaluation of results, it is also interesting to approach a problem which has never been solved other than by hand, by using very simple heuristics (Harris 1997), or by attempting to model aspects of the problem as an Integer Program (IP) (Fuller 1998). Optimal solutions are therefore not known. The particular difficulties of the nurse scheduling problem have already been studied, but by approaching a totally new problem, the specific difficulties are not yet established and so these too must be discovered.

We therefore put forward these two problems: nurse scheduling, a well known problem with interesting attributes including conflict between feasibility and optimality; and medical student scheduling, a previously unsolved problem arising at a major university and involving several potentially conflicting objectives.

The rest of the chapter is organised as follows. Section 2.2 relates to the nurse scheduling problem investigated in this thesis. First, a description of the problem in real-world terms is given and this is followed by its mathematical formulation. The section ends with a summary of the previous approaches applied to the problem and a discussion of the problem structure. In particular, a method for aiding the construction of feasible solutions is discussed. Section 2.3 relates to the medical student scheduling

problem and, again, the problem description and formulation are given. Although no literature is available on this problem, it bears a resemblance to a variant of the 3-dimensional assignment problem and so a brief discussion of this similarity and any relevant literature is given. The section ends with details of the problem structure and, again, a brief discussion of the potential methods aiding the search for feasibility. The final section compares the two problems presented in this thesis.

We begin by looking at the nurse scheduling problem.

2.2 The nurse scheduling problem

The nurse scheduling problem is a well-known scheduling problem which arises in hospital wards all over the world. Although the details of the problem vary in different countries, the essence of the problem is to assign shifts to each nurse for the given planning period, such that constraints are adhered to and objective costs minimised. The problem presented here has been much studied and provides a well-established database of 52 individual problems based on real-world data. For each of these the optimal solutions are known, and so the success of the GRASP and ACO approaches presented here may be judged individually, as well as relative to the previous heuristic methods applied.

The next sections give the details and mathematical formulation of the nurse scheduling problem arising at a major UK hospital, which is to be the focus of this research, after which follows a summary of the particular difficulties encountered by previous approaches.

2.2.1 Problem description

Weekly rosters for individual hospital wards are required. Each roster must ensure that the hospital has sufficient staff at all levels of seniority throughout the week, whilst adhering to the nurses' contracts and fulfilling their personal preferences as far as possible. The working day is divided into three shifts: an 'early' day shift (0700 – 1430), a 'late' day shift (1400 – 2130) and a night shift (2100 – 0730). Only the allocation of nurses to either days or nights is considered, since assigning the nurses to

either an early or late shift can be solved exactly using a network flow model in a post-processing phase (Thompson and Dowsland 2000).

The majority of nurses work either day shifts or night shifts in a one week planning period, rather than a combination of the two and due to the different shift lengths, a nurse working days will typically work more shifts than a nurse working nights. For example, a full time nurse works either 5 day shifts or 4 night shifts. This will be denoted by (5,4). All wards considered have a number of part time nurses, working combinations such as (4,3) and (3,2). A small number of nurses may work combinations with equal numbers of day and night shifts, such as (3,3), and a small number of nurses will work a specific number of both days and nights in the same week. A nurse working both 1 day shift and 2 night shifts in a week will be denoted by (1&2).

The possible nursing contracts are (5,4), (4,4), (4,3), (3,3), (3,2), (2,2), (2,1), (1,1), (1&3), (1&2) and (0,0) where a nurse has a scheduled week off, giving eleven possible nursing 'types'. Practically, however, the weekly contracts may vary slightly, since nurses for whom either day or night shifts have been disallowed would essentially have a different contract; a full time nurse with night shifts disallowed would, in effect, have a (5,0) contract for that week. Where a nurse has a scheduled training day, this may also result in an altered weekly contract. As will be discussed in more detail in Chapter 3, Dowsland and Thompson (2000) modified the data in a pre-processing phase to ensure that any surplus shifts are spread evenly across the weekdays, as requested by the hospital. This was done by adding surplus requirement to even out the demand and including 'dummy' nurses to cover the extra shifts and is explained more fully in Appendix C. Many of the nurses with equal numbers of day and night shifts are examples of such dummy nurses and so these often have night shifts disallowed. The nurses are divided into three grades according to their level of knowledge and experience. These grades can be categorised as follows:

Grade 1: Senior nurses, capable of being in charge of the ward.
Grade 2: Trained nurses, with a high level of nursing expertise.
Grade 3: Trained nurses, with the lowest level of training.

For each shift, there is a minimum requirement for the number of nurses of each grade on duty. Although lower grade nurses may not replace higher grade nurses, the reverse is allowed and so these staffing constraints, known as the 'cover' constraints, are expressed cumulatively. For example the requirement 2/4/9 for a particular shift means that at least two grade 1 nurses, four nurses of grade 2 or higher and nine nurses in total are required.

The primary objective is to produce a feasible schedule, that is, one which satisfies the following constraints:

C1. Cover constraints satisfied for all shifts.

C2. Each nurse works the contracted number of shifts.

C3. Annual leave and study days adhered to.

C4. No nurse works a night shift followed immediately by a day shift or a day shift followed by a night shift on the same day.

In addition to producing feasible schedules, it is desirable that the rosters a number of secondary requirements, most of which relate to enabling the nurses to enjoy a healthy social and family life. To this end, several objectives are defined:

P1. Nurses can request not to work certain shifts / days.

P2. Each nurse has a full Saturday or Sunday off at least every other week.

P3. Night shifts are rotated between nurses.

P4. Very senior nurses are not scheduled on nights or weekends as this is too expensive.

P5. Nurses are assigned sequences of days off together rather than individual days off being spread over the week.

P6. Nurses do not work more than seven consecutive days.

P7. Nurses with many violations of these soft constraints in previous weeks are allocated more desirable shifts in this current schedule.

For each nurse working d day shifts or e night shifts, it is possible to enumerate all workable shift patterns. A shift pattern is defined here as a binary string of length 14, where the first 7 digits represent the seven day shifts from Sunday to Saturday and the second set represents the nights. A '1' indicates that the nurse works, while a '0'

signifies a day or night off. The number of shift patterns for a nurse is ${}^{7}C_{d}$ patterns of day shifts and ${}^{7}C_{e}$ patterns of nights. For nurses working both days and nights in the same week, all their feasible shift patterns can also be considered. We choose to deal with assignments of nurses to shift patterns rather than to individual shifts in order to simplify the problem, as do all previous approaches which tackle this problem in the literature. Aickelin (1999), in particular, used shift patterns in conjunction with his Genetic Algorithm approaches and, given that many of the constructive heuristics developed for the nurse scheduling problem in this thesis are based on this work and the data available is therefore in this format, it is sensible for us to adopt this shift pattern approach as well.

The objectives P1-P7 have been combined to give a single penalty cost, in the range [0, 100] for each nurse-shift pattern combination, with '0' implying the shift pattern is ideal and '100' signifying it is unacceptable. These values were obtained by Dowsland and Thompson from dialogue with the nurses. Full details of how the scores are derived can be found in Appendix B. Thus, the problem is reduced to finding a feasible roster that minimises the sum of these penalty costs and, since all factors affecting optimality are combined by this shift pattern approach, there are fewer aspects of the problem to balance.

Note that any constructive heuristic approach would have two possible approaches for creating the nursing schedules: either a nurse may be selected, along with a shift pattern for that nurse, or a shift pattern may be selected, and a nurse assigned to it. Thus the schedules may be created by systematically working nurse by nurse, until all nurses are assigned or pattern by pattern. The latter, however presents a fairly illogical way to proceed, since each shift pattern may be used once, more than once or not at all. By working through the nurses systematically, however, the task becomes much easier to handle and so, in Chapters 4 and 5, it is this approach which will be adopted.

2.2.2 Problem formulation

This problem can be formulated mathematically as follows:

Indices:

$i = 1,...,r$	nurse index, where r is the number of nurses
$g = 1,...,3$	grade index
$j = 1,...,m$	shift pattern index, where m is the total number of shift patterns
$k = 1,...,14$	day index (where 1,…,7=days and 8,…,14=nights)

Variables:

$$x_{ij} = \begin{cases} 1 & \text{if nurse } i \text{ works shift pattern } j \\ 0 & \text{otherwise} \end{cases}$$

Constants:

$$a_{jk} = \begin{cases} 1 & \text{if shift pattern } j \text{ covers day } k \\ 0 & \text{otherwise} \end{cases}$$

$$h_{ig} = \begin{cases} 1 & \text{if nurse } i \text{ is of grade } g \text{ or higher} \\ 0 & \text{otherwise} \end{cases}$$

p_{ij} = cost of nurse i working shift pattern j

If we say S_i is the set of shift patterns which nurse i can work and C_{gk} is the number of nurses of grade g or higher required on shift k, then we have the following.

Min

$$\zeta = \sum_{i=1}^{r} \sum_{j \in S_i} p_{ij} x_{ij}, \tag{2.1}$$

subject to:

$$\sum_{j \in S_i} x_{ij} = 1 \quad \forall i, \tag{2.2}$$

$$\sum_{i=1}^{r} \sum_{j \in S_i} h_{ig} a_{jk} x_{ij} \geq C_{gk}, \quad \forall g, k \tag{2.3}$$

The objective function (2.1), made up of the sum of the preference costs, must be minimised subject to the satisfaction of (2.2) and (2.3), where (2.2) ensures each nurse works exactly one shift pattern and (2.3) ensures that the cover requirements are sufficient for each shift and at each grade.

Typical problem dimensions are 25 nurses with a total of $m = 411$ possible shift patterns; the number of patterns each nurse may work varies according to their 'type', i.e. number of day and/or night shifts they are contracted to work. A full time nurse works 5 days or 4 nights and so has ${}^{7}C_{5} + {}^{7}C_{4} = 56$ potentially feasible patterns if both days and nights are allowed; part time nurses may have more or fewer patterns depending on the number of days and/or nights they must work.

Solving this IP exactly requires specialist software to which it is unlikely hospitals would have access. Even using such software, finding an optimal solution can take several hours (Fuller, 1998), although more recent software, certainly beyond the budget of a hospital ward, is more efficient. However, it is important to the hospital staff that they are able to choose from a range of varied schedules (Dowsland and Thompson 2000) and, even using today's IP software, producing a range of optimal solutions is still an issue (Tsai et al. 2008). Thus, a heuristic approach is justified.

2.2.3 Summary of previous approaches to the nurse scheduling problem

This section provides a summary of the research in the literature for this variant of the nurse scheduling problem. A more detailed review of the previous research for this problem appears in the literature review in the next chapter.

Dowsland (1998) presented a tabu search method to solve the nurse scheduling problem. Infeasible solutions were unavoidably included in the search space due to the difficulty of the problem and the disconnectedness of the feasible region made it necessary to revisit the infeasible region repeatedly in order to adequately search the solution space. In order to reach an initial feasible schedule from the random starting solution the partition of nurses into days and nights was found to be of particular importance; often solutions would partition the nurses such that no simple neighbourhood move would be sufficient to improve feasibility due to the 'imbalance'

of nurses working on days and nights. This arises from the fact that most nurses work a different number of days than nights and so it is important for the 'correct' nurses to work each type of shift. Even when solutions were not 'unbalanced', often very large and complex neighbourhoods had to be employed in order to attain feasibility. Thus, finding feasible solutions to this problem is not trivial. Dowsland and Thompson (2000) used this work as a basis for the software CARE, which employs the pre-processing phase to evenly spread excess cover as discussed earlier and a post-processing phase to allocate the day shifts to either 'early' or 'late'. It has since been employed in the hospital from which this data was gathered.

Aickelin (1999) originally solved this set of nurse scheduling problems using two Genetic Algorithm (GA) approaches. A straightforward GA approach, encoding each solution as a string of shift patterns for each nurse, produced comparatively poor results mainly due to the non-linearity between each 'gene', or nurse-pattern pair, in the string and the final solution cost. The feasibility of the solution relied on the interaction of all individual genes and so it was not necessarily the case that combining two feasible solutions would be likely to result in feasible children. This feature of non-linearity will be discussed again in more detail when discussing feedback which may be used in conjunction with the GRASP methods and the trail definitions suitable for an ACO approach.

Aickelin's second GA approach used the GA simply to order the nurses and then each nurse was assigned a shift pattern in turn using a greedy heuristic which decoded the ordering into a completed schedule. This method produced higher quality solutions than the original GA method and it is these decoding heuristics which are used as the basis for the GRASP and ACO approaches presented in this thesis. The particular details of Aickelin's (1999) decoders can be found in greater detail in Chapter 3, while details of the GRASP and ACO approaches presented for nurse scheduling are given in Chapters 4 and 5, respectively.

Other approaches used to tackle this problem have included using a hyperheuristic approach (Cowling et al. 2002, Burke et al. 2003b), where a number of low-level heuristics are each selected with dynamic probability to improve a randomly generated schedule and a Bayesian Optimisation algorithm (Li and Aickelin 2003,

Aickelin and Li 2007) which selects which rule should be used at each stage of the construction process using conditional probabilities calculated from previous constructions.

For each of these approaches, finding feasible solutions was a major challenge. As discussed, the tabu search approach in Dowsland (1998) and Dowsland and Thompson (2000) made use of large and complicated neighbourhoods in order to provide feasible solutions in all cases, and both GA approaches in Aickelin (1999) failed to find feasible solutions for all cases. The hyperheuristic approaches fared well with regards to feasibility, but to the detriment of solution quality. The Bayesian Optimisation algorithm also managed to find feasible solutions in all cases, but showed that a simplified version not using the conditional probabilities failed to produce a single feasible solution from 20 attempts for each dataset. Even though this approach performed well in terms of feasibility, the algorithm often struggled to produce optimal solutions.

Full details of the existing literature on this problem are given in Section 3.5 of Chapter 3. Here we have summarised some of the more interesting research, pointing out that finding feasible solutions is not trivial and that some effort must be put into this aspect of the scheduling process if feasible solutions are to be found. Of the methods which have provided feasible solutions for all datasets, the solution quality is generally quite poor, and only the original tabu search approach (Dowsland 1998, Dowsland and Thompson 2000) has been able to reduce both the cover and the feasibility costs to the optimal in all cases; to do this, however, a great deal of problem-specific information was utilised and the resulting method relies heavily on large, complex neighbourhoods involving chains of moves in order to achieve this result.

2.2.4 Nurse scheduling problem structure

We now provide a discussion relating to the nurse scheduling problem structure and the particular difficulties inherent in the problem which must be tackled effectively for good solutions to be found.

As has been mentioned previously, the nurse scheduling problem is one for which it is not trivial to find feasible solutions and a large part of any solution approach must be devoted to this task if the algorithm is to have any chance of producing feasible schedules. According to Garey and Johnson (1979), the staff scheduling problem, consisting of simplified version of the problem of finding a feasible solution, where all nurses are of the same grade and work the same number of shifts, and where the is no distinction between working days and night, is NP-hard.

However, an interesting feature of the nurse scheduling problem is that it is possible to exploit the problem structure to exclude a large number of infeasible solutions from the search space. In the last section, it was mentioned that Dowsland (1998) showed that the particular distribution of nurses working days or night was an integral part of whether or not a feasible solution could be obtained and this idea was also exploited in Aickelin's second GA approach. Dowsland and Thompson (2000) showed that it is possible to determine whether or not a particular problem has a feasible solution, given the number and types of nurses available by solving a knapsack model of the problem derived from the relaxation of constraint (2.3). This was done by summing the day and night requirement at each grade and replacing the x_{ij} variables, relating to whether or not nurse i works pattern j, with a variable which merely determines whether or not each nurse works days or nights. We build on this idea, creating a system for maintaining the day/night balance during the construction of a solution, and ensuring that large areas of the infeasible region of the search space are never visited.

Regardless of the day/night balance existing in solutions created in conjunction with the knapsack model, the particular shift patterns for each nurse are not guaranteed to create a feasible solution. Thus, creating feasible solutions remains non-trivial and creating solutions which are both feasible and optimal more difficult still. Chapter 4 will introduce this knapsack model more fully and will explain in detail how it may be used in conjunction with a GRASP solution approach.

2.3 The medical student scheduling problem

This section provides an introduction to the medical student scheduling problem. Unlike nurse scheduling, this is a new problem with little previous work in the literature with which to draw comparison. However, the problem does share similarities with nurse scheduling in that we must allocate assignments to people over the given time-frame. Obviously these problems are only loosely related and the similarities and differences between them will be discussed more fully in Section 2.4. Just two sets of real-world data for this problem have been made available by the university hospital concerned, but, from these, we have been able to manufacture a further 46 datasets encompassing a wide range of realistic variations. The details of these will be given in Chapter 6.

In the next section we give details of the medical student scheduling problem and formulate it mathematically. We then discuss the particular difficulties of this problem and compare it with the nurse scheduling problem which we discussed in Section 2.2, showing that it is a sufficiently difficult problem to warrant our investigation. We begin with an overview of the problem.

2.3.1 Problem description

During their training, medical students are required to complete several years of clinical experience, which involves completing placements in different hospitals and covering a number of specialities. Creating their schedules is a relatively unexplored problem and, with several hard and soft constraints, finding optimal solutions is not trivial. The particular variant of the problem we will be looking at arises yearly at a large university in Wales, although other universities in the UK and around the world will face a similar scheduling challenge.

The first clinical year is split into five timeslots, with each student studying one of five specialities in each stage. In the first timeslot, all students study an introductory module; the students then complete the other four specialities exactly once in any order in the remaining time periods. The hospitals each contain a number of 'firms' consisting of groups of one or more consultants; each consultant may belong to more

than one firm. The introductory module is offered across the board; each hospital will offer a number of firms designed for this purpose, while subsequent modules will only be offered by firms associated with that particular speciality. The number of places available in each firm is known as the 'capacity' of that firm and these values are known in advance. A solution is a list of which firm each student will attend during each of the five time periods.

For a solution to be feasible, each of the following hard constraints must be satisfied:

F1. Each student must cover the introductory module in time period 1.
F2. Each student must cover the remaining four specialities exactly once in time periods 2-5.
F3. The number of students placed in each firm in each time period must not exceed the capacity of that firm.

It is also desirable for students to get as broad a range of experience as possible during their placements and so, for a solution to be optimal, we seek to minimise violations of three further soft constraints.

O1. Each student is not allocated to the same consultant more than once.
O2. Each student is not allocated to the same hospital more than once.
O3. Each pair of students is not allocated to the same firm at the same time more than once.

In the case of nurse scheduling, in Section 2.2, the objectives P1-P7 were combined into a single score for each nurse-shift pair. For this problem, however, it is not sensible to enumerate the cost for all possible firm combinations for each of the students. The first reason for this is that the students' costs are not independent; part of the cost is derived from the repetition of student pairings. As such we cannot know how much each student will contribute to the cost until all students have been scheduled. A secondary consideration is the large number of feasible firm-timeslot combinations. For the smaller of the two datasets, specialities 1 – 5 consist of 38, 9, 6, 12 and 4 firms, respectively. Thus the number of possible combinations is $38\times9\times6\times12\times4\times4!$, since specialities 2 – 5 may be studied in any order, giving over

2.3×10^6 potential combinations. For the larger dataset this figure is more than 20 times as great and to calculate these partial scores would not be computationally worthwhile. Further, it is interesting not to amalgamate these soft constraints into a single cost as it allows us to consider balancing the trade-offs between them. We therefore abandon the idea of creating an amalgamated score for each possibility in advance.

Given that the medical student scheduling problem can be large, with students numbering in the hundreds, finding a zero-cost solution to a particular problem is not guaranteed. In order to ensure that any compromise in our final solution rests with the least important of the soft constraints, weightings were applied to each constraint according to their relative importance. These were produced in consultation with the university.

It is seen as more important that the consultant constraint (O1) is adhered to than the hospital constraint (O2) and, in turn, the hospital constraint (O2) than the student pairs constraint (O3). We assign a weighting to each of these of 50:10:1, respectively. These match the trade-off in solution quality as perceived by the university so that ten violations of the student pair constraint is viewed equally undesirably as a single hospital constraint violation, for example.

2.3.2 Problem formulation

Formulating the problem mathematically, we have the following:

Indices:

$s = 1,\ldots,r$	student index, where r is the number of students
$t = 1,\ldots,5$	time period index
$f = 1,\ldots,v$	firm index, where v is the number of firms
$q = 1,\ldots,5$	speciality index

Variable:

$$x_{sft} = \begin{cases} 1 & \text{if student } s \text{ is placed in firm } f \text{ in timeslot } t \\ 0 & \text{otherwise} \end{cases}$$

Constants:

C_f = capacity of firm f

$$c_{fg} = \begin{cases} 1 & \text{if firms } f \text{ and } g \text{ have a consultant in common} \\ 0 & \text{otherwise} \end{cases}$$

$$h_{fg} = \begin{cases} 1 & \text{if firms } f \text{ and } g \text{ are in the same hospital} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{fq} = \begin{cases} 1 & \text{if firm } f \text{ offers speciality } q \\ 0 & \text{otherwise} \end{cases}$$

The part of the cost associated with the consultant clashes (O1) is given by

$$\sum_{s=1}^{r}\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{f=1}^{v}\sum_{g=1}^{v} c_{fg}x_{sft}x_{sgu}\,, \tag{2.4}$$

and the part associated with the hospital clashes (O2) is given by

$$\sum_{s=1}^{r}\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{f=1}^{v}\sum_{g=1}^{v} h_{fg}x_{sft}x_{sgu}\,. \tag{2.5}$$

The cost for the student pairs (O3) is slightly more complicated. For the hospital and consultant clashes, we count every instance where a student is in the same hospital or with the same consultant more than once. The way we count this means that if the student is in the same hospital twice, this is counted once, but if the student is in the same hospital three times, each pair is counted and so this occurrence is penalised with a cost three times that of the case where the student only has two hospitals in common. In the same way, if a student had all five placements in the same hospital, counting each pair would result in a cost 10 times that of the situation where the hospital was visited only twice. This reflects the fact that the requirement for students not to visit the same hospitals or be placed with the same consultants is not linear, and it is more desirable for a student to visit 2 hospitals twice than to be placed in one hospital three times, for example.

When dealing with the student pair costs we have a quadratic term. Only pairs of student pairs must be counted. Single instances of a student pairing must clearly result in no additional cost. We are able to count such instances of repeated pairings using the following formula:

$$\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{s=1}^{r-1}\sum_{z=s+1}^{r}\sum_{f=1}^{v}\sum_{g=1}^{v} x_{sft}x_{zft}x_{sgu}x_{zgu}\,. \tag{2.6}$$

Combining these three with the weights given earlier, we arrive at the final cost function.

We must minimise (2.7) subject to (2.8), (2.9), (2.10) and (2.11).

Min

$$\zeta = \sum_{s=1}^{r}\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{f=1}^{v}\sum_{g=1}^{v}\left[50c_{fg}x_{sft}x_{sgu} + 10h_{fg}x_{sft}x_{sgu}\right] + \sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{s=1}^{r-1}\sum_{z=s+1}^{r}\sum_{f=1}^{v}\sum_{g=1}^{v} x_{sft}x_{zft}x_{sgu}x_{zgu} \tag{2.7}$$

Subject to:

$$\sum_{t=1}^{5}\sum_{f=1}^{v} a_{fq}x_{sft} = 1,\ \forall s,q \tag{2.8}$$

$$\sum_{f=1}^{v} a_{f1}x_{sf1} = 1,\ \forall s \tag{2.9}$$

$$\sum_{f=1}^{v} x_{sft} = 1,\ \forall s,t \tag{2.10}$$

$$\sum_{s=1}^{r} x_{sft} \le C_f,\ \forall f,t \tag{2.11}$$

Equation (2.8) ensures that each speciality is covered exactly once, (2.9) ensures that each student covers speciality 1 in time period 1, (2.10) ensures that each student is assigned to only one firm in each timeslot and (2.11) ensures that the capacities of

each firm are adhered to in all time periods. Thus equations (2.8) – (2.10) ensure that F1 and F2 are satisfied, while F3 is covered by the last inequality.

Given that the medical student scheduling problem has both quadratic and quartic terms, it is not currently possible to solve it using an IP formulation and professional software due to the unreasonably large number of variables which would be required in order to linearise it (Fuller 1998). We are therefore justified in seeking heuristic approaches to tackle the problem.

2.3.3 Literature relevant to the medical student scheduling problem

Fuller (1998) created an IP model which tackled only the hard constraints and the hospital clashes. A combination of exact and heuristic approaches were used to find a solution to one instance of the medical student scheduling problem as presented here. The main difference in the problem formulation is that the consultant constraints were only considered after the development of the initial model and are only assigned a weight of 5, relative to the other costs, as supposed to the weight of 50 later adopted. The method consisted of finding a zero-cost partial solution for a subset of students using a greedy approach and then applying the IP to schedule the remaining students.

Since, there is little relevant literature about this problem, we look instead at a closely related classical combinatorial optimisation problem. Since we are assigning triplets, by assigning each student to one available place in a firm in each timeslot, we essentially have a variant of the three-dimensional assignment problem, and we now briefly discuss the relevant features of this problem and the solution approaches applied.

The three-dimensional assignment problem (3AP) comes in two forms: axial and planar. It is the planar version which the medical student scheduling problem most closely resembles and we now give its formulation for comparison purposes.

Min

$$\sum_{i=1}^{i'}\sum_{j=1}^{j'}\sum_{k=1}^{k'} b_{ijk}x_{ijk}\,, \qquad (2.12)$$

subject to

$$\sum_{i=1}^{i'} x_{ijk} = 1 \; \forall j,k \tag{2.13}$$

$$\sum_{j=1}^{j'} x_{ijk} \leq 1 \; \forall i,k \tag{2.14}$$

$$\sum_{k=1}^{k'} x_{ijk} \leq 1 \; \forall i,j \tag{2.15}$$

$$x_{ijk} \in \{0,1\} \; \forall i,j,k \tag{2.16}$$

where b_{ijk} is the cost of assigning the 3-tuple (i,j,k). The planar three-dimensional assignment problem has been shown to be NP-hard (Frieze 1983, Hofstra 1983). An overview of this and other assignment problems can be found in Gilbert and Hofstra (1988) and Pentico (2007).

Clearly these two problems have very different objective functions and the quadratic and quartic terms present in the objective of the medical student scheduling problem mean that algorithms based on an IP approach are unlikely to be successful for this problem. However, approaches which are not based on the linearity of the objective function, such as the types of heuristic approaches we are investigating, may be equally applicable to both problems and it is for this reason that we call attention to the planar three-dimensional assignment problem and look for literature related to it which may be relevant to medical student scheduling. First we discuss the similarities between the constraints of these two problems, which are even more apparent if we consider only timeslots 2 – 5. Although there is a similarity between constraints (2.13) – (2.15) of the 3AP and constraints (2.8) – (2.11) of the medical student scheduling problem, this is complicated by the fact that the dimension relating to the specialities is partitioned into different firms. Further the right hand side of the constraint relating to this dimension is not equal to 1.

The relationship between the two problems becomes clearer if we consider a different model of the 3AP, where $i' = j' = k' = n$ for some integer n and the constraints (2.14) and (2.15) are equations, rather than inequalities. Then there is a one-to-one relationship between the solutions of the 3AP and the set of $n \times n$ Latin squares. A

Latin square is an $n \times n$ array such that in each row and column every integer from 1 to n appears exactly once. Thus if we let j be the row index, k the column index and i the entry value, we can set up the equivalence relation between the Latin square and a solution to the 3AP by taking values

$$x_{ijk} = \begin{cases} 1 & \text{if } a_{jk} = i \\ 0 & \text{otherwise} \end{cases}$$

where a_{jk} is the entry in row j and column k. Then the constraints correspond exactly to each a_{jk} containing exactly one value, each row j containing entry i in exactly one column k and each column k containing entry i in exactly one row j. Figure 2.1 shows an example of such a Latin square for the case $n = 4$.

		k			
		1	2	3	4
j	1	1	3	2	4
	2	3	2	4	1
	3	2	4	1	3
	4	4	1	3	2

Figure 2.1. An example of an $n \times n$ Latin square with entries $a_{jk} = i$.

Magos (1996) was successful in applying a tabu search method to this problem based on the idea of using a Latin square representation, which forms the basis of the neighbourhood structure. A neighbouring move is, essentially, a Latin square which can be reached from the current one by changing the value of a single entry and then repairing it via a chain of moves.

Now consider a medical student scheduling problem in which the total number of places available is equal to the number of students. Let i be the student index, j represent the index of places and k the index of timeslots such that all indices relating to a given speciality are allocated consecutively forming 4 blocks. Feasible solutions are the set of $n \times 4$ Latin rectangles with the additional constraint that each i occurs in a different block in a different column. Figure 2.2 shows this setup with 192 students and 48 places available in each of specialities 2 – 5 in each timeslot.

			Timeslot k			
			2	3	4	5
Place j	Spec. 2	1	192	58	22	16
		2	38	52	160	181
		⋮	⋮	⋮	⋮	⋮
		48	7	90	63	19
	Spec. 3	49	16	192	90	63
		50	160	181	58	22
		⋮	⋮	⋮	⋮	⋮
		96	52	19	7	38
	Spec. 4	97	58	38	192	160
		98	19	16	52	7
		⋮	⋮	⋮	⋮	⋮
		144	22	63	181	90
	Spec. 5	145	181	7	19	192
		146	63	22	16	58
		⋮	⋮	⋮	⋮	⋮
		192	90	160	38	52

Figure 2.2. Latin rectangle setup of the medical student scheduling problem for specialities 2 – 5.

In the example setup shown in Figure 2.2, the students i in positions a_{jk} in the array have been chosen randomly and positioned merely to demonstrate the effect of the extra constraint which ensures that each student i appears in a different block in each column.

Although we have managed to demonstrate a similarity between the medical student scheduling problem and the 3AP, there are several important differences. One of these is the fact that the total available capacity need not equal the number of students and this is the case for many of our 48 datasets. The initial solutions used by Magos (1996) set each entry i in position a_{jk} to be assigned using

$$a_{jk} = \begin{cases} (j+k-1) \bmod n & \text{if } (j+k-1) \bmod n > 0 \\ n & \text{otherwise} \end{cases}. \tag{2.17}$$

Where the number of students does not equal the number of places available, this leads to the necessary inclusion of empty cells which add further complexity and influences the ease with which feasible solutions may be created and maintained. Certainly using a straightforward algorithm such as (2.17) to create an initial solution

would not be practical, since it requires a one-one correspondence between the number of students and the number of places available.

Note also that Magos (1996) studies problems of size up to $n = 14$, and indicates that problems of size $n > 9$ could not be solved optimally within a reasonable timeframe. A problem of size $n = 14$ gives a 14×14 Latin square containing 196 values with which to experiment. Our problem would generate a Latin square of at least size 192×4 which presents 768 entries and so even if a similar approach were possible, it may not be computationally feasible.

Therefore, this sort of approach, as well as presenting problems in terms of getting an initial solution, would be likely to struggle for a problem as large as that of medical student scheduling and the neighbourhoods could easily become unmanageable. For this reason, we do not attempt to apply a similar method to our problem.

Gilbert and Hofstra (1987) show that the planar three-dimensional assignment problem is a suitable model for a scheduling problem arising in trade shows, where one-to-one meetings must occur between a number of buyers and sellers in a limited number of time periods. There are fewer buyers than sellers and all buyers must be assigned a meeting in every time period, while each seller must have some specified minimum number of meetings. Costs are assigned to each buyer-seller pairing depending on the desirability of the meeting according to both parties. However, due to the fact that the time of the meetings is of no consequence, the solution method is able to be done in two stages; in the first stage a set of minimum cost buyer-seller pairings is determined by ignoring the time constraints, while the second stage assigns these pairings to timeslots. By breaking down the solution approach in this way, the problem can be solved in polynomial time. The first stage reduces the problem to two dimensions with variables $z_{jk} = 1$ if buyer j meets seller k, and 0 otherwise. The second part of the solution process, assigning these pairs to timeslots, can then be solved using simple bounds on the number of times each buyer and seller can appear in each timeslot. Note that this type of approach would not be suitable for the medical student scheduling problem, since assignments would not be able to take into account the firm capacities as well as the fact that every student must cover every speciality.

Furthermore, for the medical student scheduling problem, there are no costs relating to individual assignments and so this type of approach would not be sensible.

Another interesting area of research which is relevant to the medical student scheduling problem is that of the quadratic three-dimensional assignment problem (Q3AP). This is essentially an extension of the quadratic assignment problem, but shares some features with our medical student scheduling problem. It is formulated as follows.

Min

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} b_{ijk}x_{ijk} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\substack{l=1\\ l\neq i}}^{n}\sum_{\substack{m=1\\ m\neq j}}^{n}\sum_{\substack{p=1\\ p\neq k}}^{n} d_{ijklmp}x_{ijk}x_{lmp} \tag{2.18}$$

subject to

$$\sum_{i=1}^{i'}\sum_{j=1}^{j'} x_{ijk} = 1 \;\forall k \tag{2.19}$$

$$\sum_{i=1}^{i'}\sum_{k=1}^{k'} x_{ijk} = 1 \;\forall j \tag{2.20}$$

$$\sum_{j=1}^{j'}\sum_{k=1}^{k'} x_{ijk} = 1 \;\forall i \tag{2.21}$$

where b_{ijk} is the cost of assigning the three-tuple (i,j,k) and d_{ijklmp} is the cost of assigning both (i,j,k) and (l,m,p). Although the medical school scheduling problem has no costs b_{ijk} related to an individual assignment, the second part of the Q3AP objective function (2.18) resembles the part of the medical school scheduling problem objective related to hospital and consultant clashes. The cost d_{ijklmp} as concerns medical student scheduling, related to making both assignments x_{ijk} and x_{lmp}, is given by

$$d_{ijklmp} = \begin{cases} 50c_{jm} + 10h_{jm} & \text{for } i = l, k \neq p \\ 0 & \text{otherwise} \end{cases}.$$

Note that if $i = l$ and $k \neq p$, this implies $j \neq m$.

The two problems are therefore similar, although constraints (2.19) – (2.21) are related to the axial version of the 3AP, rather than the planar version the medical student scheduling problem resembles. Further, although the objective function is similar, the medical student scheduling problem contains still higher order terms, which is one of the features which makes it so interesting and also difficult to solve using exact methods. The exact method proposed for the Q3AP in Hahn et al. (2008) linearizes the problem by replacing each occurrence of a quadratic term $x_{ijk}x_{lmp}$ with a variable v_{ijklmp}. This would not be a realistic approach for the medical student scheduling problem since the higher order terms mean that the number of variables required in order to linearize the problem would be impractical. The heuristic methods put forward by Hahn et al. were tabu search, simulated annealing, an ACO approach and an iterated local search (ILS). Of these, the ILS was the most successful, while the ACO approach proved the worst performing of the four. Although we later suggest strong reasons why an ACO approach may not practically be applied to medical student scheduling, the weakness of its success as applied to the Q3AP does nothing to encourage its use. However, all heuristic methods tested relied on the underlying structure of the Q3AP which, as for the QAP and the axial version of the 3AP which this problem resembles, allows solutions to be represented as permutations. Again, this approach is not applicable to the medical student scheduling problem given that the number of students need not equal the number of places available. However, the success of the heuristic methods applied to the Q3AP goes some way to justify applying such techniques to the medical student scheduling problem.

2.3.4 Medical student scheduling problem structure

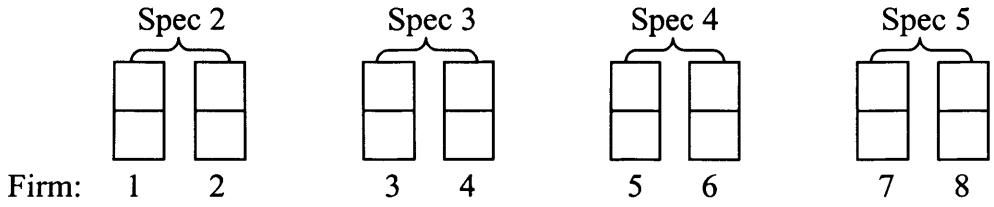
Although little is known about the specific difficulties of this problem and what type of approach is necessary in order to produce high-quality solutions, it is possible to recognise features which may be important when going through the solution process.

As with the nurse scheduling problem, we consider the difficulty of finding feasible solutions. In this case, however, it is easy to construct feasible solutions. Since each time period offers the same number of spaces for each speciality, with the obvious

exception of time period 1, we can easily create a feasible solution by splitting the students evenly into four distinct groups, assigning each a place in speciality 1 in time period 1 and then assigning the rest of the specialities in a cyclical manner. For example, if the first group were assigned specialities {2, 3, 4, 5} in that order, the second group assigned {3, 4, 5, 2}, the third, {4, 5, 2, 3} and the final group, {5, 2, 3, 4}, a feasible schedule must be possible since it is known that there are enough spaces for a quarter of the students to study each speciality at any given time and since no two groups will study the speciality at the same time. Obviously any other non-overlapping arrangement of specialities would give the same result. Thus, if necessary, we are able to guarantee feasible solutions.

However, by creating a timetable in this fashion, we are severely restricting the region of the search space which can be explored and the solution quality is likely to be very poor as a result. For example, the student-pair costs (2.6) are likely to be higher than necessary since each student is restricted to studying with a subset of only a quarter of the total number of students. It is likely, therefore, that a large number of repeated pairings will arise as the students necessarily study in the same places as each other repeatedly. Consider the following small example in Figure 2.3.

Consider the case where 16 students must each study 5 specialities in 5 timeslots as usual. Let speciality 1 be provided by 8 firms of two places each and each of specialities 2 – 5 be covered by 2 firms each providing two places. Thus there is just enough space for all students to study speciality 1 and a quarter of the students to study each speciality in the remaining timeslots. If we disregard the hospital and consultant constraints for the purposes of this example, then we are left with the problem of merely distributing the students adequately such that no pair of students is in the same firm twice. Let us first consider the case where the 16 students are divided into 4 groups of 4 to cover specialities 2 – 5 in rotation. Thus group 1 would study specialities 2, 3, 4, 5 in order, group 2 would study 3, 4, 5, 2 and so on, such that the resulting timetable will be feasible. We must therefore find, for each group, a firm for each student in the given speciality such that no students are paired together more than once. If such an allocation can be found for one group, the same can be applied to all other groups and so only one such allocation is necessary. If this can be found, a further suitable grouping of students must be found for speciality 1. Consider the following setup for specialities 2 – 5, where each box represents a place in a firm.



In order to minimise the number of student pairs resulting when allocating 4 students to specialities 2, 3, 4 and 5 in that order, we may begin by allocating the first 2 students the following firms:

Student 1: 1, 3, 5, 7

Student 2: 1, 4, 6, 8

Figure 2.3. Example demonstrating how using a group-rotation approach to guarantee feasibility may result in higher than necessary student pair costs. (Continued on next page).

Note that once student 2 has been allocated to firm 1 in timeslot 2, the student has no choice of firm for the remaining timeslots if there are to be no further pairings with student 1. Note that Student 1's choice of firms is unimportant since, initially, all places are homogeneous and student 2's choices must also exist since there must be 2 students studying in firm 1 in timeslot 2. At this point, however, there is no way to allocate the third student without incurring more than one pairing with another student. The first three timeslots can be allocated thus:

Student 3: 2, 3, 6,... (or alternatively 2, 4, 5...)

But the fourth place will necessarily result in a repeated pairing with either student 1 or 2. Thus using this group-rotation method of assignment we cannot avoid student pair costs.

However, by allowing students to consider the specialities in any order, an allocation is possible such that all students have a feasible timetable and no student pair costs are incurred. Allow the 8 firms in timeslot 1 to be denoted by the letters A – H and the firms in each other speciality as previously. Then the following is an example of such a feasible distribution.

Student	Firms	Specialities
1	A, 1, 3, 5, 7	1, 2, 3, 4, 5
2	B, 1, 5, 7, 3	1, 2, 4, 5, 3
3	A, 2, 8, 4, 6	1, 2, 5, 3, 4
4	B, 2, 4, 6, 8	1, 2, 3, 4, 5
5	C, 3, 1, 8, 5	1, 3, 2, 5, 4
6	D, 3, 7, 6, 1	1, 3, 5, 4, 2
7	C, 4, 6, 1, 7	1, 3, 4, 2, 5
8	D, 4, 2, 7, 5	1, 3, 2, 5, 4
9	E, 5, 3, 1, 8	1, 4, 3, 2, 5
10	F, 5, 7, 3, 2	1, 4, 5, 3, 2
11	E, 6, 2, 8, 3	1, 4, 2, 5, 3
12	F, 6, 8, 2, 4	1, 4, 5, 2, 3
13	G, 7, 5, 3, 1	1, 5, 4, 3, 2
14	H, 7, 1, 5, 4	1, 5, 2, 4, 3
15	G, 8, 4, 2, 6	1, 5, 3, 2, 4
16	H, 8, 6, 4, 2	1, 5, 4, 3, 2

Figure 2.3 contd. Example demonstrating how using a group-rotation approach to guarantee feasibility may result in higher than necessary student pair costs. (Continued on next page).

Thus by allowing each student the freedom to study the specialities in any order, a solution has been found of a higher quality than that possible using the group-rotation method.

Figure 2.3 contd. Example demonstrating how using a group-rotation approach to guarantee feasibility may result in higher than necessary student pair costs.

By restricting the available choices for each student, it is not only the student-pair costs which are likely to suffer, however. Removing options can only limit the quality of the resulting timetables, and if the optimal schedule does not take this cyclic form then by choosing such an approach we are effectively restricting the landscape of solutions to preclude this optimal schedule.

Chapter 6 introduces a network flow model which may be used in conjunction with a medical student scheduling approach in order to force solutions to be feasible without utilising such a rigid solution structure. In the case of nurse scheduling, it was a knapsack model used on a relaxation of the hard constraints which may improve the feasibility of the resulting solutions; we will show how a network flow model of this problem will result in all solutions being feasible. However, depending on the difficulty of finding feasible solutions, shown thus far to be relatively simple, such an approach may be more restrictive than helpful. A full discussion of this model and its application will be given in Chapter 6.

Given that we have shown that feasible solutions can easily be guaranteed, the particular difficulty of the medical student scheduling problem does not lie solely with feasibility: it is possible to create a low-quality, feasible solution and then use other measures to improve it. For the nurse scheduling problem this is not a realistic option. The balance between the constraints is therefore different between these two problems and so is likely to give rise to different issues.

Unlike the nurse scheduling problem, the medical student scheduling problem does not have a cost allocated to each assignment; the costs gradually become known as the schedule is built. Therefore near the beginning of the construction all assignments will be cost-free and it is only when several students have been assigned and choices of

firms become restricted that costs are incurred. The student-pair cost in particular is likely to be difficult to minimise during a solution's construction in many cases; when an allocation is made for a particular student, the hospital and consultant costs arising will be known, whereas the student pair costs will not become apparent until a later stage. Obviously the very last students to be allocated in the final timeslot will have a good idea of the resulting student pair costs, but at this stage there will be limited places remaining and so the cost of the allocation would be relatively inconsequential.

A further noteworthy feature of the medical student scheduling problem is the inherent symmetry in the problem structure. For comparison purposes this symmetry makes it difficult to distinguish between any two solutions found. While in the case of nurse scheduling, there is no symmetry and all rosters can be easily identified by the shift pattern allocated to each nurse, or to a nurse of a specific type, in the case of medical student scheduling the problem of distinguishing between one solution and another with the same cost becomes a much more arduous task. This has repercussions on our ability to utilise feedback, as will be discussed later, as well as the ability to judge the range of solutions being produced by each solution method. A further consequence of the multitude of symmetries in the problem is the lack of variation between choices in the early stages of the construction. The symmetries involved are as follows:

1. *Between students.* Until students are allocated their first placement, students remain homogeneous. Even after some allocations have been made, there may be symmetry between students depending on the particular firms to which they have been assigned, since, as is discussed below, there may be an element of symmetry between certain firms. Obviously students assigned to the same firms will still be homogeneous.

2. *Between timeslots.* Timeslot 1 is unique; all students study the introductory module at this stage. Timeslots 2-5, however, are interchangeable. Once a timetable is obtained, the fundamental nature of the solution remains unchanged, irrespective of the order in which timeslots 2-5 are presented.

3. *Between firms*. There is an amount of firm-specific information with each dataset. Such information includes: the firm capacity, the speciality covered, the hospital the firm belongs to, as well as any consultant clashes with other firms. Although this firm-specific information reduces the symmetries between firms, in several cases there will be groups of firms identical in each of these respects and so, again, we have an opportunity to alter the final timetable without changing the fundamental nature of the solution.

Even beyond these more obvious symmetries, other symmetries arise. For example, in some instances hospitals may present new symmetry. In more than one dataset we have hospitals offering only one firm. If the firms in such hospitals are indistinguishable, as happens when they have the same number of places available, offer the same speciality and have no or identical consultant clashes, a new type of symmetry arises between hospitals. When studying a particular solution, we may ignore this hospital symmetry to concentrate on the symmetry between the firms within those hospitals.

When investigating the lowest cost and poorer solutions, this inherent symmetry will make it more difficult to decide whether two solutions are fundamentally identical or whether the differences between them are large or small. When looking at a solution before and after local search is implemented, for example, the number of actual differences between the solutions may be large, although when symmetry is removed they may actually be more similar than at first appeared.

One of the consequences of the symmetry in the medical student scheduling problem is that using feedback is not viable. Since two equivalent solutions may comprise very different particular assignments, feeding back the individual success of solution components becomes redundant; an equivalent assignment in one solution is likely to be completely different from that of another. Thus feeding back information on which to base further constructions cannot be helpful. As we will see in Chapter 6, this is one of the reasons why ACO will not be applied to this problem.

2.4 Comparison of the two problems

The nurse scheduling and medical student scheduling problems are both highly constrained. In the case of nurse scheduling each nurse must be assigned to a set of shifts such that none of the hard constraints are violated. The medical student scheduling problem seems more complicated at first glance, since students are assigned to firms in five different timeslots, whereas for the nurse scheduling problem we merely assign each nurse one shift pattern.

Although the two problems seem relatively different, they share many similarities and one of the reasons why it is interesting to study them together is in the fact that construction ideas applied to one problem may be carried over to the other. The particular constructive heuristics applied to each problem will be presented in Chapters 4 and 6 and the analogies between them will be discussed. Furthermore, the actual structures of the two problems are more similar than is at first apparent. The idea of assigning a nurse to a shift pattern is, after all, an idea manufactured from the original problem; the nurses are each to be assigned to a number of individual shifts, on different days. By combining the soft constraint costs into a single cost, and enumerating the possible shift patterns, the problem has become rather simpler to deal with than regarding individual shifts. Furthermore, since Dowsland and Thompson (2000) have shown that assigning nurses to early or late shifts, if they are known to be working days, is relatively simple, the problem becomes one of assigning the nurses to the correct number of 14 'timeslots', with each timeslot receiving either a '1' or a '0' depending on whether the nurse is working or not; timeslots 1-7 then represent the seven day shifts, while timeslots 8-14 represent the 7 night shifts, as explained earlier.

Originally we would have had seven timeslots, one for each day of the week, and four choices for each nurse for each timeslot: early shift, late shift, night shift or day off. This is a similar idea to the medical student scheduling problem, where, for each of five 'timeslots', each student must be assigned to one of a number of possible firms, although the particular constraint sets involved are obviously very different. However, by reducing the nurse scheduling problem to one of a binary nature, and enumerating all possible shift patterns and associated cost for each nurse, the problem definition

and approach has been greatly simplified and so no longer bears the same resemblance to medical student scheduling.

We have mentioned the discrepancy between the two problems in the way that the cost of the soft constraints is calculated: while the nurse scheduling problem has single costs for each shift pattern, the medical student scheduling problem must necessarily calculate the costs during the scheduling process or once the timetable is completed. This difference does more than affect the time required to compute the costs involved, but also provides a fundamental difference in the way an algorithm is able to judge the success of a partial solution. For the nurse scheduling problem, a simple lower bound for the preference cost may be calculated easily, by summing the minimum preference cost possible for each nurse and a partial solution may easily be judged in a similar manner – an idea of the cost involved may be gleaned with every assignment. Note that there is a large variation between the nurses in terms of their shift patterns and the preference costs for these patterns. Within a nurse's list of feasible patterns, the preference costs are also very variable and although the feasibility of a solution may not be determined until the solution is complete, this variability within the preference costs for each nurse is likely to mean costs are incurred relatively early in the scheduling process, giving something on which to base the choice of one pattern over another. With medical student scheduling, this is not the case. For the initial allocations, no costs will be incurred and so there is no way to tell the quality of the partial solution; many assignments must be made before the first cost is incurred. It may be difficult, then, to steer the construction towards good areas of the solution space since it will not be known where these are.

A further comparison of the two problems is in the underlying feasibility structures. Although it is possible to force the medical student schedules to be feasible, by severely restricting each student's permutation of specialities, we have explained why this would not be a sensible course of action and for neither this problem, nor that of nurse scheduling, is there a simple way to guarantee feasibility whilst retaining the ability to explore the full search space. However, for each of these problems, as will be shown in their respective chapters, there is an underlying structure which enables the hard constraints, or at least a relaxed version in the case of the nurse scheduling problem, to be modelled as an IP which may be solved exactly with nominal

computational expense. These models can then be used to guide the construction of solutions, so that the algorithm is steered away from undesirable, infeasible regions of the solution space. For medical student scheduling, the use of the network flow model will be shown to force the construction of feasible solutions, whereas for the nurse scheduling problem, incorporating the knapsack model provides assistance in the search for feasibility, but provides no such guarantee.

We have already mentioned that finding feasible solutions to the nurse scheduling problem has been difficult for all previous researchers, and this will be discussed in more detail in Chapter 3, but the difficulty of finding feasible solutions for the medical student scheduling problem, without the use of a restrictive system, is unknown. While both of these problems may have the feasibility of their schedules improved by such assistance, the implementation of such an aid is likely to be in direct competition with the final quality of solutions; it will be interesting to see the results of these experiments with feasibility assistance for the two problems in Chapters 4 and 6, especially since finding feasible nursing schedules is known to be difficult, while finding feasible medical student timetables has been shown to be relatively easy.

2.5 Conclusions

This chapter has served to introduce the two scheduling problems investigated in this thesis and it has been shown that these problems are similar enough to be able to approach them using the same types of construction techniques. Both also have a structure that enables at least some of the feasibility constraints to be modelled exactly using an IP formulation and so it is known that feasibility may be improved for each problem once such an implementation is employed.

The problems have two conflicting objectives: to create schedules which are feasible whilst also maintaining a high solution quality. Balancing these two objectives will be a key part of the research and will feature heavily in the solution approaches. However, the medical student scheduling problem, with its many separate objectives, also allows investigation into resolving balance other than that between feasibility and optimality.

In terms of solution approaches, the nurse scheduling problem offers a problem more amenable to construction and improvement type strategies, since it has smaller size, more variation between solution components and a much lower order of symmetry; this lack of symmetry means an improved chance of successfully implementing feedback. The medical student scheduling problem offers an opportunity to solve a much larger, more homogeneous, problem with many conflicting constraints where solution techniques such as feedback may not practically be employed and construction and local search techniques must be more heavily relied on.

The known optimal solutions to the nurse scheduling problem will make initial evaluation of solution methods easier and the previous research will give a guideline as to the relative success of each approach, while the medical student scheduling problem allows the chance of tackling a previously unsolved problem.

The next chapter will give a review of the available literature, before going on to deal with solution methods to these problems in more detail. Note that the notation introduced in this chapter is summarised in Appendix D.

Chapter 3

Literature review

3.1 Introduction

This chapter provides a background to the work presented in this thesis and places it within a wider context. Two problems are considered: the nurse scheduling problem and the medical student scheduling problem. The nurse scheduling problem, described in detail in Chapter 2, has been solved using a variety of solution approaches. However, this is just one of a number of nurse scheduling problems and this chapter will give details of the problem in its most general form and the variants which may be encountered in the literature. The medical student scheduling problem, on the other hand, is a new problem, not previously found in the literature, but a discussion of the problem structure and some of the particular similarities and differences with the nurse scheduling problem can be found in Chapter 2, along with a discussion of its similarities with other problems in the literature.

This thesis uses two metaheuristic approaches: greedy randomised adaptive search procedure (GRASP) and ant colony optimisation (ACO). This chapter will give an

overview of these methods, their variations and extensions, and the relevant problems to which they have been applied previously.

The remainder of this chapter is organised as follows. The next section gives details of the GRASP and ACO metaheuristics, applied in this thesis. This is followed by a discussion of the nurse scheduling problem in all its guises in Section 3.3. The approaches applied to nurse scheduling in the past are discussed in Section 3.4 and in Section 3.5 the chapter is concluded with an in-depth look at the research which has been done on the nurse scheduling problem presented here.

3.2 A history of methods applied in this thesis

The two scheduling problems presented in this thesis are highly constrained. The nurse scheduling problem presented here has been solved with metaheuristic approaches in the past, as will be discussed in Section 3.5, but this thesis presents the first Greedy Randomised Adaptive Search Procedure (GRASP) approach used to solve the problem. Ant Colony Optimisation (ACO) is another method never previously applied to the problem presented here, although it has been used to solve an Austrian nurse scheduling problem, as will be discussed in Section 3.4.3.1. To our knowledge, this thesis also presents the first attempt to solve the medical student scheduling problem using a metaheuristic approach. Again, a GRASP approach is exploited and Section 3.2.1 will show that it has previously been successful in solving one of the closely related classical problems in the literature, the three-index assignment problem. Although the original intention was to apply ACO to the medical student scheduling problem as well, later observations indicated that this would not be a sensible approach. In Section 3.2.2 we discuss how the ACO is particularly suited for problems whose solutions may be represented as permutations and how applying it to other types of problems, such as nurse scheduling and medical student scheduling is inherently more challenging.

This section provides details of the GRASP and ACO metaheuristics as well as an overview of the way in which they have been successfully applied to problems in the literature.

3.2.1 Greedy Randomised Adaptive Search Procedure (GRASP)

Greedy Randomised Adaptive Search Procedure (GRASP) was first introduced by Feo et al. (1994) and is an iterative or multi-start procedure (Resende 2001), in which each GRASP cycle is composed of two distinct parts. First, a solution is constructed randomly with some greedy element and then a local search is used to improve this initial solution in the second stage. The variety of the constructions in the first part of the procedure allows for several local optima to be reached in the final set of solutions. The best of these solutions is kept as the end result. Resende (2001) views GRASP as a repetitive sampling technique, with each GRASP cycle resulting in one solution from the space of all possible solutions.

The first phase of the GRASP algorithm is the greedy randomised construction which, according to Corberán et al. (2002), can be traced back to Feo and Resende's heuristic approach for solving a set covering problem (1989), although Resende likens the construction to Hart and Shogan's slightly earlier research (1987). During the construction, the solution is built piece by piece, by selecting the next element myopically. To make a random greedy choice at each stage, a selection is made from a 'restricted candidate list' (RCL) comprising only the best elements, where one element is judged to be 'better' than another if its score in some appropriate evaluation function is higher, for example. The majority of GRASP applications make the selection from the list probabilistically, according to this score. The general methodology behind the scoring is to calculate, for each potential element to be selected, the immediate impact on the solution quality of making such a selection. Note that GRASP is adaptive because the function calculating the scores for each element is adaptive, and is based on the current partial solution. The particular heuristic used to create the scores will obviously vary and is generally problem-specific. The second, improvement, phase of GRASP, local search, is also an important consideration. Since the constructions are not guaranteed to be optimal, the application of local search is "almost always beneficial" (Li et al. 1994, Feo and Resende 1995, Resende 2001).

The key to providing an effective GRASP heuristic is combining the good constructions from the first GRASP phase with a suitable local search. Lourenço et al.

(2001) state that "the most important concept in a local search is the definition of the neighbourhood for the problem under consideration" while Blum and Roli (2003) point out that it is important to choose a local search which 'fits well' with the construction phase, in order to be able to reach many different local optima.

A full survey of all GRASP literature is beyond the scope of this work, but overviews of GRASP can be found in Feo and Resende (1995), Resende (2001), Pitsoulis and Resende (2001) and Resende and Ribeiro (2002). Bibliographies can be found in Resende and Festa (2003) and Festa and Resende (2004).

3.2.1.1 Construction Phase

The heuristic used in the construction plays a large part in the success of the GRASP algorithm. Li et al. (1994) and Feo et al. (1991) note that GRASP is an improvement on a random multi-start local search; many GRASP cycles may be completed in the time taken for local search to converge given a random starting solution. In particular, Resende and Ribeiro (2002) apply GRASP to a simple combinatorial optimisation problem and show that the local search as applied to a near-greedy construction converges to a local optimum approximately 2.5 times more quickly than when the process is initialised with a purely random construction.

How greedily the solutions are constructed in this first phase, depends on the number of candidates, n, allowed in the restricted candidate list (RCL). If $n = 1$, the construction heuristic will always choose the best, or highest-scoring, value, meaning that the same solution will be produced in every cycle. This is likely to lead to relatively high-quality, but suboptimal, solutions. Choosing a larger value of n allows for more variation and, although this will be likely to worsen the quality of the solutions produced on average, the variation means that best solutions are likely to improve on those produced using a purely greedy heuristic (Feo and Resende 1995, Resende 2001). The RCL can be compiled in different ways. The first is to allow the RCL to be composed of the best n elements, as mentioned before, where n is some parameter to be decided. For example, Feo et al. (1996b), Corberán et al. (2002) and Casey and Thompson (2003) are among those using this approach. Another way to determine the RCL is to allow all elements whose scores are within some percentage

α of the greedy choice (Li et al. 1994, Feo and Resende 1995, Mavridou et al. 1998). Klincewicz (1992) combines these approaches, using a fixed maximum list size, but potentially reducing this size by disallowing all elements whose score is below a percentage of the best, and finds that the size of the RCL does have an impact on solution quality. Feo and Resende (1995) also present this as a possibility. Liu et al. (2000) also use both ways of determining the size of the RCL when solving a frequency assignment problem; they choose a vertex from a list of fixed size n, but assign it a frequency by choosing from all frequencies whose cost is within a given percentage of the smallest possible. Aiex et al. (2005) create a candidate list from the best percentage, α, but let α be a parameter whose value is selected randomly at the start of each construction. As well as determining the size of the RCL, there is variation within the literature as to how to select from it. Although the original GRASP approaches sample uniformly from this list, for example Corberán et al. (2002) and Resende and Ribeiro (2002), many researchers select from the list probabilistically, using a roulette wheel selection method, for example Binato et al. (2002), Kim and Park (2004).

Once the framework of the GRASP construction phase has been decided, there are several ways which have been introduced to improve on the basic construction method. Hart and Shogun (1987) detail several approaches for improving heuristic effectiveness and in the context of a GRASP construction discuss the virtue of using a fixed candidate list rather than sampling from all candidates whose cost is within a given fixed percentage of the best available. They conclude that while using the variable list size produces lower cost solutions than an equal number of cycles using a fixed list size, the fixed list, using a value of $n = 2$, was more robust. There have been many examples of improvements to the GRASP construction. Robertson (2001) presents two heuristics for producing the restricted candidate list for a multidimensional assignment: the first is a straightforward randomised greedy approach, but the second, *maximum regret*, is more intelligent; the difference in cost between the best and next best possibilities for a selection is used to create the candidate list. When a selection is made, the best cost assignment for this element is then accepted. Feo et al. (1996b) use a similar construction idea in a GRASP approach to a single machine scheduling problem. However, their score function associated with an unscheduled job j is made up of both a standard greedy component regarding

the changeover time cost as well as an 'opportunity cost' which determines the cost of inserting j after half the unscheduled jobs have been scheduled. Thus a balance is created between selecting greedily and looking ahead to how this will affect the rest of the schedule.

Given the time required to perform local search, many researchers place more importance on providing their algorithm with a good construction. Kontoravdis and Bard (1995), for example, apply local search only to the best of each set of five constructed solutions and Rojanasoonthon and Bard (2005) also restrict the application of the improvement phase to only the most promising of the constructed solutions. Carreto and Baker (2002) only apply local search if a feasible solution has been obtained by the construction. Note that Robertson (2001) concludes that the benefits of the local search phase are outweighed by the computational expense and Argüello et al. (1996) omit the improvement phase altogether. This indicates the importance of the construction.

One of the features Fleurent and Glover (1999) add to the GRASP construction for the QAP is the idea of correcting or improving a current partial solution before making the next assignment. In so doing, they hope to avoid one of the common problems arising from constructive procedures: that of "wrong choices" being made at an early stage in the construction, leading to additional wrong moves and resulting in an overall poor solution. Generally, correcting a completed poor solution is very difficult due to the non-linearity of the problem. They suggest two methods to ease this problem: firstly, to use a local search on the current partial solution; and secondly to oscillate between constructive and destructive moves, by periodically removing some of the most expensive assignments. Binato et al. (2002) provide another example of local search being applied periodically on partially constructed solutions, this time for a job-shop scheduling problem. Laguna and Martí (2001) describe a GRASP implementation for the graph colouring problem. The construction phase builds up the colour classes one at a time, completing one colour class before moving on to the next. Rather than just allowing a single attempt at each colour class several attempts are made and the best, according to a suitable criterion, is selected.

However, aside from implementing enhancements to a basic GRASP construction, much research has shown that utilising different construction heuristics can be beneficial. Ribeiro et al. (2002), for example, successfully employ a combination of different construction heuristics available for the Steiner problem in graphs along with strategies to intensify, diversify or merely perturb solutions, as necessary. Robertson (2001) showed that two construction heuristics applied to the multidimensional assignment problem gave large differences in solution quality.

Resende and Feo (1996) introduce a GRASP approach to solve a satisfiability problem, determining whether an assignment of values can be found for a set of logical variables, such that the conjunction of clauses containing these variables will be true. The constructions vary according to: whether or not the variable size of the RCL has a maximum; whether both the number of unassigned variables and unassigned clauses are used to create the score function or whether only the clauses are considered; and how the candidates are chosen for the candidate list when a maximum list size is employed. They found that all options considered found the necessary assignment, but that there were differences between them in terms of computational expense, with the fastest construction being the one with no restriction on the list size and considering only the clauses in the score function. Delorme et al. (2004) use a GRASP approach to solve a set packing problem where a number of variables must similarly be set to either 0 or 1 to satisfy certain constraints, in order to maximise the total value of the variables set to 1. The first approach initialises all variables with the value 0 and the heuristic selects a single node to upgrade in each stage of the construction, while the second initialises all variables with the value 1 and selects a node to remove 'from the pack' at each stage. The final heuristic includes information about how many times each variable was set to 1 in previous solutions in order to create the scores. The approach developed was most successful when using in conjunction with this third heuristic incorporating a learning strategy.

Andronescu and Rastegari (2003) provide a further example of a GRASP approach where more than one heuristic has been tested, and also conclude that the difference in solution quality between them is discernable.

Gomes and Oliveira (2001) use GRASP to solve a nesting problem, a two-dimensional cutting and packing problem where irregular shapes are to be packed onto a plate of fixed width and infinite length. The first heuristic presented considers only the length added to the solution for each assignment, whereas the second uses the wasted space added as a measure of fitness. The wasted space was chosen as the more practical of the two, since there was no way to compare two assignments for which neither increased the overall length.

There are therefore several examples in the literature where there has been a choice for the construction heuristic employed and where it has been shown that this choice is important to the overall quality of the solutions. We are especially interested in constructions which consider different ways to balance different weights, as this is an important part of our research. Pardalos et al. (1999), for example, solve a set-covering problem relating to a directed graph. To select and order vertices, which is normally done according to the degree of each vertex, Pardalos et al. consider the degree in terms of the number of edges entering and the number leaving each vertex. Three simple ways of combining these two aspects into a score are tested, with the multiplicative score producing slightly better results, on average than an additive score and one taking the maximum of the two values.

Note that in many cases obtaining a feasible solution within the GRASP construction is not an issue because feasibility can be built into the form the solution takes. Voß (2000) notes that a feasible construction can easily be obtained for problems such as the QAP and TSP, whose solutions may be represented as permutations. For other GRASP applications, simple steps can be taken to ensure the constructions are feasible, by similarly disallowing elements on the RCL which will violate necessary criteria (Rojanasoonthon and Bard 2005, Skorin-Kapov and Kos 2006, for example). There has not been a great deal of research applying GRASP to problems where the hard constraints cannot easily be satisfied, however, Atkinson (1998) presents one such example. He uses a feedback approach for a vehicle routing problem which involves making locations unvisited in previous cycles more likely to be assigned to a vehicle earlier in the search. Note that the fact that there has previously been little research in this area shows that there is room in the literature for an investigation of the sort presented in this thesis.

3.2.1.2 Improvement Phase

As was mentioned previously, in most instances the improvement phase of GRASP is a necessary inclusion since the construction phase alone cannot usually guarantee locally optimal solutions; Feo et al. (1991) assert that most solutions created by a GRASP construction could be improved by a simple local search application. Although the neighbourhoods employed must necessarily depend on the particular problem being studied, for most problems the easiest implementation of local search would take the form of applying a simple 1-opt or 2-opt neighbourhood. However, most employ a more sophisticated local search, and we now discuss some of the types of enhancements which have been used in conjunction with a GRASP approach.

A simple way in which the local search can be enhanced is to employ more than one neighbourhood and several cases of this can be found in the literature. Rojanasoonthon and Bard (2005), for example, solve a problem in which several jobs must be scheduled onto a number of non-homogeneous machines with time windows and five simple neighbourhoods are employed, successively. These are: relocating up to three jobs on the same machine, relocating up to three jobs on a different machine, swapping two jobs which are on the same machine, swapping two jobs which are on different machines and, finally, swapping a scheduled job with an unscheduled job. Although each neighbourhood definition is simple, by employing all five neighbourhoods, the space of solutions which can be searched is much greater than could be reached by employing just one or two. There are many other cases where local search has been employed with more than one neighbourhood and examples of these include Feo et al. (1991) and Xu and Chiu (2001).

Often, GRASP will be used with a more sophisticated local search technique in order to improve solutions more efficiently. Li et al. (1994) for example, present a new neighbourhood for the QAP, which may be applied to any other problem with a permutation representation, such that any pair of elements in the permutation may be exchanged provided that at least one of these has not yet been altered in the current local search phase. They determine that, in general, this neighbourhood allows better solutions to be found than the standard neighbourhoods used in conjunction with this problem. Other research into local search enhancements provides larger and more

complex neighbourhoods, such as those used in Lourenço and Serra (1998). They introduce a variable depth neighbourhood for the generalised assignment problem, capable of both simple 1-opt moves, as well as 'ejection chain' moves, which perform a 1-opt move in conjunction with another, potentially balancing 1-opt type move. That is, moving task l from agent i to agent j as well as moving task m from agent k $(k \neq i)$ to agent p, where $p \neq j$, but $p = i$ is allowed. Casey and Thompson (2003) offer another example of a local search which incorporates chains of moves within the neighbourhood definition.

Other general techniques for enhancing the improvement phase involve hybridising the GRASP with another search-based metaheuristic, such as simulated annealing (Liu et al. 2000, Laguna and Martí 2001), tabu search (Laguna and González-Velarde 1991, Lourenço and Serra 1998, Areibi and Vannelli 2000, Laguna and Martí 2001) or variable neighbourhood search (VNS) (Beltrán et al. 2004).

3.2.1.3 Other variations

The basic idea of a GRASP approach is a very simple one and one which can be applied to a variety of problems with relative ease (Li et al. 1994). However, due to the complexity of many problems, such a simple approach is not always sufficient to find optimal or even close to optimal solutions. This section has so far discussed GRASP in its most basic form as well as some of the enhancements made to the construction and improvement phases in order to improve the solutions obtained by a GRASP approach; many further examples of enhanced GRASP algorithms exist in the literature.

GRASP is a multi-start procedure which can be viewed as a repetitive sampling technique (Resende 2001) as mentioned previously in this section. However, although this means that GRASP provides a diverse range of solutions, it has been suggested that solution quality could be enhanced by introducing memory strategies, utilising some of the information gained from previous cycles rather than treating them independently. One of the common memory strategies used in conjunction with a GRASP approach is that of path-relinking. This generally involves intelligently

changing an existing high-quality solution s^1 into a second high-quality known solution s^2 and thus creating a path of solutions between the two. By monitoring the modifications made during this process, it is suggested that solutions better than these elite solutions may be encountered along this path and thus a new elite solution will have been found. Laguna and Martí (1999) were the first to develop a hybrid GRASP with path-relinking approach and it has since been applied to job shop scheduling (Aiex et al. 2003), the QAP (Oliveira et al. 2004), workover rig scheduling for onshore oil production (de Aragão Trindade and Ochi 2005), set packing (Delorme et al. 2004) and the axial three-index assignment problem (Aiex et al. 2005, Fügenschuh and Höfler 2006). As mentioned in Chapter 2, the planar version of three-index assignment problem shares similarities with the medical student scheduling problem. The constructions used by both Aiex et al. (2005) and Fügenschuh and Höfler (2006) involve evaluating the cost for each individual assignment triple (i,j,k) and selecting the best of these to assign. Both employ approaches which take advantage of the fact that each element in a set can only be assigned once. Note that this type of approach could not be applied to medical student scheduling for two reasons: firstly, each assignment triple (s,f,t) does not have a single, known cost which contributes linearly to the total solution cost and secondly, the sets of students, firms and timeslots are not of equal size and each student, firm and timeslot will necessarily appear more than once in any solution. Further, the local search phase of Aiex et al.'s GRASP algorithm (2005) relies on the permutation representation of solutions, something which we have established is not suitable for the medical student scheduling problem. The path-relinking phase, also adopted by Fügenschuh and Höfler (2006), would be difficult to implement for our problem, due to the problem of symmetry, as mentioned in Chapter 2. However, most instances of path-relinking in the literature have had very successful results, especially in terms of savings to overall computational expense.

Many other ways to introduce memory to a GRASP approach have been supplied in the literature. Atkinson (1998), for example, used GRASP with a diversifying memory function to solve a vehicle routing problem with time windows. For this problem it was not trivial to obtain a feasible solution where all locations were allocated to a vehicle. In order to use information from the previous cycles to encourage feasibility, a 'priority' cost PR_i associated with location i was modified at

the end of each cycle in proportion to the number of previous solutions in which i failed to be scheduled successfully. The construction stage of the GRASP then incorporated this information to encourage locations with a high PR_i value to be scheduled earlier. Fleurent and Glover (1999), on the other hand, use memory as a form of intensification strategy; they use information from an elite set of solutions to guide the construction process for a GRASP approach to the QAP. Particular pairs of assignments which exist in a large number of the elite solutions are more likely to be selected for the current partial solution. The influence of this information over that of the standard evaluation function increases with time, since initially the solutions in the elite set are not known to be of a high quality. Binato et al. (2002) apply the same approach to a job-shop scheduling problem and a similar adaptive memory approach is utilised in Ahmadi and Osman (2004) in response to a capacitated clustering problem.

The combination or hybridisation of GRASP with other algorithms has proved extremely successful and the literature is full of examples where GRASP has been successfully improved by careful modifications of this nature, as has already been discussed. In some cases, GRASP has been effectively hybridised with more than one other approach, and examples of this include Gupta and Smith's (2005) GRASP approach for a single machine scheduling problem which incorporates both path-relinking and VNS and Piñana et al.'s (2004) GRASP approach which incorporates path-relinking and uses tabu search in the improvement phase. Ribeiro et al. (2002) also use a GRASP with path-relinking and add a weight-perturbation strategy in the construction phase, using this weighting strategy to strategically oscillate between a bias towards intensifying, diversifying or merely randomly perturbing the construction of solutions. Casey and Thompson (2003) solve an examination scheduling problem, enhancing the basic GRASP algorithm using memory functions in the construction phase to intelligently diversify or intensify the solutions as necessary and simulated annealing with a neighbourhood definition including chains of moves in the improvement phase.

In its most basic form, GRASP is a method which is easily applied to a variety of problems, often with very good quality results in comparison to other methods of equal simplicity and with much reduced computing time over exact methods, should

they be available. GRASP has been applied successfully to a wide variety of problems, including assignment problems (Li et al. 1994, Feo and González-Velarde 1995, Resende and Pardalos 1996, Mavridou et al. 1998, Lourenço and Serra 1998), scheduling problems (Feo et al. 1991, Feo et al. 1996a, Feo et al. 1996b, Xu and Chiu 2001, García et al. 2001, Kim and Park 2004), graph theory (Laguna and Martí 2001, Corberán et al. 2002), location problems (Klincewicz 1992), and packing problems (Moura and Oliveira 2005). However, when enhanced using some of the methods discussed previously to enhance both the construction and improvement phases and with other, more sophisticated modifications and hybridisations, very robust and successful approaches arise, which are generally able to compete and improve on other available approaches.

3.2.2 Ant Colony Optimisation (ACO)

The Ant Colony Optimisation method (ACO), introduced in Dorigo et al. (1991) and Dorigo (1992), aims to emulate the system by which the Argentine ant (*Linepithema humile*) searches for food (Taillard 1999). This technique is an automatic process which enables a colony of ants to quickly converge their various paths between the nest and food source to the one which has the shortest distance. This process is the result of a feedback loop created by the ants laying down pheromone as they walk and being more likely to follow a path which is rich in such trail. Ants following a shorter route will place their pheromone trail on it more frequently and so later ants are more likely to detect and adopt it. A simplified version of this process is illustrated in Figure 3.1.

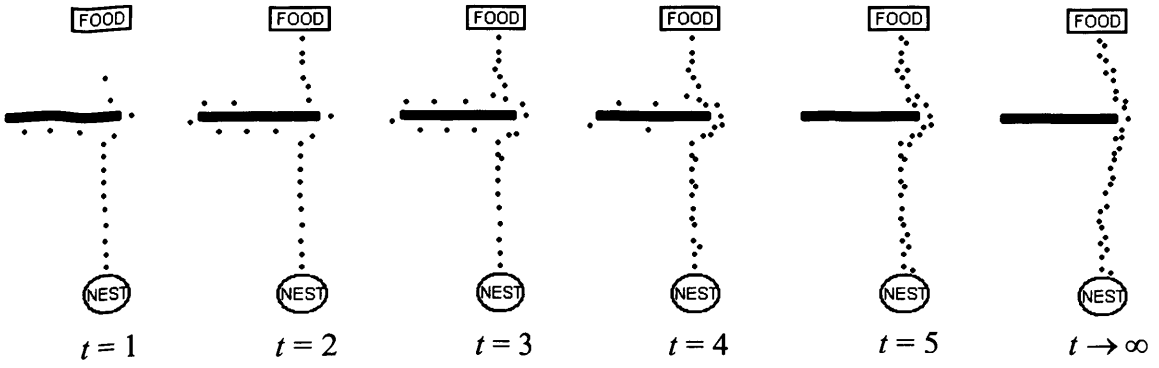


Figure 3.1. An example demonstrating the effects of pheromone reinforcement on ant path-selection over time.

At time $t = 1$, the ants set out from the nest towards food. As they reach an obstacle, each ant has an equal probability of going either left or right. All ants leave pheromone trails as they travel and the pheromone trails are roughly equal in both directions. At time $t = 2$, some of the ants which chose the shorter route have already reached the food and are heading back to the nest, while those which chose the longer route have yet to reach the food. At time $t = 3$, the first ants which chose the shorter route have travelled back past the decision point, reinforcing the pheromone trail in that direction as they pass and causing a feedback loop. The trail is now stronger on this path and an ant reaching the decision point is more likely to choose this path over the longer one. At time $t = 4$, ants which had travelled to the food by the longer route may also return by the shorter path due to the increased amount of pheromone on this path. New ants reaching the decision point are now much more likely to take the shorter route than the longer one. By time $t = 5$, all ants travel the shortest way around the obstacle and, eventually, the pheromone trails lead all ants the shortest way to the food. The most natural use for an ant algorithm would be to solve a routing problem since this is directly applicable. The Travelling Salesman Problem (TSP) is the simplest of the routing problems and many of these problems have been solved successfully using ACO. Given n cities, a salesman starting from any city must visit each city exactly once before returning to the start point. The ACO algorithm, modified to deal with the discrete case, works in the following manner. The population of artificial ants is each given a starting location. For an ant at location i, a score, $score(i,j)$, is assigned to each location j to which the ant may now move, given by

$$score(i,j) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{k \in K}\tau_{ik}^{\alpha}\eta_{ik}^{\beta}}, \tag{3.1}$$

where η_{ij} is the visibility score between i and j, indicating the "closeness" or "desirability" of location j, τ_{ij} is the trail score between i and j and K is the set of all feasible locations to which the ant may still travel. The city chosen is selected probabilistically from these scores.

The ACO method has many variations – mostly to do with the way the trails are updated. In Ant System (AS), the original ACO algorithm introduced by Dorigo (1992), the trail matrix is updated globally after all ants a have completed a full tour,

T^a. For each tour T^a, the set of arcs (i,j) which were visited during the tour add $\Delta\tau_{ij}^a$ to the trail so that each element of the matrix is updated using

$$\tau_{ij} = (1-\rho).\tau_{ij} + \sum_a \Delta\tau_{ij}^a \tag{3.2}$$

where $\Delta\tau_{ij}^a = \begin{cases} Q/c^a \text{ if } T^a \text{ contains arc}(i,j) \\ 0 \quad \text{otherwise} \end{cases}$,

ρ is the evaporation rate, Q is some constant and c^a is the total distance of tour T^a (Dorigo et al. 1991, Dorigo 1992). Basic enhancements to AS include the idea of an elite ant, where the best tour so far will contribute more strongly in the trail update (Dorigo et al. 1996). AS_{rank} is another improvement to the basic AS strategy, using the idea that each ant's tour contributes to the trail matrix in proportion to the rank of that tour, rather than in proportion to the actual tour cost (Bullnheimer 1999b). All these ideas provide slight variations on the basic AS method.

Ant Colony System (ACS), however, differs from AS in three main ways. Firstly, by the way in which the next city in the tour is selected; at each decision point, there is a chance of being allocated the best, or nearest, city, controlled by a parameter $q_0 \in [0,1]$. If a random variable in the range [0,1] is greater than q_0, the next city to be chosen is the closest, otherwise all cities not yet visited are considered as normal. Secondly, the trail matrix is not just updated globally, after all tours have been completed, but also locally; after every selection (i,j) the trail matrix is updated according to

$$\tau_{ij} = (1-\rho).\tau_{ij} + \rho.\tau_0, \tag{3.3}$$

where τ_0 is the initial trail value, usually a small constant. This has the effect of diversifying by using the trail to discourage the use of arcs which have been previously visited. The third and final main difference between AS and ACS is in the global update at the end of each cycle; only arcs from the best tour found so far are used to update the trail. This has an intensifying effect, encouraging the use of these low-cost arcs in future solutions. ACS was introduced by Dorigo and Gambardella

(1997). ACS is based on the slightly earlier algorithm Ant-Q (Gabardella and Dorigo 1995) which varies only in the value for τ_0. Where ACS takes τ_0 to be a constant, the earlier Ant-Q calculated a value for it relative to the potential distances travelled by the ants in their initial journey. However the simpler version, where τ_0 is constant, proved just as effective and so Ant-Q was abandoned in favour of ACS (Dorigo and Stützle 2004). A final example of ACO is the *MAX-MIN* Ant System (MMAS) (Stützle and Hoos 1997, 2000), the major feature of which is strongly reinforcing the best tours found, while applying maximum and minimum limits to the pheromone trail values, as the name suggests. Using this algorithm, the trail matrix is initialised with the maximum value and is reinitialised each time the system stagnates. As has been mentioned, ACO is most directly applicable to the TSP and other routing type problems and it will be shown that any other problem whose solutions have a permutation representation can apply ACO similarly. In the next sections, we give details first of how ACO has been successfully applied to the TSP and related routing problems, before describing how it has been applied to other similar problems and, finally, other problems where a permutation representation cannot or has not been exploited.

3.2.2.1 ACO applied to the travelling salesman problem

The TSP is probably the problem to which ACO algorithms have been most often applied. Dorigo et al. (1991) applied several variations of AS on the TSP and produced good results. They found that the algorithm is sensitive to changes in parameters, with the exception of the parameter Q, which they found to have a negligible influence on solution quality. Gambardella and Dorigo (1995) used Ant-Q to solve the TSP as well as using ACS to solve both the symmetric and asymmetric TSPs (Gambardella and Dorigo 1996). Dorigo and Gambardella (1997), applied ACS to the TSP and showed that ACS performed well compared with other heuristic methods. Bullnheimer et al. (1999b) introduced AS_{rank} and applied it to the TSP with promising results. Stützle and Hoos (2000) found that MMAS produced good results for the TSP compared with other ACO approaches, such as AS, AS with elite ants, AS_{rank} and ACS. Bullnheimer et al. (1999a) applied Ant System to the vehicle routing

problem, another problem to which the ant metaphor is easily extended. A review of ACO applied to the TSP can be found in Stützle and Dorigo (1999b).

3.2.2.2 ACO for permutation-based problems

As well as these more obvious applications, ACO has also been applied in its many forms to a wide range of other types of problems, including many scheduling and assignment problems. For problems such as TSP and vehicle routing, it is easy to see how the pheromone trail may be defined and, since a tour is being built, it is very easy to extend the ant analogy to a representation of this problem. For other, seemingly less related, problems a tour representation may still be used. Montgomery et al. (2005) indicate that there is a structural advantage inherent in problems with a permutation representation for the selection of pertinent trail definitions for ACO.

For example, solutions to the graph colouring problem, the problem of colouring a set of vertices in as few colours as possible such that no two vertices joined by an edge may have the same colour, may be represented by a permutation. Although they did not exploit this permutation representation, Costa and Hertz (1997) stated that there exists one or more optimal orderings of the vertices for any graph, such that the minimal colouring may be obtained by simply colouring each vertex greedily in turn. Thus a solution may be expressed as a permutation of the vertices and again a trail representation may be easily defined. Other problems with a permutation representation are similarly suited to an ACO approach.

Gravel et al. (2002) and Ferretti et al. (2006), for example, use permutation-based approaches to solve two continuous casting problems. A sequence of jobs must be scheduled and trail is defined between each pair of jobs (i,j) when job j is immediately preceded in the schedule by job i. Gagné et al. (2006) similarly treat a solution to a car sequencing problem as a permutation of car classes and so are easily able to extend the ACO trail definition for the purposes of their problem.

Merkle and Middendorf (2001) use a different approach to solve a permutation scheduling problem, where jobs must be assigned to a place in the schedule with a 'due date' associated with each job; they alternate between generations of ants using a

standard sequential approach and generations of ants which determine which job will be assigned the next place considering the places in a random order. They suggest that the new generations will better utilise the information in the trail matrix, since every place has the same opportunity to be the first that is assigned a job, while the standard generations of ants will maintain the high levels of quality obtained from the heuristics developed for the problem. They found their algorithm performed well.

The Quadratic Assignment Problem (QAP) involves pair-wise matching of two sets of n objects in order to minimise the total cost associated with all assignments. The nurse scheduling problem presented here is based on a similar idea of creating minimum-cost pairings of nurses with shift patterns. However, since a solution to the QAP is a bijection, it may also be represented as a permutation and so the QAP more easily lends itself to an ACO solution approach.

Gambardella et al. (1999) use a variant of AS to modify existing solutions to the QAP, rather than to build new ones. Each ant is initialised with a randomly generated complete solution, consisting of a permutation of n elements, π. A modification consists of swapping two elements in the permutation, π_i, π_j. The index of the first value, i, is chosen randomly, while the second, j, is chosen in such a way that

$$j = \arg\max_k \left(\tau_{i\pi_k} + \tau_{k\pi_i}\right)$$

with some probability q and j is chosen with probability

$$\frac{\tau_{i\pi_j} + \tau_{j\pi_i}}{\sum_{k \neq i} \left(\tau_{i\pi_k} + \tau_{k\pi_i}\right)}$$

otherwise. The first of these aims to create very good solutions based on the trail matrix, while the second is a more diversifying move.

Once the solution has been modified in this way a specified number of times, a standard local search procedure is implemented and the new solution becomes the starting solution for that ant at the beginning of the next generation. However, in order to intensify the search around good areas, each ant is initialised with its best permutation so far when the best solution produced so far has been improved upon.

When no improvement has been made to the best solution for a given number of cycles, a diversification strategy reinitialises the trail matrix and assigns new starting solutions for all ants other than the one representing the best solution found. In order to speed up convergence, only the pheromone trails relating to the best solution so far are updated.

Results show that this approach performs well on a variety of test problems, especially those with a more irregular data structure.

Maniezzo (1998) applies a modified version of AS to the QAP described as an approximate non-deterministic tree search and denoted by ANTS in the literature. In order to make the algorithm more computationally efficient, the standard multiplicative score function, given in (3.1), is replaced by an additive version, as given in (3.4).

$$score(i,j) = \frac{\alpha\tau_{ij} + (1-\alpha)\eta_{ij}}{\sum_{k \in K}(\alpha\tau_{ik} + (1-\alpha)\eta_{ik})} \tag{3.4}$$

The ANTS part of the algorithm comes from estimating the desirability of a move, η_{ij}, by calculating lower bounds for the cost of the completion of the partial solution containing j. Another feature is that ANTS does not apply the usual evaporation to each trail matrix element. Instead, the trail is updated using

$$\tau_{ij} = \tau_{ij} + \sum_{a} \Delta\tau_{ij}^{a},$$

where $\Delta\tau_{ij}^{a} = \begin{cases} \tau_0 \cdot \left(1 - \dfrac{c^a - LB}{\bar{c} - LB}\right) & \text{if } T^a \text{ contains arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$.

Here LB is the lower bound to the optimal solution cost, $\bar{c}$ is the moving average of the last k solutions, and k is a parameter. This dynamic scaling procedure allows distinctions to be made between small differences in the later stages of the search, when solutions become more uniform. ANTS was found to perform better than GRASP and Tabu search implementations.

Stützle and Dorigo (1999a) give an overview of ACO as applied to the QAP, covering AS, MMAS and many other variations, and apply MMAS with 2 types of local search to the problem, concluding that ANTS (Maniezzo 1998) is the best approach for this problem.

3.2.2.3 Other problems and trail definitions

Although the ACO can be easily applied to the QAP, by using the permutation representation of a solution, many other problems have no such representation, and even when applying ACO to problems which do have a permutation representation, this is not always the trail definition used. Montgomery et al. (2005) suggest that problems without such a representation may require additional heuristic assistance to ameliorate any problems of unfavourable trail bias which may become apparent.

Lourenço and Serra (1998), for example, use a version of MMAS to solve the generalized assignment problem and use a trail between specific pairings. Since they admit infeasible solutions in the search space, the objective function includes a term to penalise infeasibility. By hybridising their method with a tabu search algorithm, they are able to produce excellent results for a number of test cases and found that the modified ant algorithm was able to outperform a GRASP approach with the same modifications.

Chen and Cheng (2005) apply an MMAS approach to a task assignment problem, using a trail definition between pairs of items to be assigned. The algorithm was tested on randomly generated data and was found to be much quicker and more successful than both a GA and an approximation algorithm based on an ILP approach.

Forsyth and Wren (1997) apply AS to a bus driver scheduling problem. They represented their problem as a series of work spells and relief opportunities. By formulating the problem in this way they are able to model it as a fully connected graph, where the relief opportunities form the nodes and the work spells, the edges. The trail is then defined between each pair of nodes. They need to minimise both the amount of uncovered work, to provide a feasible solution, and the number of shifts in the solution, to minimise the number of bus drivers required. Rather than combining

these two factors, they update the trail according to just the amount of uncovered work and, once all work is covered, they change the trail update to reflect only the number of uncovered shifts. However, they were unable to rival results produced using a specialised ILP technique designed for this purpose.

Costa and Hertz (1997) do not use the permutation-based approach to solve a graph colouring problem, but instead create a trail between pairs of non-adjacent vertices; the level of pheromone added to a pair (i,j) reflects the quality of a solution in which i and j received the same colour. Levine and Ducatelle (2004) emulate these trails in an ACO approach to a bin packing problem (BPP). The problem consists of fitting items of different sizes into a number of bins in order to minimise the number of bins required. The trail between a pair (i,j) reflects the success of placing items of size i and size j in the same bin. Ritchie (2003) used a similar approach for a homogeneous multi-processor scheduling problem, essentially a variant of the BPP; pheromone trail between pair (i,j) indicates the level of success of assigning jobs of running times i and j to the same processor. For the heterogeneous variant of the problem, where the processors may not all be identical, however, the problem was no longer one of grouping and the trail between a pair (i,j) was therefore taken as representing the favourability of assigning a job i to a processor j.

Dowsland and Thompson (2005) improve the ant algorithm ANTCOL, introduced by Costa and Hertz (1997), and adapt it for application to examination scheduling, a similar problem. By making a modification to the trail updates, in order to better differentiate between high- and lower-quality solutions, increasing the trail values for more "difficult" vertices, and matching a good heuristic with suitable combinations of parameters α and β, they were able to produce the best known solutions for all data sets tested. These results were consistent across a range of trials.

Socha et al. (2002, 2003) apply MMAS to a simplified university course timetabling problem trying trail definitions which matched events to timeslots as well as a similar trail to that of Costa and Hertz (1997) which records details of which events should not be placed in the same timeslot. However, when local search was incorporated, it was found that the first of these trail definitions, matching events to timeslots, gave

significantly better results. Although not directly equivalent, note that this approach of introducing pheromone trail for specific assignments is more similar to the nurse-pattern trail adopted in Chapter 5 than that of using a relative approach.

3.2.2.4 Role of the construction heuristic

Although the construction heuristic, used to calculate the visibility scores η, is an essential part of any ACO approach, there is relatively little in the literature about evaluating different heuristics. For problems with a simple structure, such as the TSP, it is sensible to use the natural construction approach, but for other types of problems, it is sensible to investigate not only enhancements to the trail definition and parameter values, but also the form which the visibility scores, η, take. Although there are plenty of examples in the literature where the success of enhancements such as local search has been evaluated (Stützle and Dorigo 1999a, Levine and Ducatelle 2004 in the above, for example), occurrences of investigation into different construction heuristics and visibility scores are relatively rare. We now discuss some examples in the literature which do investigate this aspect of the ACO algorithm.

Costa and Hertz (1997) apply eight different construction heuristics to their ACO approach to solve a graph colouring problem. The variations are in how the next vertex to be assigned a colour is chosen as well as how the colour or the vertex is assigned. Their results show that applying different heuristics provides variation both in solution quality attained as well as the computational efficiency of the algorithm as a whole. Dowsland and Thompson (2005) apply the same heuristics to an examination scheduling problem, as well as introducing new evaluation functions, determining the success of each solution, feeding back to the trail. They find that the variation in the quality of results obtained by the different approaches is significant enough to claim that the time spent investigating and evaluating the different constructive approaches was worthwhile.

For the nurse scheduling and medical student scheduling problems investigated in this thesis, feasibility is certainly an issue and the bias between the conflicting aspects of feasibility and optimality is the focus of this research. It is interesting to note,

therefore, some of the ways in which similar problems have occurred and been dealt with in the literature.

Although, as mentioned in the introduction in Chapter 1, ACO is not, eventually, applied to the medical student scheduling problem, the way in which the balance between different aspects is attained for other problems is still interesting.

Randall (2004) introduces an ACO approach for the Generalised Assignment Problem (GAP) and remarks that for problems such as this, where feasibility is an issue, it is important to build a heuristic function, η, which deals with these constraints appropriately. The approach taken by Lourenço and Serra (1998) to solve the GAP was to penalise infeasible solutions in the objective function, however, Randall (2004) notes that "it is often difficult to find an appropriate penalty function which works across a range of problem instances and feasible solutions are not guaranteed." Instead, Randall introduces two visibility heuristics to be used in combination: one which favours low-cost assignments and one which favours assignments with higher available resource. The selection of the heuristic applied is based on a "cost to resource ratio" which can either be set in advance or changed dynamically depending on whether or not previous solutions obtained feasible solutions. Thus, the dynamic settings employ a learning strategy to guide the heuristic bias. Results showed that it was important to weight the selection process more in favour of feasibility and Randall suggests that this is likely to be the case for many other tightly constrained problems when using an ACO approach. However, this work was applied in conjunction with a local search and, without this help, it is possible that the heuristic bias in the construction will need to be even more carefully balanced.

Montgomery et al. (2004) also discuss the problem of infeasibility within the search space and note that for assignment problems, the order in which assignments are made is of great importance to the resulting feasibility of the solution; they note that finding ways to assign the most highly constrained items first can be extremely beneficial in this respect. Many of the construction heuristics put forward for the graph-colouring problem by Costa and Hertz (1997), for example, deal with different ways of choosing which vertex to colour next.

The bias given to feasibility in relation to an ACO approach to the nurse scheduling problem will be discussed in Chapter 5.

3.2.2.5 Local search and other enhancements

Some of the different ACO variations were discussed at the beginning of this section, but there have been many further enhancements to ACO in the literature. One of the most notable is the inclusion of local search within the algorithm (Dorigo and Gambardella 1997, Stützle and Hoos 1997, Gambardella et al. 1999, Maniezzo and Colorni 1999, Ritchie 2003, Randall 2004), with the added help from the local search often greatly improving results. Randall (2004) remarks that for problems where infeasible solutions are necessarily included in the search space, the ACO construction can provide a mechanism for building feasible solutions, while the local search can then be used to improve their objective costs. Dorigo and Stützle (2004) also comment that often the very best ACO algoritms are those which combine the ACO constructions with a local search procedure. Other examples of enhancements to ACO include hybridising it with some other metaheuristic. Lourenço and Serra (1998) and Ritchie (2003), for example, hybridise their ACO constructions with a tabu search algorithm and Levine and Ducatelle (2004) apply ACO in combination with an iterated local search (ILS) technique.

For a review of ACO, see Dorigo et al. (1999), Taillard (1999), Cordón, Herrera and Stützle (2002) or Dorigo and Stützle (2003, 2004).

3.3 Nurse scheduling problem variations

Chapter 2 introduced the specific nurse scheduling problem investigated in this thesis; however, the nurse scheduling problem occurs in a number of guises. The next sections will discuss the problem variations and Section 3.4 will detail the solution approaches used to solve them. Since other researchers have previously studied the variant of the nurse scheduling problem presented here, we discuss their solution methods and their relative success in Section 3.5. We begin this section with an

overview of the nurse scheduling problem in its most general form, before examining the problem variations and solution approaches.

As has been mentioned previously, the nurse scheduling problem is one which occurs in all hospitals. Due to the particular service they provide, it is important that the hospitals are not understaffed at any time. The difficulty of this problem is often to balance the hospital's staff requirement with the nurses' individual preferences. The quality of the nurses' schedules is very important to their well being and job satisfaction, as well as the calibre of the healthcare they provide (Naidu et al. 2000, Berrada et al. 1996). The schedules are also important with regards to the hospital budget; Naidu et al. (2000) and Wright et al. (2006) both estimate that nursing costs account for around 50% of total hospital costs.

Since the need for high-quality nursing rosters arises in hospitals worldwide, there is inevitably a large amount of variation in the problem definition and formulation. The contracted number of hours for a nurse will vary from country to country, as will the number of shifts each day is split into, the length of the planning period, the number of levels of nurse seniority and even the nurses' preferences. Even which of the many constraints are considered hard or soft may be variable.

The next sections detail the variations of the problem arising at different hospitals, before the research and solution approaches previously used to solve these problems are discussed in Section 3.4.

3.3.1 Cyclic and non-cyclic scheduling

There are two possible types of scheduling: cyclic and non-cyclic. Cyclic scheduling, as presented in Rosenbloom and Goertzen (1987), is where the same schedule is repeated for each planning period. Nurses rotate through set shift patterns which, in combination, constitute a feasible schedule. Cyclic schedules are often used when the scheduling has to be done manually as they reduce the scheduler's workload. Given that a cyclic schedule only needs to be produced once and will be used for a long period of time, it is worth spending a large amount of time initially to ensure the schedule is of very high quality. Non-cyclic scheduling has to be performed more

often and, as such, it is important that the schedules are created reasonably quickly. A small reduction in solution quality can be accepted if the schedule is produced quickly, especially since personnel unsatisfied with the current schedule may find the next one more favourable. So there is an exchange between spending a large amount of time finding one excellent solution and spending much shorter time finding solutions which are 'good enough'.

However, cyclic scheduling inevitably reduces flexibility and hence the control over satisfying the nurses' preferences and individual requests (Cheang et al. 2003). Furthermore, necessary changes to the schedule may become apparent due to staff illness or leave and the fixed schedule would then need to be reworked anyway, possibly using a rerostering technique as in Moz and Pato (2007). Non-cyclic scheduling, therefore, is generally more desirable and most of the research referred to here will be related to solving a non-cyclic nurse scheduling problem. The approach used by Warner (1976) aimed to incorporate some of the stability provided by a cyclic approach by having a cyclical weekend working policy, as well as the flexibility of a non-cyclic approach by allowing the rest of the schedule to be variable. The nurse scheduling problem presented in this thesis cannot be solved using a cyclic approach since the individual preferences, which vary from week to week, must be taken into account. However, examples of cyclic nurse scheduling problems and solution approaches can be found in Ahuja and Sheppard (1975), Millar and Kiragu (1998) and Bard and Purnomo (2007).

3.3.2 Planning period and shift types

The normal planning period varies from one week (Dowsland 1998, Dowsland and Thompson 2000, Aickelin and Dowsland 2000, 2004, Isken 2004) to two, three or four weeks and up to a month at a time, (Berrada et al. 1996, Valouxis and Housos 2000, Ikegami and Niwa 2003, Bellanti et al. 2004), although some research allows the planning period to be user-defined (Burke et al. 1999). Bard and Purnomo (2005a) present a nurse-rescheduling problem arising from last minute variations in staff requirements and availability and so are only concerned with scheduling the next 24 hours.

For nurse scheduling problems in general, the day-time and night-time shifts for each 24-hour period are treated separately due to practical considerations, enabling, for example, consecutive assignments of a nurse to a night shift followed by a day shift to be disallowed.

The day-time is usually, but not always (Ikegami and Niwa 2003), split further and most research splits the day-time into two (Bellanti et al. 2004, Dowsland and Thompson 2000) or even three separate shifts (Burke et al. 2003a), while the night-time is considered as a single shift; the software Plane (Burke et al. 2003a, Burke et al. 1999), however, allows users to choose the number of shift types. Ikegami and Niwa (2003) also consider a problem where it is the long 16-hour night shift which is eventually split into two separate shifts. Bard and Purnomo (2005b) split each 24-hour period into five overlapping shift types: three consecutive eight-hour shifts overlapping with two consecutive twelve-hour shifts. Isken and Hancock (1991) allow variable shift times; the number of available staff working each of the eight-, ten- and twelve-hour shift types is given, but the start-times of the shifts are unknown. The problem is therefore to determine the numbers of each type of shift which must start at each hour of the day such that the demand is met and each nurse is able to work the correct number of hours in total.

For the problem presented in this thesis, it has been found that achieving a successful balance between the day- and night-time shifts is a more difficult task than filling in the detail for the configuration of the day-time shifts once the day/night distribution has been decided (Aickelin and Dowsland 2000, 2004). However, Berrada et al. (1996) see their three different shifts as corresponding to disjoint problems and so, for each working day, only specify whether a nurse is working or has the day off. For the problem presented in this thesis, nurses are usually contracted for a different number of day shifts than night shifts due to the disparity in the working hours required for each. As such it is not possible to split this problem into three, with one for each shift type.

3.3.3 Constraints

The nurse scheduling problem is highly constrained with staffing requirements, hospital policies, contractual agreements and the nurses' individual preferences to take into account. While ensuring the hospital has adequate staff at all times is usually accepted as a hard constraint (Dowsland and Thompson 2000, Jan et al. 2000, Valouxis and Housos 2000, Burke et al. 2001, for example), that is, one which must not be violated for the schedule to be feasible, some hospitals' regulations place a stronger importance on other contractual requirements and treat these staffing requirements as soft constraints (Bellanti et al. 2004, Berrada et al. 1996). Jan et al. (2000) solve a nurse scheduling problem where the binary variables representing the assignments are of the form x_{ikw}, where i is the nurse index, k is the day index and w is the index of the particular shift worked (w = 1, 2 or 3) and an interesting feature is that the cover requirement for the day-time shift (w = 1) is given as a minimum, but the demand for the other two shift types must be satisfied exactly.

The constraints applied in any particular case vary, but the most common ones are listed in Cheang et al. (2003). These include nurse workload, nurse requests, sufficient time between shifts and staff demand. Burke et al. (2003a) include further personal constraints such as staff not allowed to work without their mentor and staff who should not be allowed to work together.

3.4 A history of nurse scheduling solution approaches

The nurse scheduling problem has been tackled using a wide variety of methods, including mathematical programming, AI and heuristics (Cheang et al. 2003). Although detailing all previous research on nurse scheduling problems is beyond the scope of this thesis, we present here some of the more relevant methods which have been applied to nurse scheduling in the past. General overviews of nurse scheduling can be found in Cheang et al. (2003), Burke et al. (2004a) while Ernst et al. (2004) provides a more general staff-scheduling review.

Some of the different approaches used to solve the nurse scheduling problem are detailed below.

3.4.1 Exact approaches

Exact methods have been employed in the past to solve nurse scheduling problems; however, it is generally the case that these approaches take an excessive amount of time to find the optimal solution and in many cases are unable to cope with the large number of variables required to solve formulations as complex as most nurse scheduling problems require. Warner (1976) uses a mathematical programming approach to solve a model of a nurse scheduling problem simplified by the use of fixed weekends and limited rotation between different types of shift. Penalties are assigned according to nurse preferences and the final model provided high-quality solutions in a fraction of the time required to create them by hand. Bard and Purnomo (2007) used CPLEX software to solve a linear programming model for a cyclic nurse scheduling problem and found that generating optimal solutions for the problem in this way was unrealistic. Only by hybridising the search with heuristic methods were they able to find good, feasible solutions quickly. Bard and Purnomo (2005b) also had to combine exact and heuristic methods in order to successfully solve a nurse scheduling problem using column generation.

Millar and Kiragu (1998) were able to successfully solve both cyclic and non-cyclic nurse scheduling problems using a branch-and-bound approach. However, the problem size is small (up to 8 nurses) and given that, due to time-constraints, they stop the search when the first feasible integer solution is found, the approach is strictly regarded as a heuristic. Bard and Purnomo (2005a) find that an integer programming approach is suitable for a short-term nurse-rescheduling problem, where existing rosters must be modified for the next 24 hours to cope with fluctuations in nurse supply and demand, again showing that integer programming approaches can be successful for reduced problems. Jaumard et al. (1998) produce an exact solution approach to a nurse scheduling problem using column generation and branch-and-bound, although they also suggest halting the search prematurely once several feasible solutions have been found. The time taken to find good solutions, however, can be several hours. Ikegami and Niwa (2003) discuss replacing their branch-and-bound

approach with a heuristic in order to speed up the solution process. Wright et al. (2006) also have to use a heuristic approach to solve their nurse scheduling problem, although an exact method is sufficient to solve a partial variant of the problem.

3.4.2 Heuristic approaches

Heuristics are algorithms employed to find good solutions to problems where an exhaustive search of the solution space would not be practical. Optimality cannot be guaranteed, but by careful construction or modification of a solution, they can often find satisfactory solutions in a reasonable amount of time. Randhawa and Sitompul (1993) use a heuristic approach to create a computerised nurse scheduling system for use in small- to medium-sized hospitals. Bresina (1996) used a heuristic-biased stochastic sampling technique to solve a scheduling problem, using a "roulette-wheel" selection; weights were applied to each score in proportion to its rank so that choosing one of the better elements was more likely.

As mentioned in Section 3.4.1, heuristic approaches also arise by using an exact method, but stopping the search early before the optimal solution is found, as in Millar and Kiragu (1998).

Heuristics which are more problem-specific generally give better results in the time allowed, however, and there is much evidence in the literature of heuristic approaches being applied successfully to nurse scheduling problems. These include Cheng et al. (1997), Abdennadher and Schlenker (1999), Meyer auf'm Hofe (2001) and Wright et al. (2006).

These types of relatively simple algorithms are the building blocks of metaheuristics, which use heuristics as part of a larger search structure to enhance the quality of solutions.

3.4.3 Metaheuristic approaches

Much of the recent nurse scheduling research in the literature draws upon metaheuristic approaches. We give details of some of these, paying particular

attention to Genetic Algorithms and Tabu Search since they have often been applied to nurse scheduling. Furthermore, they have been used on the problem presented in this thesis with some success as will be discussed in Section 3.5. Metaheuristics are really just a larger framework within which problem-specific heuristics can operate to form a more general solution methodology.

We now discuss some of the metaheuristic approaches which have been applied to nurse scheduling in the past. Note that details of the tabu search and genetic algorithm metaheuristics can be found in Appendices E and F.

3.4.3.1 Ant Colony Optimisation (ACO)

The ACO metaheuristic, described in detail in Section 3.2.2, has been little applied to the area of nurse scheduling. However, as it is one of the methods applied in this thesis, we give special attention to the work of Gutjahr and Rauner (2007), who do use an ACO approach for this purpose.

Gutjahr and Rauner (2007) present an ACO approach to solve a nurse scheduling problem arising at a hospital in Vienna. This problem, and especially their formulation of the problem, differs significantly from the one presented here. The most notable difference between the two problems is that Gutjahr and Rauner are concerned only with scheduling supply nurses, that is, they need only schedule the extra nurses required at peak times, since the basic schedule has already been completed. Further points to note are that they allow a large amount of user-definability and that their problem is dynamic, rather than having a fixed planning period. However, as there is little in the literature on ACO as applied to nurse scheduling, their work is relevant despite these differences.

The trail they suggest is between each nurse and each 'demand' for cover, where a demand is any connected time window, or shift, for which only one nurse is required. Note that this is a similar trail to that often used in ACO approaches to the QAP (e.g. Maniezzo and Colorni 1999) and also to the nurse-shift trail presented in Chapter 5. Their ACO algorithm has an element of positive reinforcement, by increasing only the trail corresponding to the best solution. After their initial testing the parameters found

to be most successful were as follows: $\tau_0 = 1$, $\beta = 3$ in the first third of the run, 2 in the second third and 1 in the final third, $\alpha = 1$. The evaporation rate and the size of the ant population used in Gutjahr and Rauner's research have not been revealed.

Despite only supplementary nurses being scheduled, finding feasible solutions was still an issue; in their illustrated example, more than 10 per cent of the generated demands were unmet. However, they found that their solutions compared favourably with a straightforward greedy algorithm.

3.4.3.2 Tabu Search

Tabu search is a very flexible metaheuristic and may be applied to even highly constrained problems. As such, it has been implemented successfully on many problems in a variety of areas. A description of the algorithm is given in Appendix E. For more details relating to the tabu search method and its applications, refer to Glover and Laguna (1995) and Gendreau and Potvin (2005).

While tabu search has been applied to many problems, the area in which we are most interested is that of nurse scheduling. Several papers have used a tabu search approach to solve a nurse scheduling problem. Many of these have been very successful, but have often required the additional use of problem-specific information in order to do so.

Bellanti et al. (2004) investigate an Italian hospital's nurse scheduling problem. Their method generates an initial solution using a greedy algorithm consisting of a combination of three greedy heuristics before applying local search. Their tabu search algorithm is applied to a partial solution including information only about the night shifts. Four neighbourhoods are considered, all related to satisfaction of the hard constraints regarding the assignment of night shifts. The four neighbourhoods involve: moving a set of night shifts from one nurse to another; moving the first, or last, night shift in a set from one nurse to another, and assigning a new night shift to a nurse as either the first or last in a consecutive set. On the real data provided the tabu search produced the best solutions compared with an iterated local search approach, but in

the longest time; it also performed well on several instances of randomly generated data.

Dias et al. (2003) offer a tabu search solution to a Brazilian nurse scheduling problem and compare it with a genetic algorithm approach. The tabu search is initiated with a random solution and an evaluation function with carefully selected penalty weights and is implemented with the use of a hybrid neighbourhood; one neighbourhood is used until a certain number of iterations without improvement, after which a second neighbourhood is employed. The genetic algorithm was found to produce slightly better results, but the tabu search was less computationally expensive. The tabu search algorithm was a vast improvement on the manually created schedules.

Aside from including problem-specific information to aid the tabu search, there has been evidence in the literature of improving a tabu search approach by hybridising it with other methods. Burke et al. (1999) and Valouxis and Housos (2000), for example, both use a hybrid tabu search approach to solve the nurse scheduling problem with some success.

Burke et al. (1999) introduce two hybrid algorithms developed into the commercial nurse scheduling software Plane. The first of these incorporates a 'diversifying' move. The tabu search is applied until no improvement has been found for a given number of iterations. This schedule is then altered by ensuring there are no incomplete weekends. If there are no such changes to be made, a second diversifying move is introduced; a number of full days which will affect those with the worst individual timetable are swapped, greedily. The second hybrid tabu algorithm introduced by Burke et al. attaches a 'greedy shuffling' algorithm to the end of the tabu search. Since it was realised that manual manipulation of the final schedule could often result in improvements, this final greedy step after the search ensured user satisfaction, by performing any 'obvious' changes to the schedule using the second diversifying move described above. They found that both hybrid techniques produced excellent results and the 'greedy shuffling' method, in particular, produced a level of user-satisfaction even greater than the cost function would suggest. De Causmaecker and Vanden Burghe (2003) noted this algorithm, which allows a large amount of user-definability, often results in overconstrained problems for which the software Plane would be

unable to provide feasible solutions. They therefore introduce ways of relaxing the covering constraints in a similar manner as would be done by a manual scheduler in order to provide high-quality solutions which also satisfy the staff involved.

Valouxis and Housos's approach (2000) hybridises a tabu search algorithm with a standard local search. The tabu search continues until no improvements can be made without violating the tabu list. To further the search a two-step procedure is applied: first, a single move violating the tabu list restrictions is accepted; secondly, a diversification measure utilising a much longer tabu list is employed to move the search to a new region of the search space. This method produced results 'significantly better' than those found using a standard local search technique.

Other nurse scheduling problems which have been solved using tabu search include Ferland et al. (1996) and Ikegami and Niwa (2003). There are several other papers which successfully apply a tabu search method to the variant of the problem presented in this thesis. These will be discussed in detail in Section 3.5.

3.4.3.3 Genetic Algorithms

A description of the genetic algorithm (GA) metaheuristic is given in Appendix F. The applications of this approach to the particular variant of the nurse scheduling problem investigated in this thesis will be discussed in Section 3.5. We now give details of some of the GA approaches applied to other nurse scheduling problems.

Moz and Pato (2007) apply a GA approach to a nurse rerostering problem, rebuilding a previously adequate schedule which has become infeasible due to one or more nurses becoming unable to work shifts which were previously suitable. One of the main characteristics of this problem is that the new schedule must be as similar as possible to the old one, so as to minimise disruption to other staff members. They use an indirect GA approach, due to the fact that the schedules resulting when the crossover operators are applied are consistently infeasible. The algorithms tested performed well compared with a purely constructive heuristic approach, although the time taken to produce the solutions was substantially longer.

Beddoe and Petrovic (2005) use a novel approach to solve a nurse scheduling problem, developed in Beddoe and Petrovic (2003). A database, or case base, of previously encountered constraint violations and the repairs performed to correct them is used to suggest repairs for constraint violations occurring in a current solution. In order to determine which case in the database is the most similar to the constraint violation considered for repair, they suggest a number of particular violation characteristics or 'features' which they may use to compare two violations. They use a GA to optimise the weighting associated with each of these features, so that more relevant aspects of each case will be given more consideration when determining which case from the database is the most similar to the current violation. Note that Scott and Simpson (1998) also use a case-based approach to solve a nurse scheduling problem, but their case base stores sets of efficient shift patterns used to build an initial solution which are then repaired with a series of shift swaps.

Tanomaru (1995) uses a GA to solve a staff scheduling problem similar to the nurse scheduling problem; the staff can be split into distinct groups, similar to the nurses' grades, with each fixed time period having a known requirement for the total number of staff as well as the numbers required for each group. However, the shift times are variable and staff may start a shift (of several time periods) at any hour. Note also that there is no hierarchy in this problem and so the issue of substitution between the different staff groups does not occur. For each nurse and shift, a preference cost of 1, 0, or -1 is assigned. Tanomaru does not use a shift pattern approach, but, for a problem with maximum allowed number of shifts S and E employees, encodes each schedule as an ordered string comprising E consecutive substrings of length $2S$. Each pair of places in the string represents the start and end times of the shift being worked. An inactive shift is represented by the pair (0,0). The evaluation function is the sum of the individual penalty costs assigned to each type of constraint violation. Since overtime is allowed, the emphasis is on minimising cost. A two-point crossover allows movement only of whole employee schedules, but heuristic operators are then applied to make further improvements. These are divided into three types: those which act only on a single shift, those which act only on a single employee and those which act on the whole solution. Tanomaru finds that the GA approach is effective in finding high-quality solutions.

Dias et al. (2003) found that a GA approach was able to produce slightly better solutions than a tabu search approach for a nurse scheduling problem at a large hospital in Brazil, although the time required to do so was longer. It is important to note that one of the main difficulties when applying a GA approach to a problem such as nurse scheduling is in creating and maintaining feasible solutions, due to the fact that the children of feasible solutions may not themselves be feasible. Some of the methods used for dealing with the general problem of maintaining feasibility with a GA approach can be found in Mäkinen et al. (1999), for example. As mentioned previously in this section, GA approaches have also been successfully applied to the nurse scheduling problem presented here (Aickelin 1999, Aickelin and Dowsland 2000, 2004) and these will be discussed in further detail in Section 3.5.2.

Memetic algorithms arise when a GA approach is hybridised with some form of local search and memetic approaches have also been applied to nurse scheduling problems with a variety of success (Burke et al. 2001, Burke et al. 2004b, Özcan 2005).

3.4.4 Other methods applied to nurse scheduling

As well as the other methods already discussed in this section, there have been several other approaches applied to nurse scheduling. Variable neighbourhood search (VNS) and simulated annealing, for example, have been discussed previously as methods used in conjunction with other approaches to form a more powerful hybrid (Liu et al. 2001, Casey and Thompson 2003, Beltrán et al. 2004, Gupta and Smith 2006). VNS has also been used successfully in its own right (Burke et al. 2003a) and although simulated annealing has been much less well instigated as a solution method for the nurse scheduling problem, it does feature in some research. Isken and Hancock (1991), for example, use a simulated annealing approach and Brusco and Jacobs (1995) use simulated annealing in a solution approach to a personnel scheduling problem which may be applied to nurse scheduling. Osogami and Imai (2000) discuss appropriate neighbourhoods for an 'elementary' nurse scheduling problem and show that a 4-opt neighbourhood gives rise to a connected solution space.

Other approaches applied to the field of nurse scheduling include hyperheuristics (Cowling et al. 2002, Burke et al. 2003b), which have been applied to the nurse

scheduling problem presented here and will be discussed in further detail in Section 3.5.3, scatter search algorithms (Maenhout and Vanhoucke 2006) and multicriteria approaches (Berrada et al. 1996, Burke et al. 2002, Burke et al. 2004b).

3.5 Previous methods tackling this instance of the problem

Previously, there have been many solution approaches to the particular nurse scheduling problem presented in this thesis. This section will describe each of the methods in some detail and compare their relative success.

3.5.1 Initial tabu search investigation

Dowsland (1998) and Dowsland and Thompson (2000) were the first to present a solution to the variant of the nurse scheduling problem investigated in this thesis, and used a tabu search approach. Dowsland and Thompson (2000) introduced the pre- and post-processing phases discussed in Chapter 2 and Appendix C. A knapsack model is used initially to analyse the numbers of nurses required and to ensure excess cover is spread evenly over the weekdays and a network flow model is used on a final solution to allocate the nurses working days to early or late day shifts. It should be noted that this knapsack model forms the basis for the feasibility check introduced in Chapter 4. These pre- and post-processing phases simplify the problem to the one investigated in this thesis, formulated in Chapter 2, and is the version solved by all subsequent approaches in the literature.

The tabu search method put forward by Dowsland (1998) is initiated with a randomly generated solution and the standard neighbourhood involves changing the shift pattern of one nurse. However, obtaining a feasible solution for this problem is non-trivial and even when the neighbourhood was initially restricted to moves which decrease the cover cost and do not increase the preference cost, it was found that this rarely led to a feasible local optimum. A main reason for this was the occurrence of so-called 'unbalanced' solutions. Due to the discrepancy between the number of day and night shifts worked by many nurses, it is often essential that the right 'types' of nurses are working days and nights, in order for there to be enough total day and night cover available. For example, by assigning a (3,3) nurse to days and a (4,3) nurse to nights,

the extra day shift which could potentially have been worked by the (4,3) nurse is lost and a total of 6 rather than 7 shifts are now available. Note that once a local optimum is reached, if the partition of nurses into days and nights is unsuitable, no move in the standard neighbourhood can be accepted as any move will necessarily result in an increase in the cover cost. In order to overcome this problem a new strategy had to be introduced, which involved restricting the candidate list of standard neighbourhood moves to those which move a nurse working days to nights or vice versa, depending on where the shortfall lies. The move which has the least negative impact on the cover cost is assigned.

However, even 'balanced', infeasible local optima were found to occur frequently which were surrounded by equal-cost or slightly higher-cost moves. Again, the standard neighbourhood was not sufficient to escape from these situations and so two new neighbourhoods, involving chains of moves, were employed. The 'shift chain' neighbourhood, involves changing the pattern of one nurse by a single shift to increase the cover on an undercovered shift, and then creating a chain of similar moves to replace the undercover created by this initial step. The 'nurse chain' neighbourhood is similar, but, after the initial move, all subsequent nurses are moved to the exact pattern vacated by the previous nurse in the chain. Once a local optimum is reached with regards to all these neighbourhoods, the standard neighbourhood is used again, but is restricted to moves which transfer cover from an overcovered shift to an undercovered shift. It is found that these five steps, applied in succession, are usually able to provide a feasible solution; when this is not the case a final random 'kick' using the standard neighbourhood is applied and the process repeated until a feasible solution is found.

However, the feasible solution found by this first set of neighbourhoods is unlikely to be optimal with respect to the nurses' preferences and so a second phase introduces moves designed to aggressively reduce these penalty costs without increasing the feasibility. However, when no improving moves are found, the search applies a single move which takes the preference cost as the evaluation function and allows all moves in the standard neighbourhood, thus removing the search to the infeasible space, whereupon the first set of moves designed to improve the feasibility are reapplied.

Although this rigorous searching of the solution space often produced high-quality solutions, it was not robust enough to secure good solutions in certain situations. In such situations the cause was always identified as the day/night partition of the nurses and it was found that the partition or partitions utilised were not suitable to produce optimal or, in some cases, feasible solutions. In order to ensure a sufficiently varied range of day/night partitions, the partitions employed were tracked throughout the solution process and a tabu list employed to prevent the search from revisiting the same partition repeatedly. The list is updated whenever a partition-changing move is accepted. If no such change is made within 50 moves, the candidate list is restricted to those which change pattern type.

A further problem affecting feasibility was found to be the occurrence of solutions where both the total day and night cover was unbalanced, rather than the shortfall occurring for just one type of shift. To deal with this case, a new 'swap' neighbourhood was introduced and the candidate list restricted to pairs of moves which move a nurse from days to nights and vice versa.

It is clear that this variant of the nurse scheduling problem is one for which finding feasible solutions is a non-trivial task and where, in particular, the particular nurses assigned to days and nights is of great importance. However, by improving on the basic 1-opt neighbourhood with larger, more problem-specific enhancements, high-quality solutions were found for all datasets and the software CARE developed in Dowsland and Thompson (2000) for use within the hospital remains one of the most successful approaches for this problem to date.

3.5.2 Genetic algorithm approaches

Aickelin (1999) presents two genetic algorithm approaches: direct and indirect. The direct approach represents a complete solution as a $1 \times r$ vector, where r is the number of nurses and element i represents the shift pattern worked by the ith nurse. By insisting each 'gene' in the string be a complete pattern, rather than allowing individual shifts to be encoded, the solution is guaranteed to maintain feasibility with regards to constraints C2-C4 (see Chapter 2) following the application of any crossover operator. For this reason, other solution representations were discarded. By

defining mutation to change the shift pattern of a nurse only to another feasible pattern, these constraints are also upheld after mutation. Only the covering constraints may be violated with such an encoding and a penalty function approach was used to tackle this.

Extensive testing was used to choose suitable parameters; it was found that while some were particularly sensitive, other parameters were less influential on solution quality. The penalty weight associated with violation of the covering constraints C1 was one of the less sensitive parameters; although extreme values negatively impacted on solution quality, a broad range of values produced results of similar quality. Since initial tests using the optimised parameter values were typically poor, a dynamic penalty weight approach was implemented. However, although this did improve feasibility and solution cost, the results were still poor compared with those of Dowsland (1998).

Aickelin points out that a possible reason for this failing is an inherent difficulty in applying a GA approach to a problem where final solution quality is not a linear combination of the quality of the individual genes. In the case of this nurse scheduling problem, each gene or 'nurse-shift pattern pair' has an associated cost, p_{ij}, and these do combine linearly. However, the genes do interact in order to determine feasibility and the covering costs are a result of the particular mix of genes present in a solution, rather than a sum of the fitness of the individual genes present. Aickelin identifies that this non-linearity is two-fold, since the interaction of the different grades is also a contributing factor.

To tackle this second manifestation of interaction within the genes a new approach to solution recombination is proposed. First, the genes are re-ordered within each string according to grade. Secondly, the population was split into 'sub-populations' according to grade and fitness functions applied to each of these. Fixed-point crossovers on grade boundaries were introduced so that some children would be created from larger building blocks of successful solutions. Migration between the different populations ensures a diverse range of solutions in each, while the fixed-point crossover operators ensure that large sections of good solutions are maintained

intact. By further applying problem-specific repair operators, providing incentives and disincentives to avoid unbalanced solutions and strategies to fix balanced ones, the quality of results was greatly improved, although results were still not as good as in Dowsland (1998) and so an indirect GA approach is employed and the problem is remodelled so as not to require any constraint-specific information within the encoding.

The indirect GA approach presented still uses a string of length r, but is now representative of a permutation of the ordering of the nurses. A separate decoder is then used to build a solution from this ordering, by applying a greedy heuristic, assigning a shift pattern to each nurse in the order provided by the GA solution.

Two decoders are presented: a 'cover highest' decoder scores each shift pattern for the current nurse based on the effect assigning it will have on the day or night shifts with the highest undercover and an 'overall contribution' decoder scores the patterns based partly on the effect to the days and nights currently undercovered and partly on the preference cost p_{ij}. Both of these decoders perform poorly and so a third 'combined' decoder is introduced. This decoder still incorporates the preference cost of the shift pattern, but for each shift covered by it, the score is related to the number of remaining uncovered shifts, rather than just a binary value indicating whether the shift is undercovered or not. This decoder improves results considerably and is enhanced further by fine-tuning the parameters and introducing a new crossover operator. By utilising a simple bound such that no pattern is assigned with $p_{ij} > C^*$ if the cost of the best feasible solution found so far is C^*, the algorithm avoids wasting time on solutions which cannot possibly be optimal.

With these modifications, the indirect GA provides good solutions, better than those produced using the direct GA and almost as good as those produced in Dowsland (1998).

Details of these direct and indirect GA approaches can also be found in Aickelin and Dowsland (2000, 2004).

3.5.3 Other approaches

Cowling et al. (2002) and Burke et al. (2003b) use a hyperheuristic approach, previously applied successfully to two other scheduling problems. The hyperheuristic is initialised with a schedule by randomly assigning each nurse a feasible shift pattern and consists of nine low-level heuristics, each involving changing or swapping one or two shift patterns. Noting that the previous methods (Dowsland 1998, Aickelin and Dowsland 2000) have relied heavily on problem specific information, Burke et al. (2003b) aim to produce an algorithm which will transfer more easily to other similar problems. They wish to develop an algorithm which will produce results which are 'good enough – soon enough – cheap enough' for a range of problems, rather than one which is problem-specific and, although possibly achieving better solution quality, will not transfer so easily

These low-level heuristics were adaptively ranked within the algorithm, using information based on three factors: the individual success of each heuristic (denoted f_1), the combined success of pairs of heuristics (denoted f_2) and the last time each heuristic was called (denoted f_3). Of these, f_1 and f_2 are intensifying procedures designed to increase the chances of successful heuristics being utilised, whilst f_3 is a diversifying move, increasing the chances of using a heuristic if it has not been called recently. The weights associated with each of these were modified dynamically within the algorithm, generally resulting with a high weight for f_3, indicating that a high level of diversification was required.

According to Cowling et al. (2002), the two heuristics called most often were changing the shift pattern of a random nurse to improve feasibility (referred to as [h2]) and changing a nurse's shift pattern type from days to nights or vice versa if the solution is unbalanced in this respect (referred to as [h6]). Burke et al. (2003b) use a tabu list to disallow the use of those heuristics which have recently failed to improve the solution, on a first in, first out basis. The list is emptied each time a change to the solution has been accepted.

The results were compared with the tabu search approach in Dowsland (1998) and the direct and indirect GA approaches in Aickelin (1999) and Aickelin and Dowsland

(2000). The hyperheuristic approaches proved robust, consistently finding feasible solutions, but were outperformed by all other approaches in terms of preference cost.

Li and Aickelin (2003, 2004), Aickelin et al. (2007a), Aickelin and Li (2007) introduce the first Bayesian Optimization algorithms to be applied to the problem. Their solution approach is to decide, for each nurse, the rule by which the nurse will be scheduled. For a problem with r nurses, then, they seek a string of length r, where each place in the string represents a nurse and the element in position i is the rule by which nurse i will be scheduled. They model the problem as a Bayesian network in which each node N_{ij} represents nurse i being scheduled by rule j. The eventual aim is to emulate the high-quality scheduling capabilities demonstrated by human schedulers, by implementing this model using relatively simple rules, but by being able to switch between these rules intelligently.

In each case a string is obtained by finding a directed path from nurse 1 to nurse m, linking m nodes. Each node is associated with a conditional probability, based on the nodes before it and these probabilities are updated based on the fitness of solutions produced.

Given a current population of rule strings, a subset of the most promising strings is selected from the population and the conditional probabilities of each node are calculated. New strings are then generated based on these conditional probabilities, using the probabilistic, roulette-wheel method, whereby the better solutions are more likely to be selected. These new strings then replace some of the previous population. Subsequent generations are produced in the same way. The process is initialised with a randomly generated population of strings. The four rules suggested are: to assign the nurse a random pattern; to assign the nurse a randomly selected pattern from the k patterns with the lowest p_{ij} values, where k=5; to assign the nurse based solely on the improvement to the feasibility of the current schedule; and to assign the nurse based on both the p_{ij} values and the feasibility, but with the emphasis on solution quality. They therefore have rules which focus solely on solution quality, solely on feasibility, a combination of the two and a final random rule.

Li and Aickelin (2004) build on this initial research by introducing an adapted classifier system. Feedback is provided by assigning a positive reward evenly across all nodes in solutions better than the previous one and a negative amount in the same way for all nodes used in solutions poorer than the previous one. The best solution so far is always kept. Thus the adapted classifier system improves a single solution rather than the population-based approach used by the initial Bayesian algorithm and although it performs well, and produces feasible solutions in every case, the results produced are slightly poorer than those created using the original Bayesian approach.

The Bayesian optimisation approach is further enhanced by modifications in Aickelin et al. (2007a). The four rules suggested in Li and Aickelin (2003) are extended to six, by the addition of two new cover and contribution rules. Previously, the cover rule scored shift patterns by the single most understaffed shift they covered. The new cover rule assigned a score based on the total level of understaffing of all shifts covered by the pattern. The new contribution is also similar to that originally presented. The initial contribution rule incorporated a score partly based on feasibility; this part of the score worked by assigning a '1' to understaffed shifts covered by the pattern and including a weighted sum of these. The new version of the contribution rule worked in the same way, but instead of scoring understaffed shifts with '1', the new rule scores them with the actual amount by which they are undercovered.

The algorithm is improved further by the addition of an ant-miner algorithm. Trails τ_{ij} are between each nurse i and each rule j. Thus each node in the Bayesian network is associated with a particular trail element, reinforcing the conditional probabilities. The addition of the ant-miner algorithm proved to be worthwhile, with large improvements in cost noticeable on several datasets, although the addition of the two new rules did not appear to have such an effect.

The results obtained from each of these approaches were compared with the optimal IP results, those produced by the GA results from Aickelin and Dowsland (2000), and two further variants of the Bayesian optimisation algorithm: one using solely the random rule and one using all four rules but with each having a constant 25% chance

of being chosen. The Bayesian optimisation algorithm performed extremely well when compared with the GA. The Bayesian algorithm found optimal or near optimal solutions for 38 of the 52 datasets compared with the 42 found by the GA. The Bayesian approach, however, found feasible solutions in all cases, which the GA had failed to do. The results from the two variants of the Bayesian approach showed that the conditional probabilities were essential to the success of the algorithm, with the constant 25% rule variant performing very poorly in comparison, and the random variant failing to find even a single feasible solution across 20 runs of each dataset.

Aickelin et al. (2007b) further tackle the problem using a squeaky wheel optimisation method which they initialise with a random solution. In a 'selection' step, each assignment is assessed and less successful components are removed, probabilistically, according to their fitness. A mutation step then discards a number of the remaining nurse assignments with a small probability. The squeaky wheel part of the algorithm rebuilds the schedule by prioritising the nurses to be scheduled according to their initial fitness scores, with the 'less fit' nurses given a higher priority. A greedy algorithm then rebuilds the schedule taking the nurses in order of priority. The fitness is then reassessed and the process repeated. The method, improving on an initial squeaky wheel approach with the introduction of the selection and mutation steps, performs well, finding optimal solutions for all but 5 datasets.

Aickelin and White (2004) investigate a method by which two algorithms applied to the same problem may be compared. As was the case for Burke et al. (2003b), their method deals with the relative success of different approaches and, as such, may be applied easily to other problems.

A focal point of the research, particularly apparent in this nurse scheduling problem, is how to compare two methods when infeasible solutions are part of the solution space, and how to rank two methods when one gives mostly infeasible but otherwise low-cost solutions and the other consistently produces poor-quality feasible solutions. Aickelin and White consider a feasible solution to be more successful than an infeasible one, regardless of the actual costs involved. Note that this is the approach chosen for evaluating the fitness of solutions in this thesis.

3.6 Conclusions

This chapter has provided a background to the work in this thesis. As discussed in the introduction in Chapter 1, the aim of this thesis is to investigate the conflict between different constraints and the balance which may be achieved using a constructive metaheuristic approach. In Section 3.2 we introduced the two metaheuristics which will be employed as the basis of our investigation and gave an overview of how they have previously been applied to other problems. Neither of these approaches has yet been used to solve the problems presented in Chapter 2. However, from the review in this section it was clear that although these two methods, GRASP and ACO, are construction-based metaheuristics, there is very little work in the literature about using the constructions to build feasible solutions or to balance different constraints; most applications have been to problems where feasibility is not an issue and for those problems where there is an existing conflict between constraints, enhancements to the basic metaheuristic are often used to improve the already-constructed solutions. Those approaches which do concentrate on the construction aspect tend to do so by enhancing the basic algorithm, rather than by comparing different ways of balancing the constraints within the basic constructive framework. There is therefore scope for a study which investigates whether a simple constructive technique is able to create the necessary balance by performing a rigorous comparison of different constructive approaches.

As has been discussed, there is little relevant work in the literature with regards to medical student scheduling and so this chapter has necessarily been focused on the work surrounding the nurse scheduling problem. This chapter has placed the variant of the nurse scheduling problem studied in this thesis within the context of a wider body of work, demonstrating that the area of nurse scheduling as a whole is one which warrants a great deal of investigation and, further, that the particular variant studied here is one which has been proven to be of sufficient difficulty to justify repeated investigation from a number of perspectives. The aspect of feasibility is one which is repeatedly mentioned in the literature with regards to this problem, with many researchers claiming success when achieving sub-optimal, but feasible solutions. The only method applied to this problem to date which has produced optimal solutions for all available datasets has been a tabu search method, incorporating a great number of

complex, problem-specific neighbourhoods. Therefore, there is room in the literature for a solution approach which bases its work on balancing these different constraints within a constructive framework.

Because the medical student scheduling problem is one which does not appear frequently in the literature, it makes an interesting subject for study. Although the nurse scheduling problem has been widely studied, however, the constructive approaches which have been applied so far have not been highly successful. This makes applying a construction-based technique more interesting, especially since the difficulty of the problem has been well established and many details regarding the problem structure are known. Given that the subject of conflicting constraint satisfaction within a constructive framework is not one which has been widely researched, applying such techniques to these problems will further add to the body of work in this area.

The aims of the thesis were laid out in Chapter 1, and this chapter, in combination with chapter 2 have served to introduce the problems and methods which will be applied, as well as placing the work investigated in this thesis within a wider context. The next chapter will provide details of the investigation into nurse scheduling using a GRASP approach.

Chapter 4

Nurse scheduling with GRASP

4.1 Introduction

The description of the nurse scheduling problem is given in Chapter 2 along with a discussion of the particular problem characteristics and difficulties. In the literature review in Chapter 3, details were provided of the methods previously applied to this problem; so far all methods have either used highly problem-specific information and complex neighbourhoods to obtain low cost solutions or have used a much more generalised approach, often finding feasible solutions, but at the expense of solution quality. In Chapter 1, one of the aims of this thesis was stated as investigating this trade-off between conflicting constraints within a constructive metaheuristic framework. The GRASP approaches applied will be as general as possible, but it will be shown in the later section of this chapter that in order to rival the most successful solution approaches for this problem, it is necessary for some problem-specific knowledge to be exploited.

Due to the complexity of the problem, generating feasible schedules is demanding and, as such, our solution space must include both feasible and infeasible schedules. Given this, we compare different ways of balancing the search between finding feasible schedules and finding ones which have an optimal preference cost. One option is to base the construction entirely on just the feasibility of the problem and to see whether the local search phase is able to optimise the schedule. Another is to use weights to balance these two factors in the construction, which leaves a better starting solution for the local search, but may reduce the probability of obtaining a feasible solution. A third method is to use a look-ahead procedure to ensure, at each stage of the construction, that the schedule does not stray from the space of partial solutions from which feasible complete solutions can easily be obtained by the local search. In Chapters 2 and 3 we mentioned the problems associated with unbalanced solutions; the look-ahead procedure would be used to ensure solutions remain balanced throughout the construction process. We try seven construction methods and compare each of these with and without a look-ahead procedure. Each construction method generates a complete, if not necessarily feasible, schedule. The look-ahead procedure would be based on exploiting the problem structure with regards to the feasibility constraints, using a knapsack model to solve a relaxed version of these constraints exactly, as suggested in Chapter 2. Note that only a relaxed version of the constraints are solved using this exact method and thus schedules incorporating this added feasibility check will still not be guaranteed to be feasible, although they will be balanced.

For the improvement phase, three simple neighbourhoods will be explored. The first is a 1-opt neighbourhood, involving changing the shift pattern of a single nurse. The second, a swap neighbourhood, swaps the shift patterns of two nurses. The third, a 2-opt neighbourhood, involves a more rigorous search for improvement: placing a nurse onto a pattern of lower cost and searching through the remaining nurses to find one who may be moved to cover the resulting uncovered shifts such that, overall, the cost is improved. This can be viewed as a chain neighbourhood utilising chain length 2.

The rest of the chapter is organised as follows. In the next section we introduce the solution approaches, including more details of the construction heuristics, look-ahead procedure and local search neighbourhoods. This will be followed by an explanation

of the experiments undertaken and their results. Initial experiments to obtain parameters will be followed by in depth testing of the different approaches and extensions to these basic methods in order to further improve the schedules produced.

The next section details the initial nurse scheduling solution approach using GRASP. Note that the notation in this chapter is as defined in Chapter 2 and is summarised in Appendix D.

4.2 Solution approach

This section is concerned with the solution approaches with which we shall tackle the nurse scheduling problem. Since we aim to solve this problem using GRASP, we shall be discussing the construction algorithms with which it may be sensible to proceed along with the difficulty of ensuring feasible solutions at the end of the construction phase. The potential local search neighbourhoods available for use in the improvement phase will be detailed at the end of this section.

We begin with an examination of the construction phase.

4.2.1 Construction

As outlined in Chapter 3, the GRASP method consists of both a construction phase and an improvement phase, the construction phase gradually building a solution from either a partial or, more usually, an empty starting solution. The manner in which the full schedule will then be assembled is not pre-determined, but may be approached in several ways; the particular heuristics used to score and select components to add to the schedule must be carefully decided upon. This section describes the particular heuristics and construction approaches we have decided to investigate, with explanations of why these choices have been made.

We have already discussed some of the difficulties of the nurse scheduling problem, in particular, the difficulty of balancing feasibility with low solution cost. As has been mentioned in Chapter 2, even finding feasible solutions is a non-trivial task, since there is further balancing required: balancing the hospital requirements with the

nurses' grades and individual contracts. Given that we are unable to guarantee feasible solutions, we must include both feasible and infeasible schedules in our solution space. Dowsland (1998) used a method of strategic oscillation between these regions, first rigorously reducing cover costs and then intensively improving the preference costs. The difficulty with improving preference costs from a feasible solution is that neighbouring solutions with a lower preference cost are all likely to be infeasible. By making a move to take the solution back into the infeasible region of the search space, there is the chance to descend to a new local optimum and making this move such that the preference cost is lowered is likely to increase the chance of the new local optimum being of a higher quality.

The GRASP approach presented here aims to find low-cost, feasible solutions without this type of move. We allow both feasible and infeasible schedules in the initial solution space through necessity, but wish to build an algorithm which will construct solutions in an intelligent manner, thereby negating the need for such 'uphill' moves in the local search phase. It is the intention to determine whether there can be an appropriate constructive technique whereby the constructed solution will lie in a good enough region of the search space for uphill moves in the local search to become unnecessary.

We have already discussed some of the ideas behind the constructive heuristics in the way they balance the two conflicting costs. The first idea was to concentrate solely on the feasibility during the construction, to see whether the local search would be able to optimise solutions. The second was to try balancing the two costs using weights within the construction to give the local search a potentially better starting solution. These two ideas are based on Aickelin's (1999) research in which his *Cover* and *Combined* decoders, used to construct a schedule from an ordering of the nurses created using a GA approach, took this form. Note that using Aickelin's approach, two further points of interest arise when adapting this for a GRASP application and we shall introduce these in turn. The first point is that, while Aickelin was using a fixed ordering created by the GA, the GRASP approach presented here has no such ordering and it must be decided whether it is sensible to use a fixed ordering or whether all nurses should be considered for assignment at each stage of the construction process. We shall discuss later why *Cover*, because of the way in which

it deals with grades, should be used with a fixed ordering of the nurses. However, *Combined* gives rise to no such difficulties and so we try both approaches here, denoting the approach where *Combined* allows all nurses to be considered by *Holistic*. The second point of interest is that, since *Combined* and *Holistic* have scores based on both types of cost, we must decide on how to most effectively combine these into a single score. Clearly it is possible to weight the two options, but the question remains whether to use an additive approach or a multiplicative approach. That is, would a score of the form $\alpha.\text{Cost}_1 + \beta.\text{Cost}_2$ be more appropriate than a score of the form $\text{Cost}_1^{\alpha} \times \text{Cost}_2^{\beta}$, where Cost_1 and Cost_2 represent the scores for the two costs and α and β their associated weights. We choose to investigate both of these scoring approaches and the implications of each type will be discussed later. Thus we have five heuristics based on the work of Aickelin (1999): one based on his *Cover* decoder and four based on his *Combined* decoder, comprising two scoring methods for *Combined* and two for *Holistic*.

The final construction idea introduced for the GRASP approach is based on an idea found in Balas and Saltzman (1991) and Robertson (2001). The difference in score between the highest and second-highest scoring shift pattern for each nurse is considered at each stage of the construction. The idea is that if there is a large difference in score between these two options for a nurse, it is important to schedule them at this stage with their 'best' pattern, since, later in the construction, they may not be able to work this pattern due to feasibility restrictions and this could be very costly to the schedule as a whole. This method, denoted *LastChance*, therefore chooses a nurse to schedule at each stage and assigns that nurse their highest-scoring shift pattern, with these scores calculated as for *Holistic*. Clearly, this method cannot be used with a fixed ordering, but may be used with the two options for combining the scores as for *Holistic*. We therefore have two variants of the *LastChance* heuristic and seven potential construction heuristics in total all investigating different ways of balancing the two costs. Although feasibility cannot be guaranteed, we will show in Section 4.2.2 how we are able to use a knapsack model, modified from the pre-processing phase used in Dowsland and Thompson (2000), in order to construct solutions which are more easily made feasible during the local search phase of GRASP. By incorporating this model into the construction, the GRASP will be shown

to be more robust with respect to feasibility, allowing the construction to give more weight to preference cost and allowing overall better quality solutions to be produced.

This section gives details of the seven heuristics investigated and the reasons why these may be successful. The two final costs, cover and preference cost, are not amalgamated into a weighted sum, but are calculated separately, given that any infeasible solutions may be discarded regardless of preference cost. The formulations for these costs are presented at the end of this section. We now describe the general constructive method, after which details of the heuristics which may be used in conjunction with this algorithm are given.

As mentioned in Chapter 2, there are two avenues for creating the nursing schedules: either a nurse is selected, along with a shift pattern for that nurse, or a shift pattern is selected, and a nurse assigned to it. The latter, however presents a fairly illogical way to proceed, since each shift pattern may be used once, more than once or not at all. By working through the nurses systematically, however, the task becomes much easier to handle and so it in this manner the algorithm will function.

Given this, at each stage of the construction, a nurse and a shift pattern for that nurse must be selected. The score function of the incorporated heuristic could be used to select both of these, or the nurses may be pre-ordered and each nurse selected in turn. A pre-ordering of the nurses would be expected to decrease the computation time, but allowing both nurse and shift-pattern to be selected simultaneously allows the algorithm more flexibility which, in turn, is likely to offer the more successful approach.

Let R be the set of nurses and R^+ be the set of nurses already allocated. Then, for any given score function, $f(i,j)$, the construction phase proceeds as in Figure 4.1.

Procedure to allocate nurse-shift pattern pairs

Step 1: Set $R^+ = \varnothing$.

Step 2: Calculate the score $f(i,j)$ associated with allocating nurse i to shift j, for all feasible pairs (i,j)[§].

Step 3: Let L be the candidate list of the n highest scoring options.

Step 4: Select $(i,j) \in L$ using roulette wheel selection, i.e. each (i,j) is selected with a relative probability proportional to $f(i,j)$.

Step 5: Update the schedule with this allocation and set $R^+ = R^+ \cup \{i\}$

Step 6: If $R^+ \neq R$ go to Step 2.

§ Note that the set of feasible allocations (i,j) may include all nurses not already allocated, or may relate to a single nurse i given by a predefined ordering.

Figure 4.1. Procedure to allocate nurse-shift pattern pairs.

We have two score functions relating to the cover. Both of these are functions of the total *undercover*, d_{gk}, of shift k at grade g defined as

$$d_{gk}(R^+) = \max\left\{0, C_{gk} - \sum_{i \in R^+} \sum_{j \in S_i} a_{jk} h_{ig} x_{ij}\right\} \tag{4.1}$$

The first of the cover scores, $covscore1_{ij}$, suggested by Aickelin and Dowsland (2004) is designed to be used without a preference cost score and in a situation where the nurse order is predefined. It is defined as follows.

Let

$$g' = \begin{cases} \min\left\{g \geq g_i : \sum_{k=1}^{14} d_{gk}(R^+) > 0\right\} & \text{if such a } g \text{ exists} \\ 3 & \text{otherwise} \end{cases}$$

and let $k' = \text{argmax}\{d_{g'k}\}$, then

$$covscore1_{ij} = \sum_{k \in K} a_{jk} d_{g'k}(R^+), \tag{4.2}$$

where $K = \begin{cases} \{1,...,7\} & \text{if } k' \leq 7 \text{ or the nurse must be on days} \\ \{8,...,14\} & \text{if } k' \geq 8 \text{ or the nurse must be on nights.} \end{cases}$

For a given nurse i, this score is a measure of the improvement in cover at grade g_i (or at the first uncovered grade below g_i if all cover at g_i is already satisfied), where g_i is the grade of nurse i, giving a greater score to those shifts with the most undercover. If the greatest shortfall in cover on a single shift is on nights then only the night shifts will contribute to the score of a shift pattern and vice versa.

As the scores for different nurses are measured according to a particular grade, this method of scoring is not suited to situations in which the nurse has not already been determined. It is sensible to use this method with a nurse-ordering where the higher-grade nurses are scheduled first, since priority is given to the higher-grade cover until no more cover at that grade is required. By scheduling the nurses in different order, where some grade 3 nurse are scheduled before grade 1 nurses, for example, the algorithm would not be able to spread the cover as efficiently.

The priority given to nights or days also make it unsuited to combination with a preference cost score, as it is common for there to be a large difference between the preference costs for days and nights for some nurses; such differences would not be considered.

The second cover score, $covscore2_{ij}$, is more general, taking all grades into account, and so may be used when the nurses are not subject to a predefined ordering. It is also suitable for use alongside a preference cost score as the undercover on all day and night shifts contributes to the score. This cover score is simply a weighted sum of the undercover, and is given by

$$covscore2_{ij} = \sum_{g=g_i}^{3} w_g \left(\sum_{k=1}^{14} a_{jk} d_{gk}(R^+) \right), \tag{4.3}$$

where w_g is the weight associated with grade g for $g = 1,...,3$.

Given that we wish to combine this second cover score with the appropriate preference cost score, there are two ways to proceed. We may either add a preference cost contribution to the score or use the preference cost as a multiplier. The following paragraphs explain how these two ideas may be put into practice.

Since the preference costs are in the range [0,100], we may simply add $100 - p_{ij}$ to $covscore2_{ij}$. This means that for a shift pattern with an unacceptable cost of 100, we add zero to the score, and for a pattern with an optimal preference cost of zero, we add 100. Note that we do not just deduct p_{ij} from $covscore2_{ij}$, since this may lead to some negative scores which would be incompatible with a roulette wheel selection method of choosing a shift pattern.

The first preference cost score then, $prefscore1_{ij}$, can be written as

$$prefscore1_{ij} = 100 - p_{ij} \tag{4.4}$$

to be used as part of a weighted sum in a combined score function.

The second method was to use a multiplicative approach. Since we wish the score to decrease for large p_{ij} values, we may use $1/p_{ij}$ as the multiplying factor. Since preference costs of zero exist, however, we shall take the denominator to be $p_{ij} + 1$ to avoid the possibility of dividing by zero.

For simplicity, we shall take the second preference cost score, $prefscore2_{ij}$, to be

$$prefscore2_{ij} = p_{ij} + 1, \tag{4.5}$$

where $prefscore2_{ij}$ is intended to be used as a dividing factor.

The difference between the effects these two approaches will produce is not immediately apparent; both will give larger scores to shift patterns with lower preference costs and smaller scores to less desirable patterns. However, the way in which these scores differentiate between patterns of similar cost can be significant.

For example, two shift patterns with the same (non-zero) value for $covscore2_{ij}$ and with preference costs of 0 and 1 will have very similar scores using the $prefscore1_{ij}$ approach and very different scores using the $prefscore2_{ij}$ approach. The additive version will add 0 or 1 to the score, but the multiplicative version will divide the cover score by either 1 or 2, thus the multiplicative version attributes double the importance to the zero-cost patterns. When the different parts of the score are weighted, this difference can only become more marked. Another example of the difference between these methods is their effect on a pattern's score when $covscore2_{ij} = 0$. The additive version will base the scores on the preference cost alone, but the scores using the multiplicative version will all be zero. In this case, either a second score related only to the preference cost must be used for the roulette wheel selection, or each zero-cost pattern must be selected with equal probability. Note that if there are some options which improve the cover, the multiplicative version will always choose one of these, whereas the additive version has the possibility of choosing options beneficial to the preference cost over those which improve the cover. Clearly, the likelihood of choosing an option which does not improve the cover, over one which does, will depend on the values of the weights associated with each aspect.

It is not clear which of the two approaches, additive or multiplicative, will be the most likely to produce high-quality solutions. In some cases, the sensitivity of the multiplicative algorithm may be what is required to differentiate between good and excellent solutions, but, on the other hand, this very sensitivity to changes in the preference cost may go too far and prevent the construction from selecting patterns required for feasibility.

Without further testing it is impossible to know which method to continue with and so we now describe seven heuristics, one without any reference to the preference cost, three using the additive approach and three using the multiplicative approach, with which we will experiment.

These heuristics are based upon decoders used in Aickelin and Dowsland (2004) to decode a string of nurses obtained through a GA approach into a complete schedule. The decoders work with a predefined ordering of the nurses since it is this ordering which the GA has sought to optimise, however, in this research, we are not seeking to

find an optimal ordering and any pre-ordering will simply take the nurses in order of grade. We also have the option not to use a predefined ordering and we have already mentioned that this will allow the algorithm more flexibility and, therefore, more potential for success.

The details of the seven heuristics are given below.

4.2.1.1 Cover

We define the score function for the *Cover* heuristic to be

$$Cover: f_1(i,j) = covscore1_{ij}. \tag{4.6}$$

This heuristic is Aickelin and Dowsland's (2004) *Cover* decoder and is dedicated solely to finding feasible solutions. As was mentioned previously, $covscore1_{ij}$ is unsuitable for use in conjunction with a preference cost score due to the often large discrepancies between preference costs for day and night shifts.

As this algorithm is designed to be used with a predefined ordering, we schedule the nurses in order of grade, with more qualified nurses scheduled first. Since we assume that nurses of a higher grade can cover for nurses of a lower grade, by scheduling the higher grade nurses first according to the amount of undercover, it seems likely that we would be improving our chances of finding feasible solutions. By restricting the choice of shift patterns to those which have a chance of reducing the maximum undercover, this algorithm seems to offer the best chance of creating feasible solutions.

Since *Cover* is based only on improving the staff coverage, the task of optimising the solutions with respect to the nurses' preferences falls entirely to the local search phase of the GRASP.

4.2.1.2 Combined

This heuristic predefines the order in which the nurses are to be scheduled in the same way as for *Cover*. The score function for the additive version, $Combined_a$, of this heuristic is defined as

$$Combined_a: f_2(i,j) = w_c.covscore2_{ij} + w_p.prefscore1_{ij} \tag{4.7}$$

where w_c and w_p are weights associated with the cover and preference scores, respectively. If $w_c = w_p = 1$, then this score function becomes the same as that for the *Combined* decoder introduced in Aickelin and Dowsland (2004).

The multiplicative, or quotient, version of the *Combined* heuristic, $Combined_q$, has the score function given by

$$Combined_q: f_3(i,j) = \frac{covscore2_{ij}^{w_c}}{prefscore2_{ij}^{w_p}}, \tag{4.8}$$

where w_c and w_p are weights as before.

A possible drawback of the *Combined* heuristic is that the nurse order is predefined. The GA implemented by Aickelin and Dowsland (2004) was used to order the nurses before the heuristic decoded the ordering into a schedule. Given that we use only a single ordering of the nurses, there are large areas of the solution space which will not be fully explored by *Combined* as it stands.

4.2.1.3 Holistic

In order to overcome the problem of inflexibility faced by $Combined_a$ and $Combined_q$, we introduce heuristics which do not assign the nurses in a given order, but instead assign a score to each nurse-pattern pair. At each stage of the construction both the nurse i and shift pattern j are selected, allowing the heuristic a more complete exploration of the solution space.

Thus, $Holistic_a$, the additive version of this heuristic, has the same score function as $combined_a$ and is therefore given by

$$Holistic_a: f_2(i,j) = w_c.covscore2_{ij} + w_p.prefscore1_{ij}. \quad (4.9)$$

Note that, in this case, i is not limited to representing a single nurse selected in advance for assignment, but is evaluated for all nurses for whom no assignment has yet been made.

The multiplicative version of the *Holistic* heuristic, $Holistic_q$, again uses the same score function as $Combined_q$ and thus it is given by

$$Holistic_q: f_3(i,j) = \frac{covscore2_{ij}^{w_c}}{prefscore2_{ij}^{w_p}}, \quad (4.10)$$

again, where the nurse index i represents all possible nurses to be scheduled rather that the single nurse given by the pre-ordering used by $Combined_q$. Again, this 'quotient' will share many of the attributes of its additive counterpart, and will have the subtle differences discussed earlier.

Although this wider search area may well give rise to better solutions, the large number of options to be considered may give rise to increased computational expense as well as a lack of focus.

4.2.1.4 LastChance

As discussed earlier, by introducing the dynamic ordering the search is more flexible and by allowing the search to cover a wider area, it is more likely that the search space covered will include optimal solutions. However, the potential drawbacks mentioned were the increase in computational expense, since the number of score functions to be calculated and assignments to select from will be greatly increased, and the possible lack of focus, since the assignments are not necessarily being made in order of grade.

A solution to these problems is proposed in the final set of heuristics presented. These will incorporate an element of look-ahead in order to help focus the search while maintaining the flexibility from the dynamic ordering. In order to do this, the same scores are evaluated as for the *Holistic* heuristics, but a second calculation determines, for each nurse still to be scheduled, the difference in cost between the best available score and the next-best. This difference in cost constitutes the score function. The nurse is then selected based on these scores and the nurse's best-scoring pattern is assigned to them. The score functions for $LastChance_a$/$LastChance_q$ can be expressed mathematically in the following way.

Let $j^*(i) = \arg\max_{j \in S_i} \{f_{2/3}(i,j)\}$. Then

$$Lastchance_{a/q}: \; f_{4/5}(i) = \max_{j \in S_i} \{f_{2/3}(i,j)\} - \max_{\substack{j \in S_i \\ j \neq j^*(i)}} \{f_{2/3}(i,j)\} \qquad (4.11)$$

Note that the score function itself is very simple, but the additive and multiplicative variations come from the cost functions used to calculate the scores for each pattern, from which the best and next-best for each nurse are then identified.

By assigning the best pattern to a nurse where there is a large difference between the best cost and the next-best cost, we ensure that the cost for assigning this nurse is kept low. Nurses where this difference is less substantial may be assigned later, since if their 'best' patterns are no longer workable the damage to the overall quality of the solution is unlikely to be as great.

Note that often, during the earlier stages of the selection process, all nurses will have a difference in cost of zero between their two preferred shift-patterns, as it is normal for nurses to have several shift patterns with the same preference cost. In these situations, the scores for each nurse will be based on the actual cost of their best shift pattern. Thus the scores for each nurse *i* would be given by

$$f'_{4/5}(i) = \max_{j \in S_i} \{f_{2/3}(i,j)\}. \qquad (4.11')$$

If these are all zero, then a random nurse is selected and one of their best patterns assigned.

This variant has fewer options at each stage as only the nurse has to be chosen. However it does not suffer from the limitations of the *Cover* and *Combined* constructions in that the shift pattern associated with each nurse also changes adaptively, depending on the allocations made in previous stages. The inclusion of some degree of look-ahead into the score should help to overcome the problem inherent in all greedy approaches, of potentially being left with a set of expensive options in the last few stages.

4.2.1.5 Costs

Once the greedy construction is complete, the costs are calculated as follows:

$$\textbf{Cover cost} = \sum_{k=1}^{14}\sum_{g=1}^{3} \max\left\{0,\ C_{gk} - \sum_{i=1}^{r}\sum_{j\in S_i} h_{ig}a_{jk}x_{ij}\right\} \tag{4.12}$$

$$\textbf{Preference cost} = \sum_{i=1}^{r}\sum_{j\in S_i} p_{ij}x_{ij} \tag{4.13}$$

Note that having more staff than required on any shift is not penalised and a score of zero will be returned as long as all staffing requirements are satisfied at each grade. Although the original problem requires that any excess cover be spread evenly over the weekdays, this is automatically handled by the pre-processing phase of Dowsland and Thompson (2000), detailed in Appendix C, whereby available excess is included in the weekday requirement and a 'dummy' nurse is made available to ensure a feasible solution will be possible, but that there is no excess cover. A dataset for which any feasible solution will have no excess cover is deemed to be 'tight', whereas a dataset for which a solution may be feasible, but also have more than the required number of nurses per shift is deemed to have 'slack'. The large majority of the 52 datasets considered in this thesis fall into the 'tight' category, due to the pre-processing phase.

By meeting the covering constraints, the solution will be feasible; the other hard constraints, those of meeting the nurses' contractual obligations and disallowing consecutive shifts (a night shift followed by a day shift, for example), are automatically built into the system of assigning each nurse with a suitable shift pattern. Therefore, to create feasible solutions, achieving a cover cost of zero is sufficient.

A preference cost of zero is not generally possible, although the optimal solutions are known and a theoretical minimum can always be obtained as the sum of the minimum cost shift-patterns for each nurse. However, the pre-processing done by Dowsland and Thompson (2000) ensures that a feasible solution, with zero cover cost, exists for all datasets through the inclusion of 'bank' nurses where there is not enough staff cover available to meet the requirement.

We have discussed, however, how these construction heuristics may struggle to find feasible solutions and how the use of a knapsack model, similar to that used in the pre-processing stage by Dowsland and Thompson (2000), may be used in order to direct the search towards feasible areas of the solution space.

In the next section we introduce this knapsack model and describe how it may be used in conjunction with each of these heuristics. This hybrid of GRASP with an exact method will be shown with experimentation to be very successful in overcoming this problem.

4.2.2 Hybridising the construction with a knapsack model

As we mentioned earlier, we introduce the knapsack model as a means of creating partial solutions from which feasible schedules can easily be obtained using local search. As will be detailed in the next section, the two neighbourhoods utilised in the local search when feasibility has not yet been achieved involve either changing the pattern of a single nurse or directly swapping the patterns of two nurses. As mentioned earlier, there are situations where the solution is 'unbalanced', as described in Dowsland (1998), and in these cases, it will not be possible to achieve feasibility using just these two types of move. Consider the following example in Figure 4.2.

Cover required:

Days								Nights							
S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21

Number of nurses available:

		Grade		
		1	2	3
Type	(5,4)	3	3	4
	(4,3)	2	2	2
	(3,3)	0	0	2

Then consider the following situation where all nurses have been assigned to either days or nights.

Grade	Days	Tot.	Nights	Tot.
1	(5,4) × 2 (4,3) × 1	14	(5,4) × 1 (4,3) × 1	7
2	(5,4) × 2 (4,3) × 1	28	(5,4) × 1 (4,3) × 1	14
3	(5,4) × 3 (4,3) × 1 (3,3) × 2	53	(5,4) × 1 (4,3) × 1	21

All nurses have been assigned, but there is one staff shortage on the day shifts at grade 3. Notice that this situation could easily be rectified by moving the grade 3, (4,3) nurse on nights to days and moving one of the (3,3) nurses to nights. This would mean that the same number of night shifts would be available, but the (4,3) nurse would be able to supply the extra shift required on days. However, the local search would never be able to rectify this situation.

Figure 4.2. Example of a situation for which the local search would not be able to obtain a feasible solution due to an initially 'unbalanced' allocation of nurses. (Continued on next page).

No straightforward swapping of patterns could be employed to perform the switch, since a (4,3) nurse may only swap shift patterns with other (4,3) nurses and the same goes for the (3,3) nurses. Equally, simply changing the pattern of one nurse at a time would not solve this problem, since any move of either the (4,3) nurse to days or one of the (3,3) nurses to nights would result in an immediate decrease in feasibility and so would never be accepted.

Figure 4.2 contd. Example of a situation for which the local search would not be able to obtain a feasible solution due to an initially 'unbalanced' allocation of nurses.

Dowsland (1998) overcame this problem using powerful neighbourhoods involving chains of several moves. Rather than allow the introduction of such complicated neighbourhoods, we instead utilise the knapsack model as a way to ensure situations such as that in Figure 4.2. do not occur so that all solutions created have the potential to be made feasible by the local search. Our aim is therefore to introduce a powerful look-ahead mechanism which will guide the construction in such a way as to make them a good starting point for the local search in terms of feasibility. We base this decision on observations from Aickelin and Dowsland (2004) and Dowsland (1998). They suggest that the types of solution landscape produced by our evaluation function and neighbourhood structures, which will be discussed more fully in the next section, are likely to consist of subspaces separated by 'ridges' so that the improvement phase will be restricted to a single sub-space, even if we were to allow some small uphill steps. These ridges are caused by the changes in cover that result in moving nurses of different types from days to nights and vice versa. The subspaces correspond to numbers of day/night allocations of different types, and for tight problems many of these subspaces will not contain any feasible solutions. Figure 4.2 is one such example of this. This problem can be overcome if we can design a construction heuristic that only produces solutions that lie in those subspaces that also contain some feasible solutions and it is with this aim in mind that we introduce the knapsack model.

As mentioned in Chapter 2, Dowsland and Thompson (2000) observe that the fact that nurses work a different number of night shifts and day shifts means that it is not possible to determine whether or not a problem is feasible simply by counting the

number of shifts available. They go on to show that this problem can be overcome using a knapsack model. We will use this model to ensure that our solutions lie in the right sub-spaces. As we will show, this is equivalent to ensuring that the day/night allocations returned by the construction phase have sufficient numbers of nurses to meet the total covering requirements on both days and nights for all grades g. It is necessary to solve the model at all three grades, since a solution found which meets the total requirements at grade 3 must also provide adequate cover for the problem at grades 1 and 2 and at grade 1. Figure 4.3 gives an example of a situation where it is possible to find a feasible allocation of nurses at each grade individually, but where no one solution exists which is compatible at all three grades.

Cover required:

Days								Nights							
S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21

Number of nurses available:

		Grade		
		1	2	3
Type	(5,4)	0	2	2
	(4,3)	1	2	4
	(3,3)	4	0	2
	(3,2)	1	1	0
	(1,1)	2	0	1

Note that this is not a very realistic setup, since there are generally a high proportion of grade 1 and full-time nurses, but provides an effective example.

Figure 4.3. Example of a situation where a feasible assignment is possible at each grade, but no compatible solution at each grade exists. (Continued on next page).

Now to provide a feasible assignment covering grade 3, which incorporates all three grades of nurse and the total cover requirement, we may have the following assignments.

Grade	Days	Tot.	Nights	Tot.
1	(4,3) × 1 (3,2) × 1	7	(3,3) × 4 (1,1) × 2	14
2	(5,4) × 2 (4,3) × 2 (3,2) × 1	28		14
3	(5,4) × 2 (4,3) × 4	54	(3,3) × 2 (1,1) × 1	21

where the columns labelled 'Tot.' provide the cumulative totals for the day and night cover at each grade.

Here the overall cover requirement has been fulfilled but it is possible to see that while the combined cover at grades 1 and 2 is also satisfied, the cover at grade 1 is deficient. However, there is a feasible allocation at grade 1, since moving two (3,3) nurses and one (1,1) nurse from nights to days would solve this problem. However, in order for the assignment to remain feasible at grades 2 and 3, some nurses at these lower grades would be required to move from days to nights in order to compensate. However, there are no nurses currently scheduled to work days who work an equal number of day and night shifts and therefore by moving these grade one nurses from nights, we effectively lose a number of day shifts. A (3,3) nurse removed from nights would have to be replaced by a (4,3), for example; the number of night shifts worked is therefore unaltered but the number of day shifts is reduced by 1. Given that there are no nurses on days who work an equal number of day and night shifts, the number of day shifts at grades 2 and 3 must necessarily be reduced by correcting the grade 1 cover. Since the day cover at grades 2 and 3 is equal to the requirement, there is no way for a feasible assignment for nurses to be compatible at all grades.

Figure 4.3 contd. Example of a situation where a feasible assignment is possible at each grade, but no compatible solution at each grade exists.

Thus it is necessary to ensure that any solution to the knapsack problem which meets the overall covering requirements is compatible with the solution to the smaller problem at the higher grades.

Essentially Dowsland and Thompson show that there are sufficient nurses to meet the cumulative cover constraints for grade g if the solution to the knapsack problem:

$$\max Z = \sum_{t=1}^{T} e_t y_{tg} \qquad (4.14_g)$$

$$st \quad \sum_{t=1}^{T} d_t y_{tg} \leq \sum_{t=1}^{T} d_t n_{tg} - D_g \qquad (4.15_g)$$

$$y_{tg} \leq n_{tg} \qquad \forall t \qquad (4.16_g)$$

is at least E_g, where

$t = 1,...,T$ is the nurse nurse type index

d_t = number of day shifts worked by a nurse of type t

e_t = number of night shifts worked by a nurse of type t

n_{tg} = number of nurse of type t of grade g or above

$D_g = \sum_{k=1}^{7} C_{kg}$ (i.e. the total day requirement at grade g and above)

$E_g = \sum_{k=8}^{14} C_{kg}$ (i.e. the total night shift requirement at grade g and above)

and y_{tg} is the number of nurses of type t and grade g or above assigned to nights.

The expression in the objective function, (4.14_g) is the total number of night shifts covered, constraint (4.15_g) ensures that there are sufficient day-shifts remaining and constraints (4.16_g) ensure that the number of nurses allocated does not exceed those available. For a given problem instance P (i.e. for given T, d_t, e_t, n_{tg}, D_g and E_g) we will denote the problem of finding a solution to (4.14_g), (4.15_g), (4.16_g) of value at least E_g by $KS_g(P)$. Clearly, if P is feasible there must be solutions satisfying $KS_g(P)$ for g = 1, 2 and 3. However, as shown in Figure 4.3 this is not sufficient, as we need

to ensure that the three solutions are compatible with one another. In order to ensure a compatible set of solutions we need to add the constraint:

$$0 \leq y_{tg+1} - y_{tg} \leq n_{tg+1} - n_{tg} \quad \forall t, g = 1,2 \tag{4.17}$$

The left hand inequality ensures that the number of night allocations at grade g+1 and above is at least as many at that at grade g and above, and the right hand inequality ensures that there are sufficient grade g+1 nurses to make up the difference. We will refer to the set of constraints given by $KS_1(P)$, $KS_2(P)$, $KS_3(P)$ and (4.17) by Cons(P). For a given problem instance, P, we ensure that our constructions always satisfy Cons(P) as follows.

Let $P(R')$ denote a partial solution to instance P in which the number of day and night shifts worked by those nurses in the set R' is known. We start by defining R^{both} as the set of nurses who work both days and nights, and R^{fixed} as the set of nurses who must be allocated to days together with those who must be allocated to work nights. Let R^{+} be the set of nurses already allocated by the construction phase. Then the remaining nurses can be allocated to days and nights in such a way as to satisfy Cons(P) if there is a feasible solution to Cons($P(R^{\text{both}} \cup R^{\text{fixed}} \cup R^{+})$). Therefore during the construction phase, once a nurse i and shift j have been selected, if $i \notin R^{\text{both}} \cup R^{\text{fixed}}$, we attempt to find a feasible solution to Cons($P(R^{\text{both}} \cup R^{\text{fixed}} \cup R^{+} \cup \{i\})$). If there is a solution then the allocation is made and the construction moves on to step 5, as given by Figure 4.1. If not, then, if j is a night shift, nurse i must be restricted to days and vice versa. We therefore restrict the set of feasible shift patterns for nurse i accordingly and return to step 2 without making the allocation. The restriction is enforced for the remainder of the construction phase only. It remains to be shown that we can determine whether or not there is a feasible solution to Cons($P(R')$) for any P and R' at minimal computational expense.

Our approach is based on the following observations.

Observation 1.

For any R' we can obtain Cons($P(R')$) from Cons(P) by setting $n_{tg} = n_{tg}$ - 1 for each nurse of type t and grade g or above in R' and reducing D_g and E_g by the number of day and night shifts of grade g or above covered by the assignment of nurses in R'.

Observation 2.

Given a solution Y_g* to $KS_g(P(R'))$, any solution to the knapsack problem:

$$\max Z = \sum_{t=1}^{T} e_t y_{tg-1} \qquad (4.14_{g-1})$$

$$st \quad \sum_{t=1}^{T} d_t y_{tg-1} \leq \sum_{t=1}^{T} d_t n_{tg-1} - D_{g-1} \qquad (4.15_{g-1})$$

$$y_{tg-1} \leq \min(n_{tg-1}, y^*_{tg}) \qquad (4.18_{g-1})$$

$$y_{tg-1} \geq \max(0, y^*_{tg} - (n_{tg} - n_{tg-1})) \qquad (4.19_{g-1})$$

satisfying $Z \geq E_{g-1}$ is a compatible solution to $KS_{g-1}(P(R'))$.

Note that the lower bounds given by (4.18_{g-1}) are obtained from (4.16_{g-1}) and the left-hand part of (4.17) and the upper bounds given by (4.19_{g-1}) are simply the right-hand part of (4.17). We denote this problem $KS_{g-1}(P(R')|Y_g^*)$

Observation 3.

The usual way to solve the knapsack problem is to use the algorithm given by Martello and Toth (1990), which involves solving the linear programming relaxation and branching on the functional variables. However, this approach seeks to find the optimal assignment, and will remove suboptimal solutions from the search tree. We are not looking for an optimal solution, but potentially require all feasible solutions with the desired objective value. We cannot, therefore, use this approach as we may remove branches which still contain feasible solutions. Since we want to use the heuristic as the method of assignment and only refer to the knapsack model to check our solution is balanced, removing branches which contain feasible solutions could mean that a desired solution is overlooked; the knapsack model would be capable of labelling certain assignments as unacceptable even when there is a balanced day/night partition which includes it. Instead, we use a branch and bound tree search in which

the branches at level i represent the set of feasible values for the ith variable. The problems represented by the nodes in the tree are smaller knapsack problems in which some of the variables have been fixed. Therefore local lower bounds can be obtained from the linear programming relaxations of these problems. This involves ordering the items in value/weight order, given by e_t/d_t, and filling the knapsack with as many units of the highest priority item as possible before considering the next item. Considering the value attained by filling the knapsack exactly, by including a fractional amount of the first item not to fit fully, allows the calculation of an upper bound. In order to reach good solutions quickly and make the bounds easier to calculate the variables are considered in decreasing e_t/d_t order and the branches are considered in decreasing value order. The tree is pruned by backtracking whenever the bound is less than Z_{target} and the bounds are recalculated every time the search backtracks. See the example given later in this section, in conjunction with Figure 4.5, for an illustration of the solution process.

We can make the process more efficient by noting that we do not necessarily need to solve a new knapsack problem every time a new allocation is considered.

We can use the above observations to solve the appropriate Cons($P(R')$) at each stage of the construction using a hierarchical tree search. As we are unable to build the different grades directly into the knapsack model, we first solve the problem for all grades of nurse. The tree at the top of the hierarchy represents a branch and bound search for all solutions of $KS_3(P(R'))$. Once a solution has been found, represented by a terminal node, Y_3*, we then reduce the problem to the nurses of grades one and two and the cover requirement at grade two only, and try to find a solution at this level compatible with the overall solution. As was shown in Figure 4.3, it is not enough to find a feasible allocation of nurses for the overall problem, the problem at grades 1 and 2 and the problem at grade 1 only since we require a single allocation which satisfies the requirements at all grades and simply knowing that a feasible solution exists at all grade levels is not enough; for a workable template, we require an allocation which will be feasible at all grades, simultaneously. Thus it is essential that the knapsack model be used in an iterative fashion to produce solutions which are feasible at each grade $g - 1$, while remaining compatible with the solution at grade g.

Each terminal node, Y_3*, spawns a new tree representing a branch and bound search for solutions to problem $KS_2(P(R')|Y_3^*)$. Similarly the nodes representing these solutions spawn new trees representing a branch and bound search for solutions to $KS_1(P(R')\,|Y_2^*)$. The whole structure is searched using a depth first strategy. Once a feasible solution with respect to grade g is found the search proceeds to the next tree in the hierarchy i.e. grade $g-1$. If the search is not successful then control is returned to the tree relating to grade g until the next feasible solution is found. When a solution is reached at the third hierarchy i.e. grade 1, then a feasible solution to Cons($P(R')$) has been obtained and the allocation of nurse i to shift j can be made. If the search terminates without finding a solution at grade 1 then Cons($P(R')$) is infeasible and the current allocation cannot be accepted.

Once appropriate solutions have been found at all grades, the number of nurses of each type and grade who must work days and nights is known. This process is completed before a construction heuristic is executed. At any point in the construction we have the y_{tg} values for a feasible solution to Cons($P(R')$) and each time a nurse is allocated these values will be reduced accordingly. As the heuristic picks a nurse and shift pattern, implying a day/night allocation, and before making the assignment, it consults this template to check whether there are any remaining allocations for this nurse type to work this sort of pattern. If we then consider allocating nurse i of type t and grade g to nights, then the existing solution will still be feasible as long as $y_{tg} \geq 1$ and (if $g \neq 1$) $y_{tg} > y_{tg\text{-}1}$. Similarly a day allocation will be feasible if $n_{tg} - y_{tg} \geq 1$ and $n_{tg} - y_{tg} > n_{tg\text{-}1} - y_{tg\text{-}1}$ If these conditions hold we can allocate nurse i without solving a new knapsack problem. Having made the allocation we update the variables and constants to reflect the new solution.

If, however, the template will not allow nurse i to work the selected pattern, rather than immediately returning to the construction heuristic to make a new choice, we continue searching the updated knapsack tree to look for a feasible allocation of nurses which will allow this pairing. If one is found, we can accept the heuristic's choice, update the new template and continue to the next stage of the construction. Otherwise, we must reject this choice and return to the heuristic to pick again, noting

that this nurse must now work the opposite type of pattern to the one originally selected.

The process is summarised in Figure 4.4.

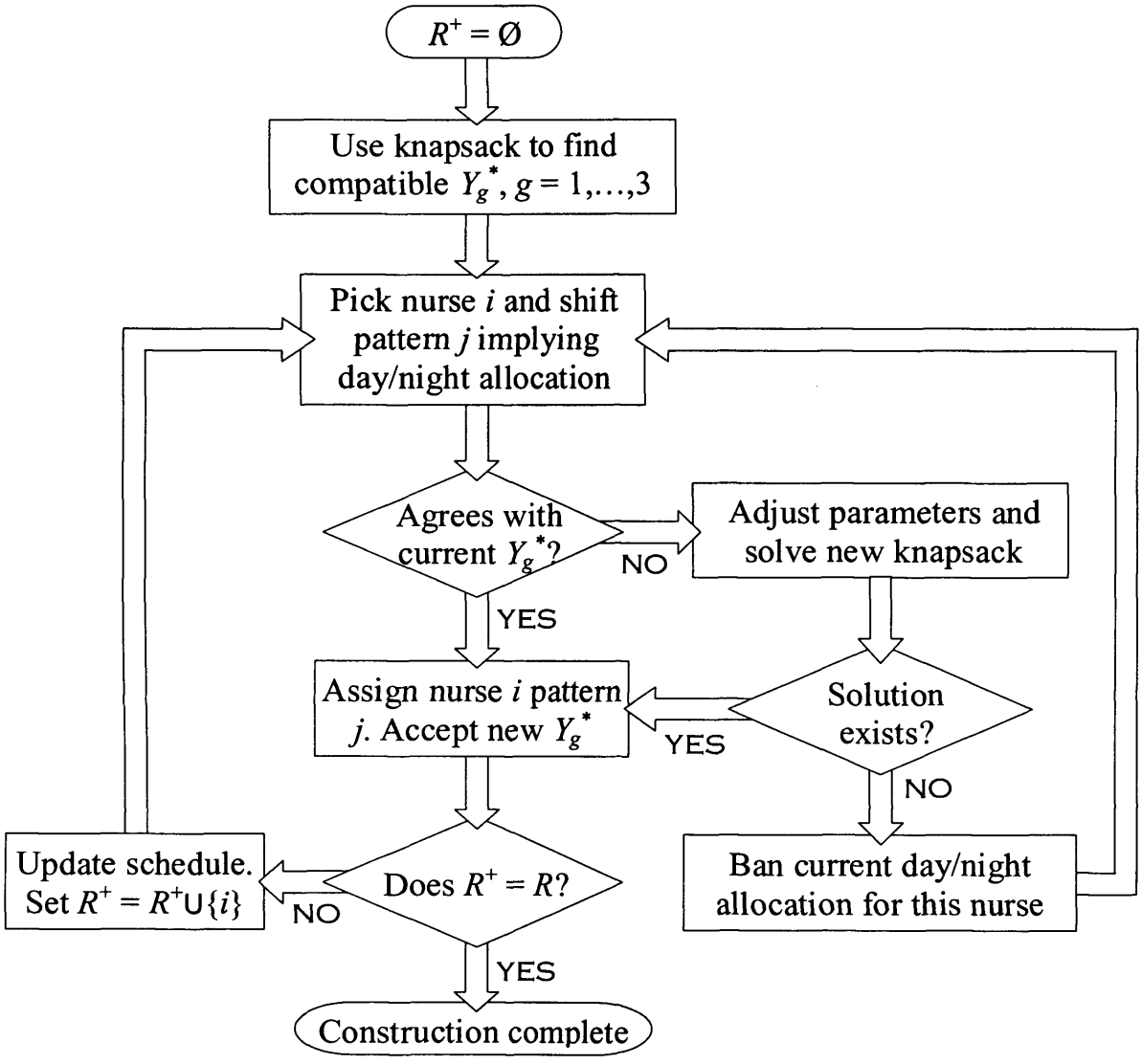


Figure 4.4. Flow chart showing implementation of the knapsack.

We now give an example of the knapsack implementation as applied to a partial solution.

Let the remaining unscheduled nurses be given by

		e_t/d_t	Grade 1	2	3
Type	1: (3,3)	1.00	0	1	1
	2: (5,4)	0.80	2	3	4
	3: (4,3)	0.75	2	2	3
	4: (3,2)	0.67	0	2	3

Note that the nurses available are now given cumulatively at each grade so, for example, there is only one nurse of type 1, a grade 2 nurse, who can cover shifts at both grade 2 and grade 3.

We number each type of nurse, in decreasing order of e_t/d_t. Let the remaining total cumulative cover requirement for days and nights at each grade be given by

Grade	Days	Nights
1	8	7
2	19	10
3	26	14

Finally let the current knapsack solution Y^* be given by

Grade	Days	Tot.	Nights	Tot.
1	(5,4) × 1 (4,3) × 1	9	(5,4) × 1 (4,3) × 1	7
2	(5,4) × 1 (3,2) × 2	20	(3,3) × 1	10
3	(4,3) × 1 (3,2) × 1	27	(5,4) × 1	14

where the columns labelled 'Tot.' give the cumulative totals for the day and night cover at each grade. Now assume the heuristic picks a nurse i to work nights, where i is a type 4, (3,2) nurse of grade 2. First we check if this is compatible with the current knapsack solution, Y^*, and see that it is not, since the current solution only allows type 4 nurses to work day shifts. We therefore re-solve the knapsack problems to check whether a solution exists which will allow this current allocation. That is, we solve Cons($P(R' \cup \{i\})$), where i is assumed to be working nights. This gives a new set of available nurses, with one grade 2, type 4 nurse removed and a new set of night

requirements with these 2 night shifts removed from grades 2 and 3. The new total night requirement is therefore cumulatively 7, 8 and 12 at grades 1, 2 and 3 respectively. The knapsack capacities given by the right hand side of equations (4.15_g) are 10 at grade 1, 10 at grade 2 and 15 at grade 3. Figure 4.5 shows the tree relating to the search for a feasible solution.

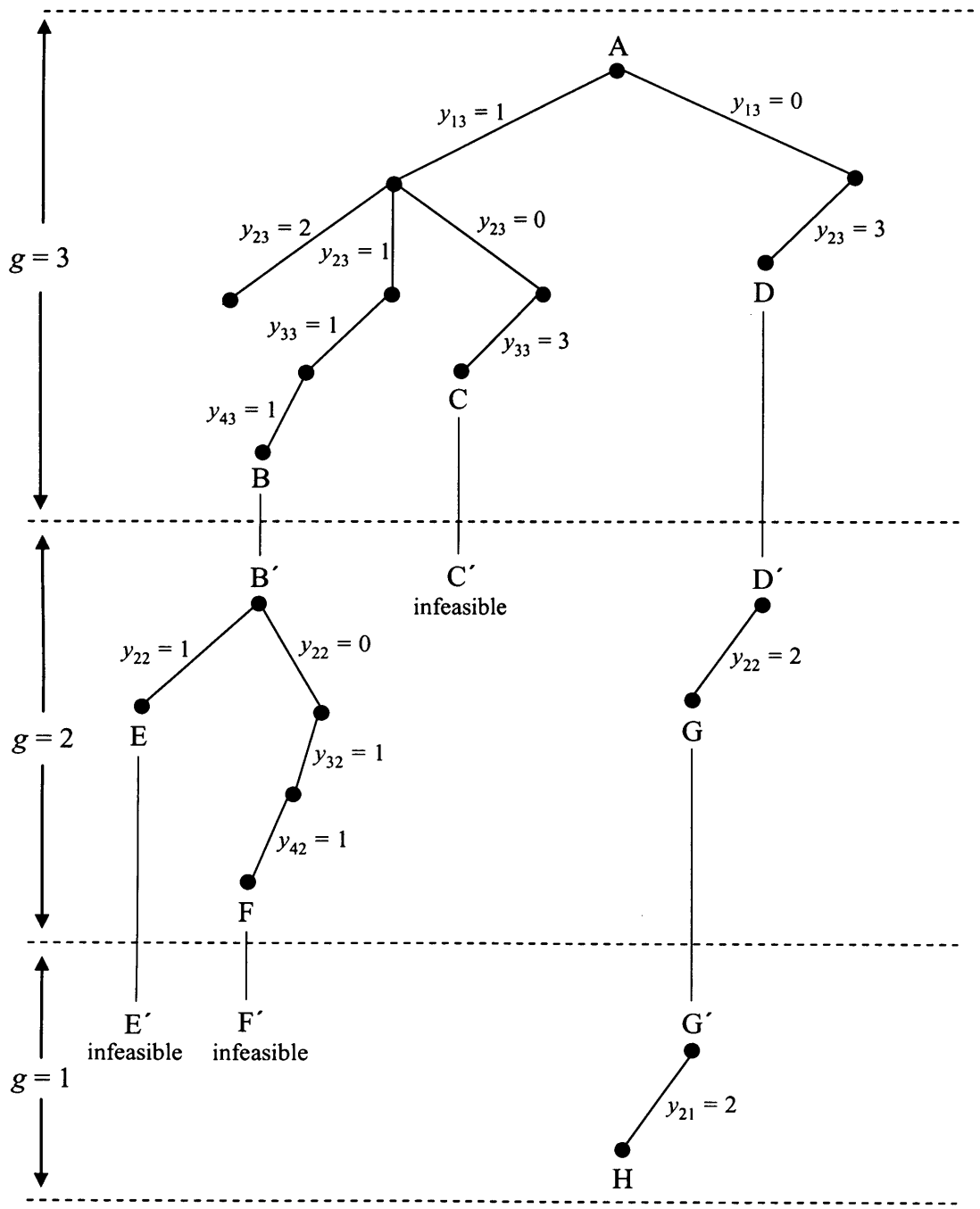


Figure 4.5. The tree relating to the search for a feasible solution.

At the top level of the search, node A represents the knapsack problem KS_3 relating to $g = 3$, with target value $E_3 = 12$, knapsack capacity = 15 and the upper bounds on the variables given by the total number of nurses available of each type. The top segment of Figure 4.5, that is, the region denoted $g = 3$, shows a tree search for all feasible solutions to this problem with objective value at least 12, using bounds based on the linear programming relaxation of the problem to prune the tree. Note that the types are considered in decreasing e_t/d_t order and the branches relating to each variable are ordered in decreasing value order. This improves the efficiency of the search.

The type 1 nurses are therefore the first to be considered. There is only one such nurse available, with a 'weight' of 3 days. The first branch to be considered is therefore $y_{13} = 1$ and this leaves a remaining capacity of 15 – 3 = 12, enough space for 12/5 = 2.4 type 2 nurses. This gives us an initial lower bound of 1×3 + 2×4 = 11, and an upper bound of 1×3 + 2.4×4 = 12.6. However, only whole nurses can be considered and with one type 1 nurse and two type 2 nurses, there is a remaining capacity of 15 – 3 – 2×5 = 2, which means no more nurses would be able to work nights without violating the capacity. Since the current branch only gives 11 night shifts, which is less than the target value, we backtrack to the previous branching point and consider instead allocating just one type 2 nurse. At this point the bounds are recalculated. With one type 1 nurse and one type 2 nurse, there is a remaining capacity of 15 – 3 – 5 = 7. This leaves enough space for 7/4 = 1.75 type 3 nurses, giving a new upper bound of 1×3 + 1×4 + 1.75×3 = 12.25. The process continues in this manner. Obviously once a complete solution at grade 3 is found, a compatible grade 2 and then grade 1 solution must be obtained. Figure 4.5 shows the full search tree.

The first solution is found at node B and is given by $Y_3^* = (1,1,1,1)$, with 1 nurse of each type working nights and the remainder on days. Having reached the solution we spawn a new tree to search for solutions to the problem $(KS_2|Y_3^*)$ starting at node B′. The target value is now 8 and the knapsack capacity is 10. The vector of upper bounds, given by the right-hand side of constraint set (4.18_2) is (1,1,1,1) and the vector of lower bounds, given by the right-hand side of (4.19_2) = (1,0,0,0). Thus y_{12} is fixed at $y_{12} = 1$. Substituting the fixed value leaves us with a problem in the three remaining variables with an adjusted target of 8 – 3 = 5 and adjusted knapsack

capacity of 10 – 3 = 7. The tree search corresponding to this problem starts at root node B′ and the first feasible solution is found at E corresponding to $Y_2^* = (1,1,0,0)$.

We now spawn a new tree to search for solutions to $(KS_1|Y_2^*)$. The target is 7 and the knapsack capacity is 12. The upper bounds are given by (0,1,0,0) and the lower bounds by (0,0,0,0). Thus all variables except y_{32} are fixed at 0 and as y_{32} has an upper bound of 1 the problem is clearly infeasible as we cannot meet the target value. We therefore return to node E and continue searching the tree corresponding to $(KS_2|Y_3^*)$. A second solution at this level in the hierarchy is found at F with $Y_2^* = (1,0,1,1)$. This spawns a new problem $(KS_1|Y_2^*)$ with upper bounds (0,0,1,0) and lower bounds (0,0,0,0). Once again we have a problem in just one variable which is clearly infeasible so control is returned to node F. Continuing the search at this level shows that there are no more feasible solutions to $(KS_2|Y_3^*)$ and control is returned to point B and the search for solutions to problem A continues.

The next solution is at C corresponding to $Y_3^* = (1,0,3,0)$. This spawns a new problem for $(KS_2|Y_3^*)$ at C′ with upper bounds (1,0,3,0) and lower bounds (1,0,2,0). This becomes a problem in y_{32} with a target of 5 and knapsack capacity = 7. This problem is clearly infeasible as the lower bound on y_{32} causes the knapsack capacity constraint (4.15_2) to be violated. Control therefore returns to C. The next solution occurs at D corresponding to $Y_3^* = (0,3,0,0)$ with upper bounds (0,3,0,0) and lower bounds (0,2,0,0). The tree for the resulting problem in y_{22} starts at D′ yielding a feasible solution at G corresponding to $Y_2^* = (0,2,0,0)$. This spawns problem $(KS_1|Y_2^*)$ at G′ with upper bounds (0,2,0,0) and lower bounds (0,1,0,0) and yielding a feasible solution at H. Thus there is a feasible solution that is compatible over all grades given by

Grade	Days	Tot.	Nights	Tot.
1	(4,3) × 2	8	(5,4) × 2	8
2	(3,3) × 1 (5,4) × 1 (3,2) × 1	19	—	8
3	(4,3) × 1 (3,2) × 1	26	(5,4) × 1	12

where the columns labelled 'Tot.' give the cumulative totals for the day and night cover at each grade. We therefore make the allocation of nurse i to pattern j. Had the search failed to find a feasible solution nurse i would have been added to the set R^{fixed} (tagged as days only), the matrix Y^* restored to its previous set of values and n_{24} and n_{34} reduced by 1 and D_2 and reduced by 3.

We incorporate this look-ahead rule based on the knapsack calculations described in this section into each of our 7 construction heuristics to form 7 further variants. Figure 4.6 describes a full construction incorporating this knapsack routine.

Total cover required at each grade:

Grade	Days	Nights
1	14	7
2	28	14
3	91	21

Available nurses of each type at each grade:

Type	Grade1	Grade2	Grade3
(5,4)	6	4	7
(4,3)	3	1	1
(3,2)	2	0	0
(3,3)	1	1	0

Initial solution created by the knapsack for use as a template during the construction:

Grade	Days	Tot.	Nights	Tot.
1	(5,4) × 5 (4,3) × 3 (3,2) × 2	43	(5,4) × 1 (3,3) × 1	7
2	(5,4) × 3 (4,3) × 1	62	(5,4) × 1 (3,3) × 1	14
3	(5,4) × 5 (4,3) × 1	91	(5,4) × 2	22

where the columns labelled 'Tot.' provide the cumulative totals for the day and night cover at each grade. Let the notation $(d,e)_g \rightarrow \mathrm{D}$ indicate that the construction heuristic is recommending the assignment of a (d,e) nurse of grade g to days and $(d,e)_g \rightarrow \mathrm{N}$ indicate the nurse is being assigned to nights. If assignments which do not agree with the knapsack template are indicated with an asterisk (*), we may track the progress of each stage of the construction as follows.

Figure 4.6. Example construction of a solution from a real dataset, demonstrating the incorporation of the knapsack model. (Continued on next page).

Stage	Suggested assignment
1	$(5,4)_2 \rightarrow$ D
2	$(5,4)_2 \rightarrow$ D
3	$(5,4)_3 \rightarrow$ D
4	$(5,4)_1 \rightarrow$ D
5	$(3,3)_1 \rightarrow$ D*

The first four assignments agree with the knapsack template and so the assignments can be accepted. However, the fifth stage of the construction places one of the (3,3) nurses onto days, when the template has both of these nurses working nights. In order to check whether a feasible set of assignments exist where this nurse is working days, we attempt to solve the new knapsack problem where the cover required and number of nurses available are both reduced to exclude the current nurse and all nurses which have previously been scheduled. Therefore, at this point the knapsack model would be required to solve the problem where the required cover has been reduced to

Grade	Days	Nights
1	6	7
2	10	14
3	68	21

Note that the night shift requirements are unaltered as no nurses have yet been assigned to work on nights. The available nurses at each grade are now:

Type	Grade1	Grade2	Grade3
(5,4)	5	2	6
(4,3)	3	1	1
(3,2)	2	0	0
(3,3)	0	1	0

Figure 4.6 contd. Example construction of a solution from a real dataset, demonstrating the incorporation of the knapsack model. (Continued on next page).

The knapsack model produces a new solution which does satisfy these new criteria:

Grade	Days	Tot.	Nights	Tot.
1	(5,4) × 3 (4,3) × 3 (3,2) × 1	30	(5,4) × 2 (3,2) × 1	10
2	(5,4) × 1 (4,3) × 1	39	(5,4) × 1 (3,3) × 1	17
3	(5,4) × 5 (4,3) × 1	68	(5,4) × 1	21

where the columns labelled 'Tot.' provide the cumulative totals for the day and night cover at each grade.

The assignment $(3,3)_1 \rightarrow \mathrm{D}$ is therefore accepted and the new template referred to in subsequent stages of the construction. The construction continues with the following assignments.

Stage	Suggested assignment
6	$(4,3)_1 \rightarrow \mathrm{D}$
7	$(3,2)_1 \rightarrow \mathrm{D}$
8	$(4,3)_2 \rightarrow \mathrm{D}$
9	$(5,4)_3 \rightarrow \mathrm{D}$
10	$(4,3)_3 \rightarrow \mathrm{D}$
11	$(5,4)_1 \rightarrow \mathrm{D}$
12	$(4,3)_1 \rightarrow \mathrm{D}$
13	$(5,4)_1 \rightarrow \mathrm{N}$
14	$(3,2)_1 \rightarrow \mathrm{N}$
15	$(3,3)_2 \rightarrow \mathrm{D}$ *

At this stage of the assignment process, the construction heuristic suggests putting the other (3,3) nurse onto day shifts. However, this time the assignment $(3,3)_2 \rightarrow \mathrm{D}$ is rejected and thus we know this nurse must work nights.

Figure 4.6 contd. Example construction of a solution from a real dataset, demonstrating the incorporation of the knapsack model. (Continued on next page).

By disallowing future day shift patterns for this nurse, we may also reduce the required amount of night cover for grades 2 and 3 by 3 shifts in future calculations. Stage 15 has therefore made no assignment as yet and we try another nurse before continuing to the next stage.

Stage	Suggested assignment
15'	$(5,4)_1 \rightarrow$ D
16	$(5,4)_1 \rightarrow$ N
17	$(5,4)_1 \rightarrow$ N *

By this time, our current template has indicated two (5,4) nurses at grade 1 should work nights; stage 17 suggests allocating a third. We therefore check the knapsack model for a feasible distribution of nurses which includes this assignment. The remaining cover requirement, given that we know a (3,3) nurse will be assigned to nights, is:

Grade	Days	Nights
1	0	0
2	0	0
3	34	4

The nurses available to cover these shifts are:

Type	Grade1	Grade2	Grade3
(5,4)	0	2	5
(4,3)	1	0	0
(3,2)	0	0	0
(3,3)	0	0	0

The knapsack is able to satisfy these constraints and produces the following new template for use with the remainder of the construction.

Figure 4.6 contd. Example construction of a solution from a real dataset, demonstrating the incorporation of the knapsack model. (Continued on next page).

Grade	Days	Tot.	Nights	Tot.
1	(4,3) × 1	4	—	0
2	(5,4) × 2	14	—	0
3	(5,4) × 4	34	(5,4) × 1	4

where the columns labelled 'Tot.' provide the cumulative totals for the day and night cover at each grade.

For the remainder of the construction, all assignments suggested by the heuristic match with the template and the construction concludes with these assignments.

Stage	Suggested assignment
18	$(5,4)_3 \rightarrow$ D
19	$(5,4)_3 \rightarrow$ D
20	$(5,4)_3 \rightarrow$ D
21	$(5,4)_3 \rightarrow$ D
22	$(5,4)_2 \rightarrow$ D
23	$(4,3)_1 \rightarrow$ D
24	$(5,4)_2 \rightarrow$ D
25	$(5,4)_3 \rightarrow$ N
26	$(3,3)_2 \rightarrow$ N

Thus all 26 nurses have been assigned to shift patterns in such a way that the resulting solution is balanced and the local search has a good chance of making it feasible if it is not feasible already.

Figure 4.6 contd. Example construction of a solution from a real dataset, demonstrating the incorporation of the knapsack model.

Thus we have explored the three ways of balancing the search between feasibility and optimality: the *Cover* decoder looks only at the feasibility of the problem, leaving the task of finding low-preference cost solutions to the local search part of the GRASP algorithm; the *Combined*, *Holistic* and *LastChance* decoders balance these two factors with weights in different ways, with the *LastChance* decoder incorporating an element of look-ahead; and each of these algorithms, combined with the knapsack algorithm has a look-ahead procedure capable of steering the construction towards regions of the search space that may contain high-quality feasible solutions.

4.2.3. Local Search Neighbourhoods

None of the above construction heuristics is guaranteed to produce a feasible solution. Therefore the solution space for the improvement phase is the set of solutions in which each nurse is allocated to a feasible shift pattern, regardless of whether or not the covering constraints are satisfied. Our basic search strategy is that of random descent i.e. at each iteration we sample the space of neighbours of the current solution uniformly and accept an improving solution as soon as it is encountered. However, a typical data set may have many variables with identical p_{ij} values. This suggests that there may be large plateau-like areas in the solution landscape and for this reason we also experiment with accepting equal cost solutions. Our evaluation function reflects the importance of meeting the covering constraints over that of optimising the preference cost, and we define a solution as being better than its neighbour if the cover is improved or if the cover is unchanged and the penalty is improved.

More precisely our evaluation function is defined as:

$$\zeta = \sum_{i \in R} \sum_{j \in S_i} p_{ij} x_{ij} + W \cdot \sum_{k=1}^{14} \sum_{g=1}^{3} d_{gk}(R), \tag{4.20}$$

where $W >> p_{ij}$, $\forall i, j$.

Our objective is to exploit the construction part of the GRASP algorithm, in order to determine whether it is possible to produce good solutions without the use of complex local search neighbourhoods. We therefore introduce three simple neighbourhoods which, used in combination, should be flexible enough to allow sufficient exploration of the search space. The neighbourhoods considered are: a 1-opt neighbourhood, which involves changing the shift pattern of one nurse; a 2-opt neighbourhood, which involves changing the shift patterns of two nurses; and a straightforward swap neighbourhood, which allows two nurses of the same type to swap their shift patterns. The neighbourhoods are combined by selecting each with a given probability. In all cases, sampling was carried out without replacement. The next sections provide definitions of each of the neighbourhoods.

4.2.3.1. Change neighbourhood

$N_C(s)$ is the set of solutions obtained from any solution, s, by changing the shift-pattern of a single nurse. A nurse and a new shift pattern are chosen at random, such that the pattern is one which fulfils the nurse's contractual requirements. Only patterns with a preference cost lower than 100 are considered. The choice of pattern is kept for this nurse if it results in an improvement. Changing the shift pattern of a nurse is the most obvious local search method and allows for a large neighbourhood. As a move in $N_C(s)$ can change both the cover and the preference cost, in theory any solution is reachable from any other by a series of moves using this neighbourhood. However it is not sufficiently flexible in the context of our descent strategies, as if the cover constraints are tight, any solution, s, satisfying the covering will not have any improving moves within $N_C(s)$.

4.2.3.2 Swap neighbourhood

$N_S(s)$ is the set of solutions obtained from s by swapping the shift patterns of two nurses where their patterns are compatible. Two nurses are chosen at random and their shift patterns are swapped, if their contracts allow and if this results in an improvement. Again, moves where a nurse would be allocated a shift pattern with a preference cost of 100 are not included. Swapping two nurses' shifts allows for improvements to the preference cost with minimal or no change to the cover. As swapping the patterns of two nurses on days (nights) will not affect the cumulative cover at grade 3, this neighbourhood is likely to allow the search to progress further once a feasible solution is found even on tight problems. However, as swaps are limited to nurses of a single type some areas of the solution space are likely to be unreachable.

4.2.3.3 Extended neighbourhood

$N_E(s)$ is the set of solutions obtained from s by changing the shift-patterns of two nurses. This gives more flexibility than the above neighbourhoods when s satisfies the covering constraints. However it is a larger neighbourhood and it may be computationally expensive to find improving moves close to local optima when they

may be sparsely distributed. We overcome this by restricting its use to situations where s is already feasible and sampling from a candidate list of neighbours that is a superset of those that result in an improvement. This is achieved as follows. Given s, an element of $N_E(s)$ is defined by the quadruple (i_1,j_1,i_2,j_2) where i_1 and i_2 are the 2 nurses involved and j_1 and j_2 are the new patterns for these nurses. The 4 parameters are sampled in the above order with all the options for the remaining parameters being exhausted before parameters higher in the order are changed. i_1 is sampled uniformly from the full set of nurses, but the candidate sets for the remaining parameters can be considerably reduced using the following logic. The brackets in the following refer to the case where we accept moves of equal cost as well as improving moves.

Given that s satisfies the cover constraints, an improving move must result in a reduction in the preference cost. For any two nurses involved in such a move then at least one must be moved to a pattern with lower (or equal) preference cost. As we are sampling from the set of nurses uniformly we can assume that this is true for i_1. Thus, given i_1 we sample j_1 from the set of patterns with penalty cost less than (or equal to) the cost of i_1's current allocation. Further efficiency gains are possible if the patterns for each nurse are pre-sorted into p_{ij} order. If there is little or no slack in the covering constraints then moving nurse i_1 to pattern j_1 may result in some shifts being uncovered. The second half of the move must repair this. Thus nurse i_2 must be of a sufficiently high grade to cover the shortfall and must currently be off duty on all the under-covered shifts. Our candidate list for i_2 is therefore restricted to nurses who satisfy these conditions. Finally pattern j_2 must be chosen from those patterns for which the increase in preference cost for i_2 is less than (or equal to) the reduction obtained from moving nurse 1.

4.3. Experiments and results

Fifty-two datasets have been obtained from a large UK hospital and each has been pre-processed by adding bank nurses in situations where there are not sufficient nurses to satisfy demand and adding dummy nurses to equalise numbers on each shift where there were too many nurses available. Further details can be found in Dowsland and Thompson (2000) and Appendix C. The details of each dataset can be found in

Appendix A which lists the numbers of each nurse type and lists the numbers of each of these forced to work days or nights. A sample dataset is also given.

These datasets have also been considered by other researchers including Dowsland and Thompson (2000), Aickelin and Dowsland (2004), Burke et al. (2003b) and Aickelin and Li (2005). Optimal solutions have been obtained for all datasets using specialist IP software. See Fuller (1998) for a description of the IP solution process for this problem.

4.3.1 Parameter testing

Values for the parameters n, w_c and w_p are required. The values of n we try must differ between heuristics as in each case we are selecting something different. For the *Cover* and *Combined* heuristics we select n shift patterns for a nurse from the patterns available for that nurse's type, typically around 60, while for the *Holistic* heuristics we are selecting n nurse-pattern pairs from around 1,500 such pairings. For the *LastChance* heuristic, however, our list is of n nurses and we would only have about 25 of these. For each heuristic we have chosen a low value, a high value and an intermediate value for n to test. Table 4.7 shows the values investigated for each of the parameters.

Decoder	n	w_c	w_p	Total combinations
Cover	3,12,20	—	—	3
$Combined_a$	3,12,20	1,2,...,5	1,2,...,5	75
$Combined_q$	3,12,20	1,2,...,5	1,2,...,5	75
$Holistic_a$	3,10,60	1,2,...,5	1,2,...,5	75
$Holistic_q$	3,10,60	1,2,...,5	1,2,...,5	75
$LastChance_a$	3,6,10	1,2,...,5	1,2,...,5	75
$LastChance_q$	3,6,10	1,2,...,5	1,2,...,5	75

Table 4.7. The combinations of parameters investigated.

We have fourteen decoders, seven with the knapsack and seven without. For each of these, and for each of the parameter combinations, we performed 100 GRASP cycles on each of the 52 datasets. We originally experiment with just the first two, simpler, neighbourhoods, $N_C(s)$ and $N_S(s)$. The local search chooses one of these

neighbourhoods randomly for each iteration and continues until a local optimum has been reached and no moves remain within either neighbourhood.

To compare each parameter combination, results must be combined over all 52 datasets. Two scores are required: a feasibility score, to determine how many feasible solutions this combination produces compared with other parameters; and an optimality score, to determine the quality of the feasible solutions found, given that the optimal is known in each case. To measure the success of each parameter combination in terms of feasibility, we choose to compare the average number of feasible solutions found from the 100 GRASP cycles over the 52 datasets. To compare their success in terms of preference cost we will compare the average preference cost of all the feasible solutions found over the 52 datasets. The preference costs of those solutions which were not feasible are not of interest as no infeasible solutions will be accepted.

4.3.2 Choosing parameters

To choose sensible parameters for each of the 14 heuristics, we need to examine the effect of the parameter choices on both feasibility and optimality. The first consideration is that we must combine information about solution quality from the different datasets, since the optimal preference costs vary greatly between datasets. In order to compensate for this variation we standardise the results by subtracting the optimal value in each case. Figures 4.8 (a)-(g) show plots of the percentage of feasible solutions against the average standardised preference cost for these feasible solutions, over all 100 cycles, for each heuristic. Each point represents a single combination of n, w_c and w_p.

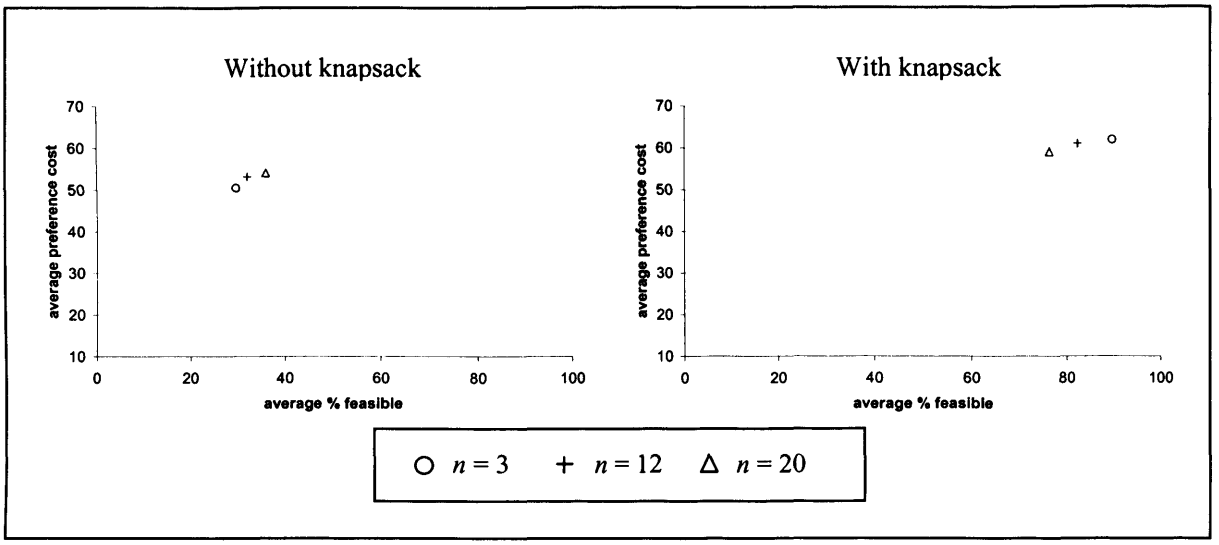


Figure 4.8 (a). Plots of average preference cost against percentage feasibility for the *Cover* heuristic with and without the knapsack, for the three values of n.

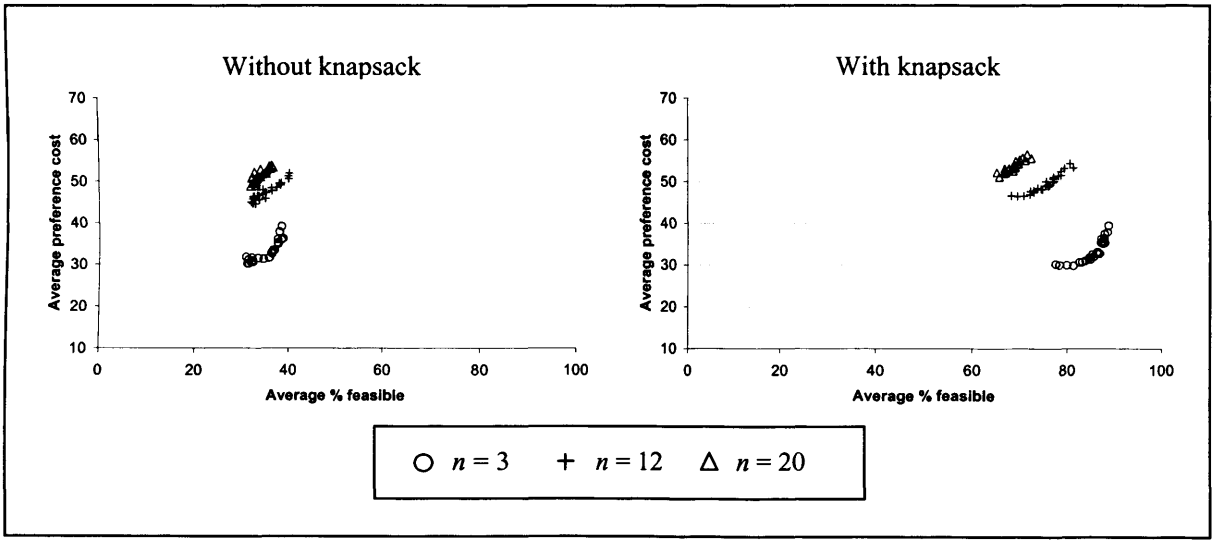


Figure 4.8 (b). Plots of average preference cost against percentage feasibility for the $Combined_a$ heuristic with and without the knapsack, for each parameter combination.

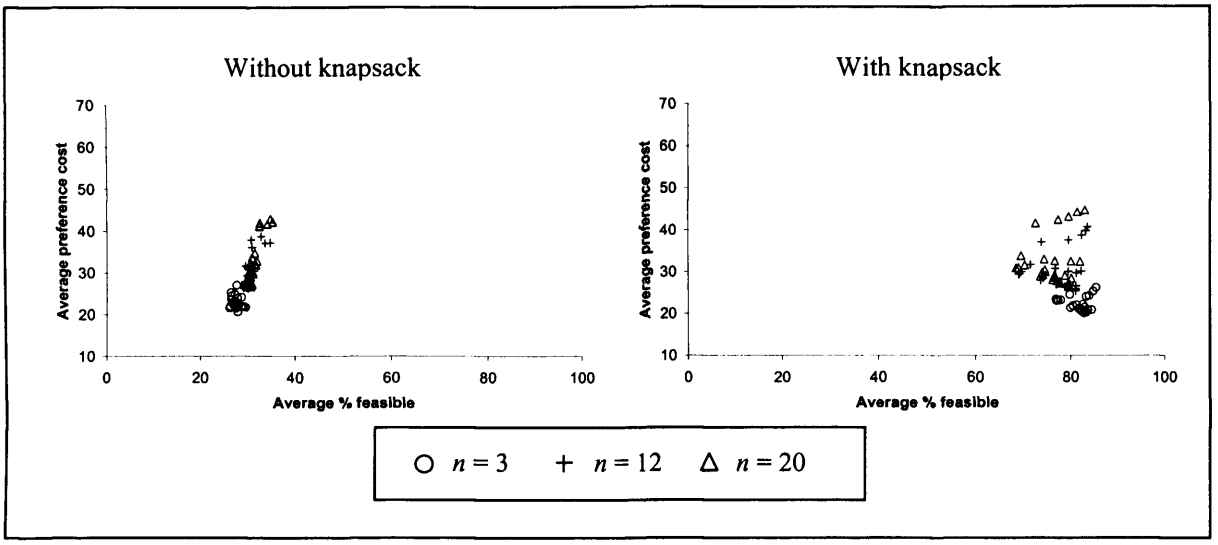


Figure 4.8 (c). Plots of average preference cost against percentage feasibility for the $Combined_q$ heuristic with and without the knapsack, for each parameter combination.

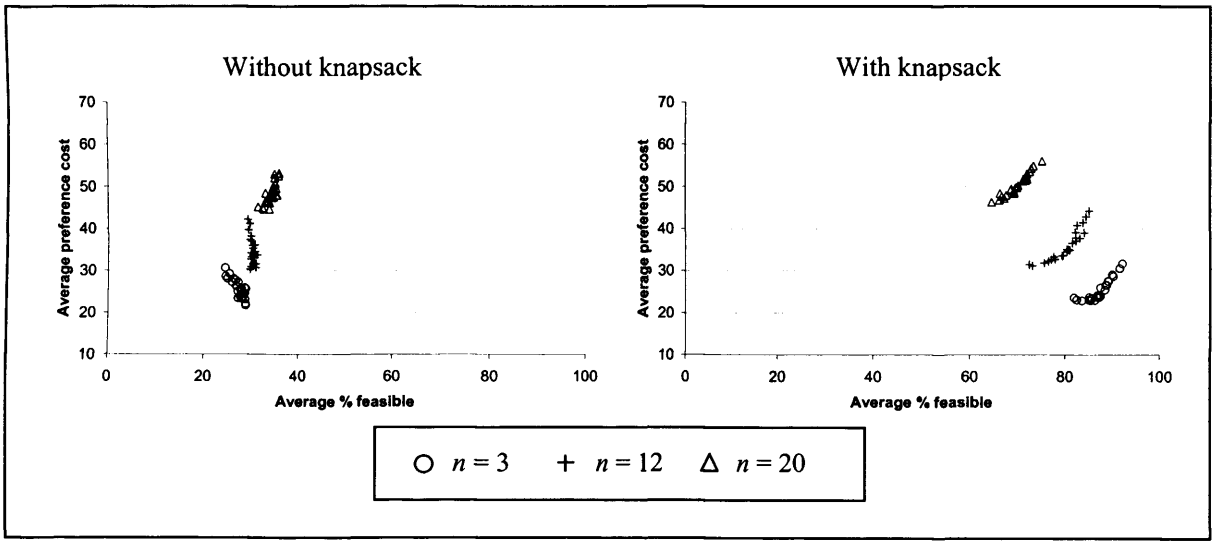


Figure 4.8 (d). Plots of average preference cost against percentage feasibility for the $Holistic_a$ heuristic with and without the knapsack, for each parameter combination.

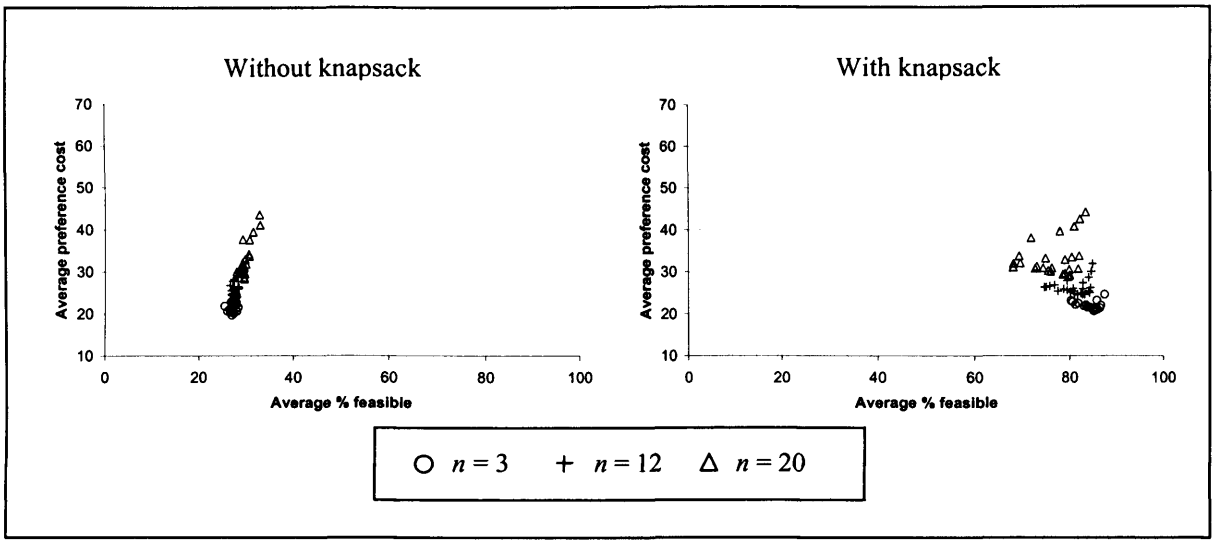


Figure 4.8 (e). Plots of average preference cost against percentage feasibility for the $Holistic_q$ heuristic with and without the knapsack, for each parameter combination.

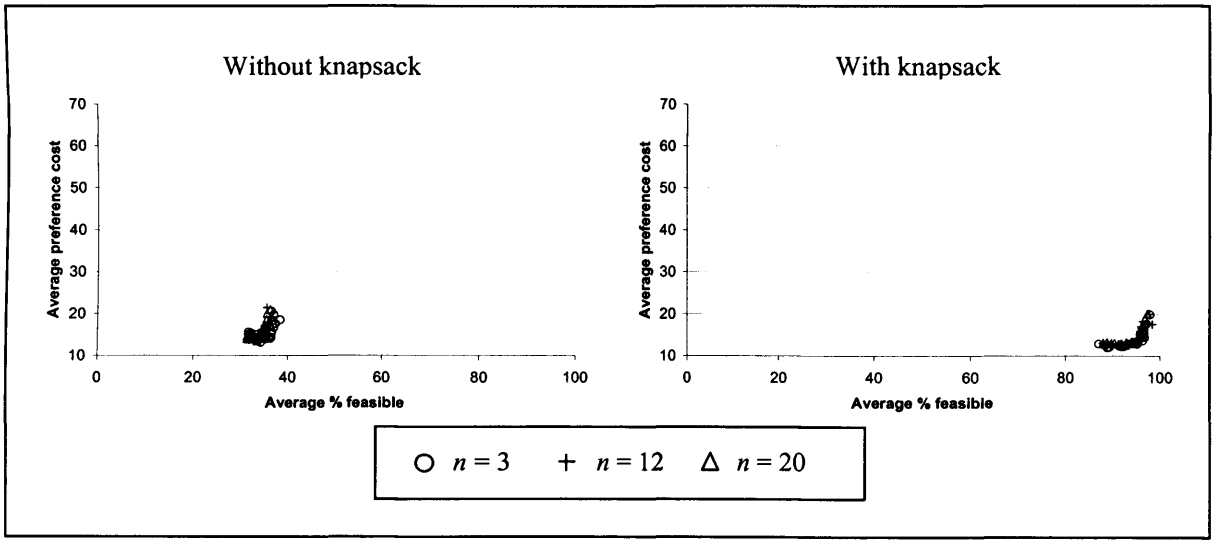


Figure 4.8 (f). Plots of average preference cost against percentage feasibility for the $LastChance_a$ heuristic with and without the knapsack, for each parameter combination.

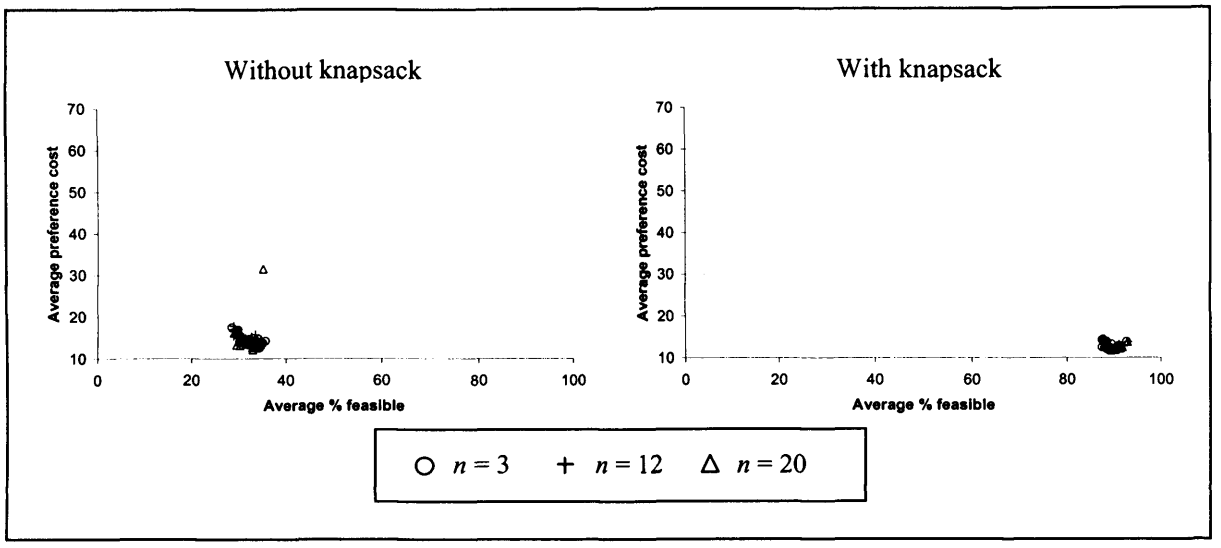


Figure 4.8 (g). Plots of average preference cost against percentage feasibility for the $LastChance_q$ heuristic with and without the knapsack, for each parameter combination.

It is clear that the seven approaches incorporating the knapsack model fare much better in terms of feasibility. Furthermore, while all knapsack-enabled approaches produced at least one feasible solution for every dataset, without the knapsack only 5 of the 453 approaches managed this, and these solutions were generally of relatively poor cost in terms of the nurses' preferences. In terms of computational expense, the incorporation of the knapsack model adds a negligible amount to the construction time and, on average, actually decreases the total solution time by a small amount. Thus, it can be deduced that the knapsack is a beneficial addition to the construction phase.

For all but the *LastChance* constructions, there is evidence that smaller values of n outperform larger ones suggesting that the added diversity of larger n is less beneficial than a greedier approach. Of the knapsack heuristics, Figures 4.8 suggest that the *LastChance* variant is the most promising. In order to confirm this, each variant was run 10 times for 100 cycles on each of the 52 data sets using a single set of parameter values. Unlike the above experiment where results were averaged over all cycles we now consider each block of 100 cycles as a complete GRASP run and record only the best solution over these cycles. Since all seven heuristics with the knapsack produce a high proportion of feasible solutions, the choice of parameters with which to continue experimentation was based on the average preference cost over the 52 datasets.

For the *Cover* heuristic with the knapsack, only three values for n were tested and it is clear from Figure 4.8 (a) that the lowest preference cost is given by $n = 20$. For all other heuristics the best parameter combination from 75 must be chosen. In order to select these parameters, plots of the average feasible preference cost against the ratio w_c/w_p were created in order to give the impression of which w_c and w_p values would be suitable. Figure 4.9 shows an example of such a plot for $Holistic_a$ with knapsack. Clearly any parameter combination with $n = 3$ and $w_c < w_p$ would be acceptable. Note that since we are considering preference cost, it is to be expected that higher values of w_p would be successful and since we know that a reasonable percentage of feasible solutions are found in all cases, we can deduce that feasibility can be obtained even with a relatively low value of w_c. Although any of these values would be likely to perform adequately, we select the value pertaining to the lowest average preference cost, which relates back to the values $w_c = 1$ and $w_p = 3$. Thus, for this heuristic, we have the parameter combination $(n, w_c, w_p) = (3, 1, 3)$.

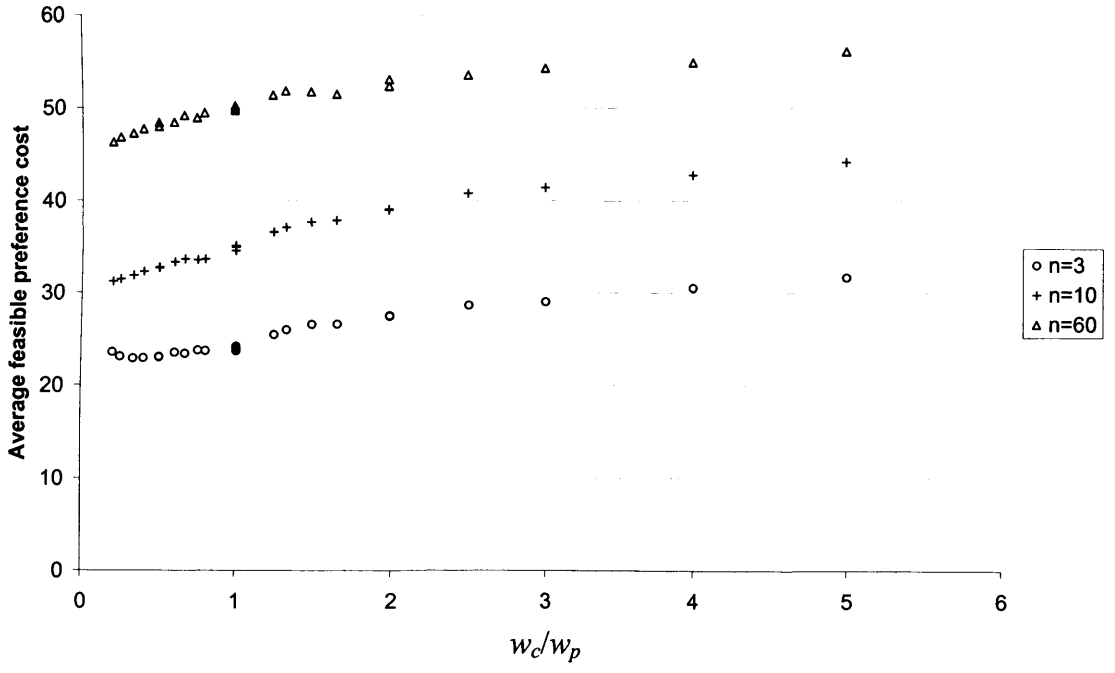


Figure 4.9. Sample plot for $Holistic_a$ with knapsack showing the average feasible preference cost against the ratio for the weights w_c/w_p for each value of n.

For all additive heuristics, these graphs of preference cost again the ratio w_c/w_p have a similar shape, but the quotient graphs produce a slightly different result as shown in Figure 4.10.

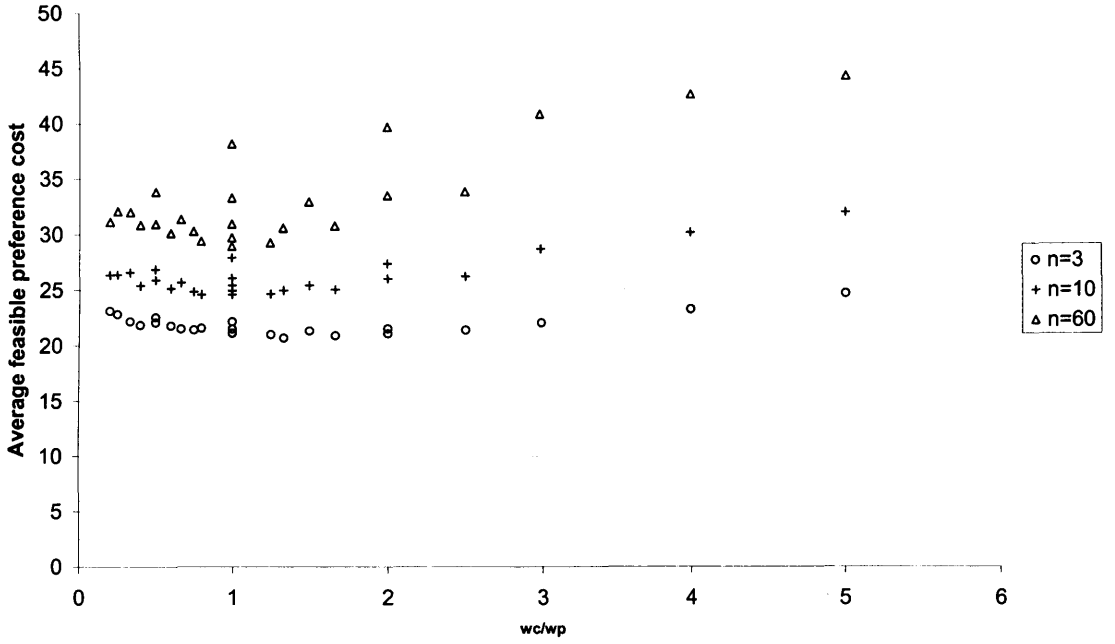


Figure 4.10. Sample plot for *Holistic*$_q$ with knapsack showing the average feasible preference cost against the ratio for the weights w_c/w_p for each value of n.

As would be expected, choosing values of w_c and w_p with the same ratio give similar results using the additive heuristics, as these experiments are essentially repeated results; variation comes only from the difference in the random seed. However, for the multiplicative method, there is obviously a difference between choosing $w_c = 1$, $w_p = 1$ and $w_c = 2$, $w_p = 2$, for example, and Figure 4.10 demonstrates this difference. We can also see that a ratio in the range (1,2) is likely to be more successful for this method. Again, this graph shape is typical of all multiplicative heuristic approaches. Although choosing high w_c and low w_p is still shown to give poorer results, the main difference between the additive and multiplicative approaches is in preference costs where w_c is significantly lower than w_p. Where this was beneficial in an additive context, the results for the multiplicative versions slightly worsen as w_c drops below w_p. A potential reason for this is that, using a multiplicative approach, selecting a much higher w_p value may lead to the influence of the cover part of the scores being negligible and so in order for good solutions to be found, a balance must be struck whereby the cover has a significant enough impact to be able to steer solutions towards feasibility. Although the local search may be able to 'correct' the feasibility of constructed solutions with a very high cover cost, the more moves required to attain

feasibility, the more significant the effect on the preference cost is likely to be. Thus, multiplicative heuristics perform better with a different ratio of w_c/w_p than their additive counterparts. Again, we select the value with the best cost in this suitable range for further study. In Figure 4.10 this value gives $w_c = 4$ and $w_p = 3$ giving the compete combination $(n, w_c, w_p) = (3, 4, 3)$. The parameters selected for each heuristic were used to produce a further 10 complete GRASP runs in order to more fairly compare each heuristic. These parameters, along with the results produced by each, are given in Table 4.11. Note that, apart from the *Cover* heuristic, all approaches produce the best results with a value of $n = 3$. The column labelled 'Cost' provides the mean best solution cost from each run, standardised by subtracting the optimal costs from each dataset. The column labelled '% Feasible' gives the percentage of all solutions produced which were feasible and '% Feasible$_c$' gives the percentage of solutions which were feasible at the end of the construction phase. Column 'Time' gives the average time in seconds taken to complete one run of 100 cycles on one dataset, while 'Time$_c$' gives the average time taken to produce just the constructions of one run.

Heuristic	n	w_c	w_p	Cost	% Feasible$_c$	% Feasible	Time$_c$	Time
Cover + kn.	20	—	—	22.92	0.07	76.40	0.08	0.73
Combined$_a$ + kn.	3	2	5	7.73	0.08	81.68	0.07	0.42
Combined$_q$ + kn.	3	5	5	3.85	0.76	83.04	0.08	0.38
Holistic$_a$ + kn.	3	1	3	3.84	0.17	83.66	1.05	1.51
Holistic$_q$ + kn.	3	4	3	3.77	2.39	85.19	0.72	1.01
Lastchance$_a$ + kn.	3	2	5	1.18	11.23	89.06	0.79	1.06
Lastchance$_q$ + kn.	3	4	3	2.03	19.03	89.64	0.67	0.91

Table 4.11. Parameters chosen for each heuristic with the knapsack.

It can be seen that all methods produce a high percentage of feasible solutions, although all methods struggle to produce feasible constructions. Of all the methods tried, the two *Lastchance* heuristics produce both the highest percentages of feasible solutions as well as the lowest average standardised preference costs. The additive version produces the solutions of the best quality, on average, and is chosen as the most successful algorithm to be the subject of further experimentation. Although the multiplicative version gives a slight improvement in terms of the percentage of feasible solutions produced, this difference is negligible considering the large number of feasible solutions that both approaches are able to produce. Although the *Holistic*

and *LastChance* heuristics take longer to produce 100 cycles, as is to be expected since they consider all nurses at each stage, all methods are fairly fast, taking around just 1 second to produce a complete run, on average. Another interesting point to note with regards to the length of time taken for each approach is that the *Cover* heuristic, despite creating each construction fairly quickly, takes much longer to descend to a local optimum and this again indicates that providing the local search with a good starting solution can reduce the time taken to find a local optimum as well as providing higher quality solutions. In the next section we take a closer look at the results produced using $Lastchance_a$ and apply the third, extended, local search neighbourhood with the aim of improving solution quality.

4.3.3 Applying the extended neighbourhood

Although Table 4.11 appears to show that $Lastchance_a$ gives very good results, there is a great deal of variation between the datasets and, in some cases, between different runs on a single dataset.

Figure 4.12 shows a breakdown of results for this approach by dataset, providing the number of optimal solutions from 10 and the number within 3 of the optimal for each of the 52 datasets. The value 3 has been chosen as a comparative measure because this indicates that only the most minor types of penalty have been incurred and thus, although the schedule may not be optimal, it would certainly be acceptable for hospital use. Furthermore, this allows a direct comparison with the results obtained by Aickelin (1999) and other approaches in the literature for which results are reported in the same way.

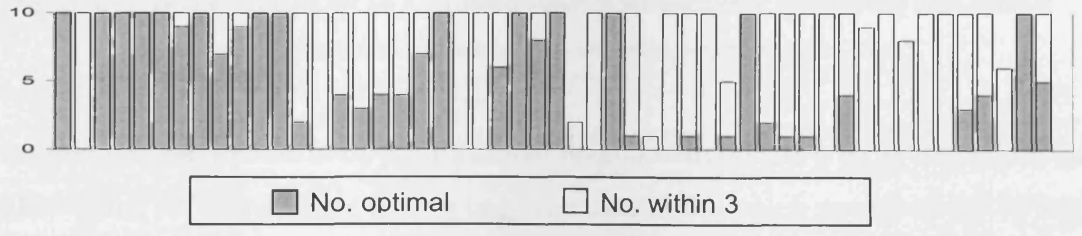
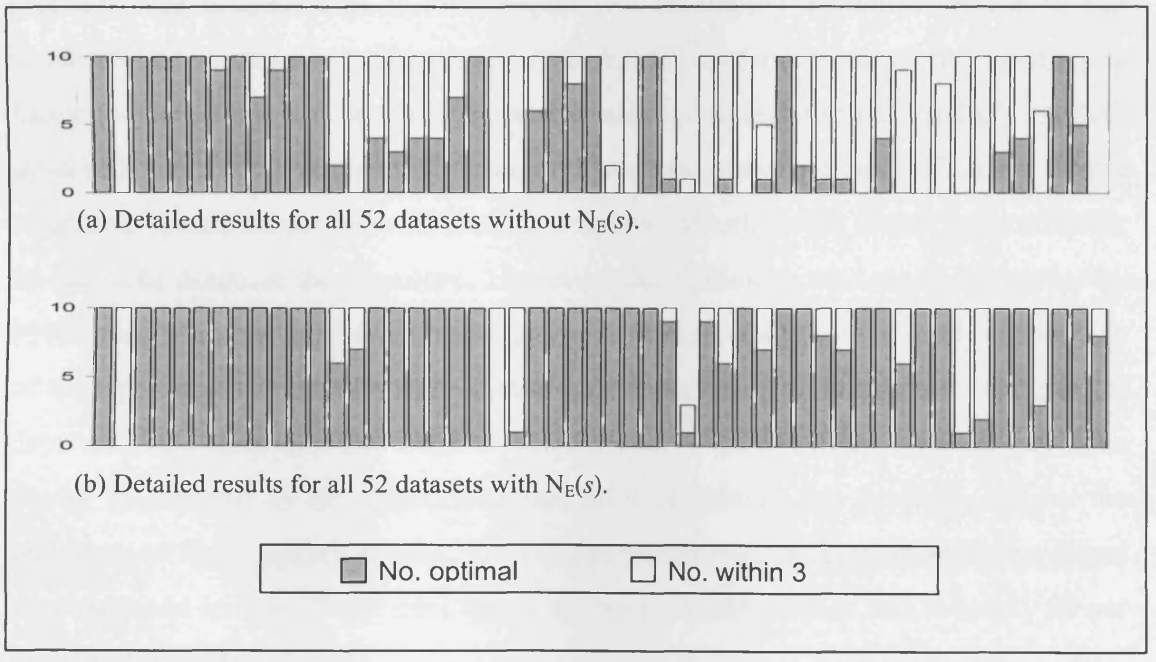


Figure 4.12. Detailed results for all 52 datasets showing the number of optimal and near-optimal solutions obtained using $Lastchance_a$ with the knapsack model.

From Figure 4.12 it is clear that while some datasets are solved to optimality in every run, there are several for which the optimal solution is never found and some for which the algorithm struggles to return even near-optimal solutions. At this point, the third, extended, local search neighbourhood $N_E(s)$ is implemented; due to the fact that this neighbourhood contains moves not in either of the other two neighbourhoods, it may introduce enough flexibility to allow solutions to escape from situations where any moves resulting in a lower preference cost would result in the solution becoming infeasible, since this neighbourhood allows, essentially, a chain of two moves in order to redistribute any undercovered shifts resulting from the initial shift change. As mentioned earlier, this neighbourhood is only applied once solutions are feasible. The resulting breakdown of solutions by dataset is shown in Figure 4.13 (b). The original results in Figure 4.10 are given in Figure 4.13 (a) for the purposes of comparison.



(a) Detailed results for all 52 datasets without $N_E(s)$.

(b) Detailed results for all 52 datasets with $N_E(s)$.

Figure 4.13. Detailed results for all 52 datasets showing the number of optimal and near-optimal solutions obtained with and without the extended neighbourhood, $N_E(s)$.

It is clear that the inclusion of the Extended neighbourhood has a large impact on the quality of the solutions; there is now only one dataset for which not all of the 10 runs produce near-optimal solutions and far more of the optimal solutions are found for each of the datasets, in general. In terms of computation time, the use of the Extended neighbourhood increases the time taken to perform the local search by about three

times, but this is still very quick in terms of actual time, taking around 2 seconds, on average, to complete one complete run. However, there are still a small number of datasets for which the algorithm struggles to produce optimal solutions and so at this point the algorithm, although generally producing good solutions and fairly robust across the datasets, is not yet performing as efficiently as the Tabu Search approach in Dowsland and Thompson (2000), which found optimal solutions to all datasets. In the next section we discuss the success of the algorithms produced so far with regards to the original aims of the chapter, as well as offering suggestions for improving solution quality still further.

4.3.4 Initial conclusions

Thus far, we have introduced a basic GRASP method for solving the nurse scheduling problem. The original aim of this chapter was to employ a simple version of this construct-and-improve metaheuristic in order to investigate how the conflicting factors of feasibility and optimality may be balanced in order to successfully produce good solutions. By investigating a range of heuristic constructions with some simple neighbourhoods, the problem of finding a feasible solution was found to be difficult, as has been noted in the literature. However, this difficulty was eased somewhat by hybridising the construction with the knapsack model, which solved a relaxed version of the problem in order to ensure that the total number of nursing shifts assigned to days and to nights did not become unbalanced. This day/night balance has been shown in the past to be a particular difficulty in solving this problem and by the inclusion of the knapsack model, the average percentage of feasible solutions found was increased to over 75 per cent for all the heuristic approaches and to nearly 90 per cent for the most successful.

Although these hybridised versions of the heuristics were capable of producing a large number of feasible solutions, they showed a great deal of variation in the quality of these solutions, with the most noteworthy finding being that the heuristic which did not include any information about the preference costs of the shift patterns in the construction phase produced solutions of a significantly worse quality than any other. Clearly, the local search part of the algorithm requires a reasonably good starting solution in order to be able to find high-quality local optima. Furthermore, once the

knapsack enables all heuristics to find a large number of feasible solutions, the usefulness of a construction focusing only on feasibility is undermined. Thus we can conclude that the balance between the influence that feasibility and optimality each have on determining the assignments during the construction is vital. Note that we cannot create the assignments based solely on the preference costs since the knapsack does not influence the actual feasibility of the schedule, but only ensures that the day/night balance is maintained.

Another interesting point which was investigated was whether or not using a pre-ordering of the nurses had an impact on solution quality. Although the difference is not as marked as for the decision to include the preference cost in the scores, comparing the *Combined* and *Holistic* values in Table 4.11 indicates that enforcing a fixed ordering of the nurses produces results of an inferior quality. The final type of construction investigated, using look-ahead to schedule potentially difficult nurses first and ensure they are given a good shift-pattern now if this may prove difficult at a later stage of construction, was shown to be the most successful of the four types evaluated. By further enhancing the local search, adding a more sophisticated neighbourhood to allow it to escape certain types of local optima, we have found good results.

Essentially, it has been shown that a straightforward construct and improve method such as GRASP is capable of balancing the conflicting objectives of this problem, once it has been hybridised with a simple exact method to ensure that the solution avoids certain regions of the solution space, which have proved difficult to escape from without employing very complicated local search neighbourhoods. The results in Figure 4.13 (b) show the algorithm presented here to be capable of producing a large number of optimal solutions for most datasets and near-optimal solutions in nearly all other cases.

However, as mentioned earlier, while these results improve on some other approaches, such as the GA results in Aickelin (1999), the algorithm is not as robust as the Dowsland and Thompson (2000) TS approach. Although the aim of this thesis is to explore balancing different constraints, rather than to find high-quality solutions for the problems investigated, it is interesting to determine whether these types of

techniques may be developed further into a robust algorithm rivalling the best so far. Nonetheless, for the GRASP approaches in this chapter and the ACO approaches introduced in the next chapter, finding high-quality solutions, although desirable, is a secondary consideration. After the initial investigations into the question of balance have taken place for both approaches, however, the GRASP approach is shown to be the more promising of the two with regards to creating a robust algorithm for this problem and so, in the next section, the most successful GRASP method is developed with the new aim of improving its performance further still.

4.3.5 Further experimentation

This section details the experiments and results carried out to further improve the results obtained using $Lastchance_a$ with the knapsack model by enhancing the local search and incorporating feedback. At the end of this section we will compare our final results with those produced using the TS method of Dowsland and Thompson (2000), the best performing algorithm applied to this problem so far, and show that our results compare favourably. We begin with a simple enhancement to the local search.

During the local search thus far, we have accepted only improving moves; we have found that by constructing a suitable starting solution we are able both to reduce the feasibility cost to zero and to produce a low preference cost without the uphill moves necessary to find optimal solutions in using the TS algorithm (Dowsland 1998, Dowsland and Thompson 2000). However, often a solution is surrounded by many neighbouring schedules of equal cost and we suggest that allowing some acceptance of these 'plateau' moves may allow a fuller exploration of the search space, leading to an overall improvement in final solution quality. Certainly, a non-optimal solution where all neighbouring moves are of equal or higher cost, may be afforded some assistance in escaping from such a region by allowing plateau moves; an equal-cost neighbouring schedule may have a lower-cost neighbour not reachable from the original solution by only a single local search move. Thus we investigate the inclusion of these plateau moves, noting that we must include new terminating criteria for the local search to prevent moves being accepted indefinitely.

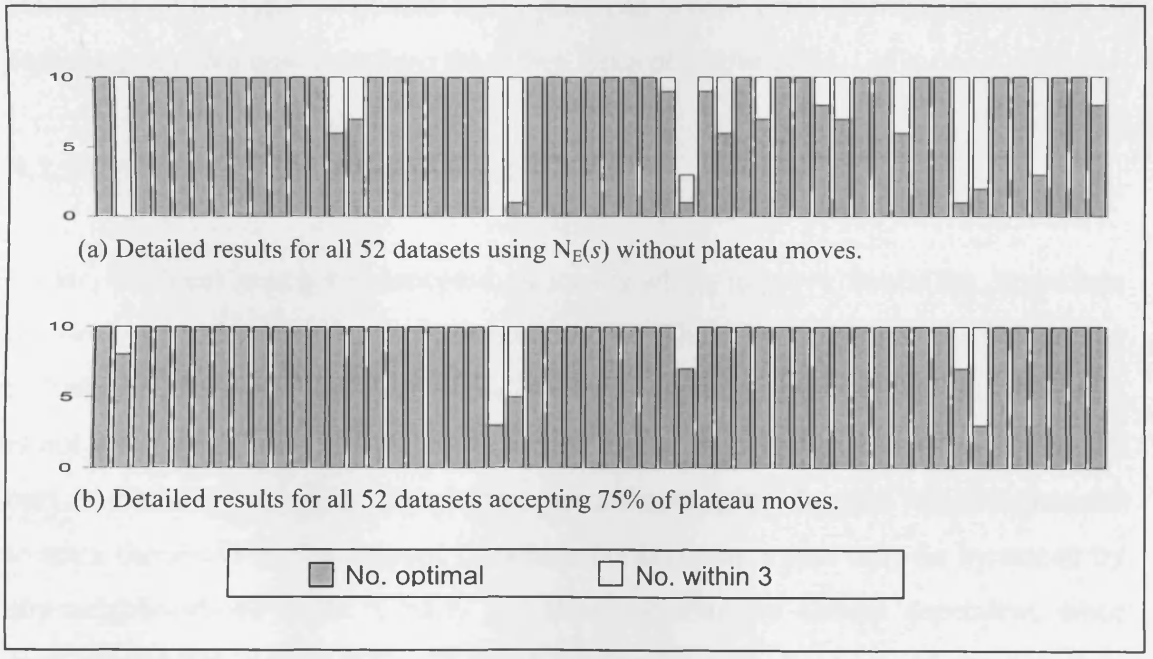
We are only interested in applying plateau moves once we have a feasible solution in order to fully explore the region around a good solution; accepting plateau moves before feasibility has been reached is likely to greatly increase computation times without adding much to solution quality. Once a feasible solution is found there are not likely to be plateau moves within the Change neighbourhood $N_C(s)$, since changing one nurse's pattern will be likely to increase the cover cost, and, since the Swap neighbourhood $N_S(s)$ is exchanged for the Extended neighbourhood $N_E(s)$ once feasibility is reached, it is only $N_E(s)$ for which plateau moves become useful. It is not necessarily sensible to include all plateau moves, since this may result in long periods spent in the same region of the search space and increase computation times more than is required. We aim to provide a balance such that the plateau moves become a tool which may be used to escape regions where solutions are surrounded by equal-cost solutions if necessary, but minimising wasted time spent exploring equal-cost areas of the search space when this is not the case. Thus we allow equal-cost moves to be accepted with a given probability. The probabilities tested were 0.25, 0.5, 0.75 and 1, and these were compared with the previous results obtained from effectively applying a probability of zero. Obviously by allowing the inclusion of plateau moves, the computation time required for the local search may be greatly increased and, depending on the stopping criteria may be allowed to continue indefinitely. The search is stopped after 10,000 moves have been accepted or all neighbourhoods have been exhausted, whichever comes first.

In order to compare the results obtained from using different probabilities of accepting plateau moves, for each dataset a value was calculated, indicating by how much the best solution of each run was sub-optimal, on average. Each approach could then be compared in four ways: summing these costs across all datasets, taking their maximum and mean values and counting the number of datasets for which no sub-optimal solutions were found from 10 runs. The results obtained from these experiments are shown in Table 4.14.

%	Sum	Max	Mean	Count/52
0	13.5	3.3	0.260	36
25	4.2	1.6	0.081	44
50	3.0	0.8	0.058	45
75	3.6	1.2	0.069	46
100	3.3	0.9	0.063	46

Table 4.14. The results obtained accepting different percentages of equal-cost moves.

Clearly, the introduction of these plateau moves significantly increases the number of datasets for which optimal solutions are found from all 10 runs, although it is difficult to choose between the different percentages as to which is the best performing. For comparison purposes, the results for each dataset using a 75% chance of accepting each plateau move are shown with those obtained using no plateau-move acceptances in Figure 4.15. The reason for choosing to depict the results with 75% chance of accepting plateau moves will become apparent in Section 4.3.6.



(a) Detailed results for all 52 datasets using $N_E(s)$ without plateau moves.

(b) Detailed results for all 52 datasets accepting 75% of plateau moves.

Figure 4.15. Detailed results for all 52 datasets with and without acceptances of plateau moves.

Comparing the results with and without plateau moves in Figure 4.15, the improvement in solution quality obtained with this addition to the algorithm is evident. There are now only six datasets for which optimality is not achieved in every run and all datasets produce the optimal solution at least 3 times out of 10. The computation time is increased, as expected, with the local search now taking

approximately nine times as long with the plateau moves as without. The total time to complete one run is now about 10 seconds, on average. Although this is longer than for the previous experiments, it is still very reasonable.

Thus far, we have used a basic GRASP approach in order to solve this nurse scheduling problem. We have carefully selected the heuristic and parameters which gave the best results and hybridised the GRASP with the knapsack model in order to ensure a good number of feasible solutions. Further, we have included a more flexible neighbourhood to allow greater exploration of the feasible region and have accepted a percentage of equal-cost plateau moves during the search in order to more thoroughly explore and prevent optimal solutions near the current solution from being overlooked. The results are suitably impressive and for nearly all datasets the optimal solution is found in all runs. However, six of the datasets are still difficult to optimise and so, in order to try to improve their success rate, we introduce two feedback elements into the GRASP so that later cycles can benefit from information attained in earlier cycles. We now introduce these two types of feedback.

4.3.5.1 Preference cost threshold

So far, the local search has accepted all moves which improve feasibility, regardless of the impact this has on the preference cost. Thus, by obtaining feasibility, the influence of the preference cost in the construction phase can be negated. However, it is not sensible to only accept moves which do not negatively impact the preference cost as this may prevent solutions from becoming feasible. Instead, we now propose to set a threshold on the amount by which the preference cost may be increased by any neighbourhood move. Clearly this threshold must be dataset dependent, since each dataset has a very different set of preference costs for each of the nurses; a suitable threshold for one dataset would be too low and therefore restrictive for a dataset incurring larger costs, but too high, and therefore of no consequence, for a dataset where all preference costs and the optimal value are much lower. In the initial cycles, then, the threshold will not be utilised and it will only be employed once a feasible solution has been found, after which it will be employed with the value $Best(c) + ThreshP$, where $Best(c)$ is the preference cost of the best solution found in cycles 1 to $c - 1$, c is the current cycle and $ThreshP$ is a dynamic value determined by

previous cycles. This means that we can iterate towards a value of *ThreshP* which is large enough not to hamper the realization of feasible solutions, but small enough so that solution quality is maintained as far as possible. *ThreshP* is initialised with a value $ThreshP = ThreshP_0$ and is increased each time a cycle c fails to find a feasible solution, using

$$ThreshP = ThreshP.(1 + \lambda .\text{Cover\_cost}(c)), \tag{4.21}$$

where Cover_cost(c) is the cover cost of cycle c, as given by (4.12), and λ is a constant. In the following experiments we set $ThreshP_0 = 10.0$ and $\lambda = 0.4$. Thus the threshold on the preference cost increase is initialised $ThreshP_0$ higher than the cost of the best feasible solution found so far and whenever a cycle returns an infeasible solution *ThreshP* is increased by an amount based both on the current value of *ThreshP* and on the amount by which the solution failed to be feasible.

The improvement phase of GRASP then accepts moves which improve the feasibility only if the preference cost is less than $Best(c) + ThreshP$. Obviously moves which improve the preference cost but not the cover are accepted in the usual way.

4.3.5.2 Knapsack-based diversification

The second type of feedback employed is designed to diversify the solutions obtained, ensuring that the cycles in each run visit different areas of the search space. Each solution can be categorised by the numbers of each type of nurse working days or nights and the use of the knapsack ensures that the solutions which are produced are balanced in this respect. However, for each dataset, there is usually more than one such balanced partition of the nurses into days and nights and it is often the case that some of these partitions will form the basis for the final solutions more often than others. In Section 4.2.2, we denoted the numbers of nurses of type t and grade g assigned to nights by y_{tg} and a complete solution at grade g comprising the set of all values y_{tg} by Y_g. Let Y denote the set of values Y_g at all grades g and thus represent a complete knapsack solution. In order to ensure diversity, we therefore wish to prevent any set Y from occurring too frequently. As for the preference cost threshold, the

initial solutions proceed unaffected, but once the number of times a set Y has been exploited reaches a certain threshold, *ThreshK*, further solutions pertaining to this day/night partition of the nurses are rejected for this cycle. This threshold is again calculated dynamically, using

$$ThreshK = \min(Freq(c)) + V(c)/3, \tag{4.22}$$

where $\min(Freq(c))$ is the minimum frequency with which any set Y occurs in cycles 1 to c, $V(c)$ is the number of different sets Y which have been employed in cycles 1 to c, and c is the current cycle, as before.

Note that the construction process is very fast and so as long as the number of rejected constructions is not extreme, as can happen when a set Y has a very small probability of being generated, it is practical to apply this type of feedback. In order to ensure this is the case and that computational expense is not wasted on low-probability Y, if no construction satisfying the acceptance conditions is sampled after 20 trials, then one of the solutions relating to the set Y with the minimum frequency of these 20 is accepted.

4.3.6 Further results

Both the preference cost threshold and the knapsack diversification feedback procedures were added to the best performing approach so far and 10 runs performed on each dataset. Since it was not clear from Table 4.14 which probability for the plateau move acceptances was the best-performing, we ran these experiments using probabilities of 0.25, 0.5, 0.75 and 1. The results from these experiments, providing the same details as for Table 4.14, are shown in Table 4.16.

%	Sum	Max	Mean	Count/52
25	1.8	1.6	0.035	50
50	1.1	0.8	0.021	48
75	0.5	0.4	0.010	50
100	2.2	2.0	0.042	50

Table 4.16. The results obtained accepting different percentages of equal-cost moves alongside the knapsack-based diversification and the preference cost threshold.

From Table 4.16 we can see that it is sensible to select 75 per cent for the plateau move acceptances and we show the breakdown of these results by dataset in Figure 4.17(d).

Figure 4.17 shows the gradual improvement of the algorithm's performance with the addition of each enhancement for comparison purposes. All graphs in Figure 4.17 show results using $Lastchance_a$ with the knapsack and with parameters $n = 3$, $w_c = 2$ and $w_p = 5$. Then, Figure 4.17 (a) shows results for each dataset without using the Extended neighbourhood, $N_E(s)$, (b) shows results with this neighbourhood included, (c) shows results with the inclusion of 75 per cent of the plateau moves and, finally, (d) shows the final improvements made using the two feedback mechanisms.

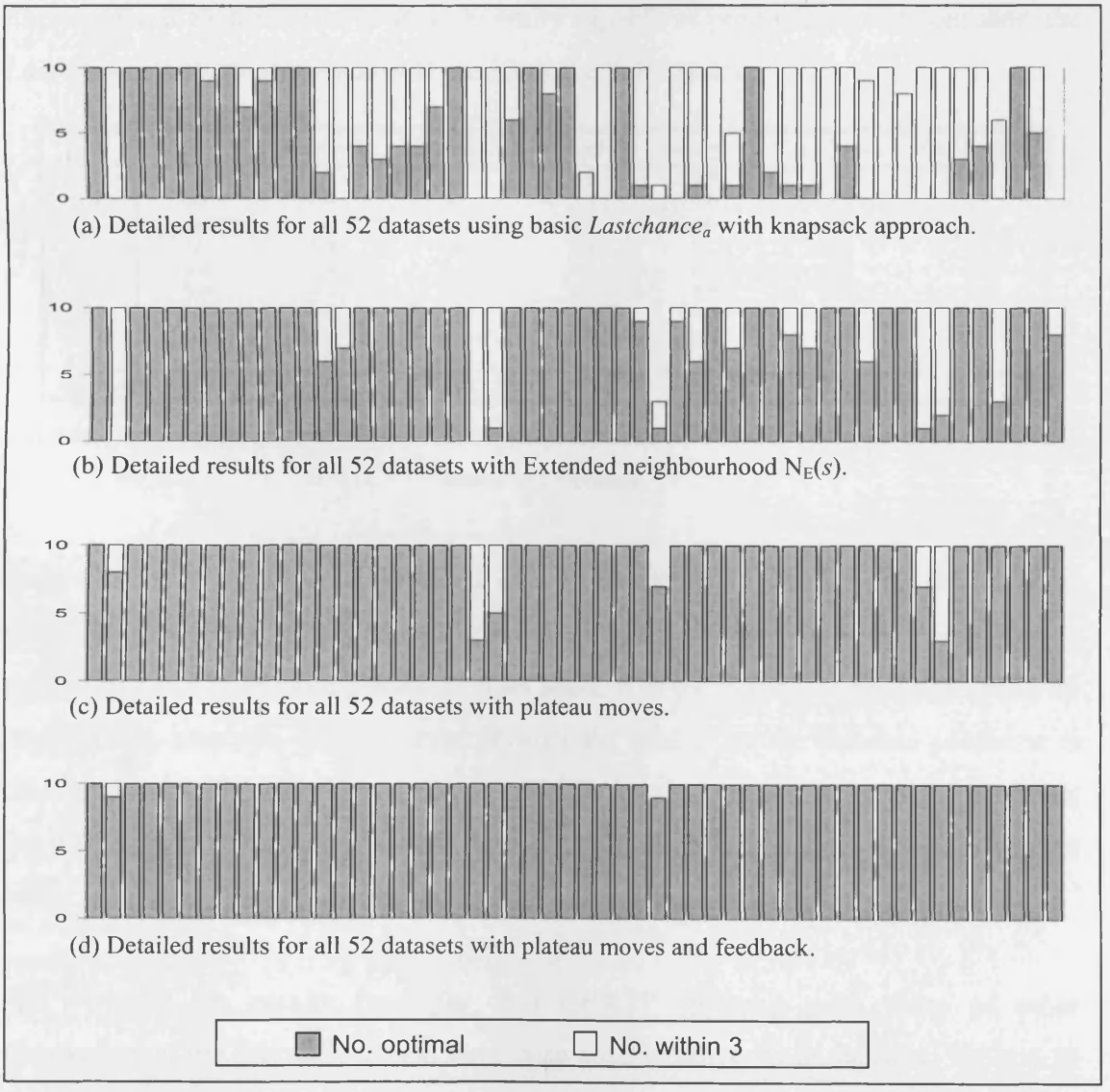


Figure 4.17. Detailed results for all 52 datasets using each of the different enhancements.

Thus Figure 4.17 clearly shows the improvement which each enhancement has afforded the basic GRASP approach and Figure 4.17 (d) shows that the final inclusion of the two feedback elements gives rise to an extremely robust approach, whose results are optimal in nearly all cases, rivalling those of the TS method in Dowsland and Thompson (2000).

In order to test how the earlier versions of the algorithm fare with these added enhancements we re-run all other heuristics with the knapsack, with the extended neighbourhood, plateau moves and feedback. Optimal solutions were produced in all 10 runs for all heuristics on 43 of the datasets, but results for the other nine datasets are shown in Table 4.18 along with the average cost above the optimal found in the 10 runs for each method. It can be seen that although each heuristic is clearly much improved and each method is now generally capable of producing good solutions, the *LastChance* additive heuristic still produces the best results.

Data	*Cover*	*Combined* $_a$	*Combined* $_q$	*Holistic* $_a$	*Holistic* $_q$	*Lastchance* $_a$	*Lastchance* $_q$
2	0.1	0.1	0.1	0.0	0.0	0.1	0.7
20	0.0	5.0	6.5	1.7	6.0	0.0	0.0
31	1.6	1.2	0.0	2.8	4.0	0.8	0.4
33	0.0	1.0	1.0	0.0	0.0	0.0	0.0
35	0.0	0.0	0.4	0.0	0.0	0.0	0.0
42	0.0	0.0	0.0	0.7	0.5	0.0	0.6
44	0.0	7.6	8.8	4.0	10.0	0.0	0.0
46	0.0	1.0	0.0	0.0	0.0	0.0	0.0
52	0.0	0.0	10.0	0.0	0.0	0.0	0.0
Average	0.19	1.77	2.98	1.02	2.28	0.10	0.19

Table 4.18. Results for all heuristics with the knapsack using the Extended neighbourhood, plateau moves and feedback.

A further set of runs were performed with no construction heuristic, where the local search and feedback enhancements acted on an initial solution for which each nurse's pattern was assigned on a purely random basis. It is interesting to note that when no construction heuristic is used, even though the quality of the feasible solutions is reasonable, the algorithm struggles to produce feasible solutions for many datasets; for five datasets infeasible solutions were returned as the best results found for several runs.

We compare the results from the best GRASP approach with those of other approaches in the literature which have been used to solve these datasets. Table 4.19 shows the best results obtained from GRASP as well as those found by Tabu Search

(Dowsland and Thompson 2000), Genetic Algorithm (GA) (Aickelin 1999) and Estimated Distribution Algorithm (Aickelin and Li 2005) approaches.

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
IP	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
GRASP	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
Tabu	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
GA	8	50	50	17	11	2	11	14	3	3	2	2	2	3	3	38	9	18	1	7	0	25	0	1	0	48
EDA	8	56	50	17	11	2	14	15	14	2	2	3	3	4	4	38	9	19	10	7	1	26	1	1	0	52

Dataset	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
IP	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
GRASP	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
Tabu	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
GA	3	63	15	35	65	41	10	39	37	32	5	13	5	7	54	41	22	20	3	4	3	4	28	108	74	58
EDA	28	65	109	38	159	43	11	41	46	45	7	25	8	8	55	41	23	24	6	7	3	5	30	109	171	67

Table 4.19. Comparison of best results for all 52 datasets using different approaches.

We can see from Table 4.19 that the GRASP outperforms the Estimated Distribution Algorithm and GA in terms of cost. Further comparison with the GA results in Aickelin (1999) show that the GRASP is also superior in terms of feasibility, since the GA did not consistently produce feasible solutions. Although the Tabu Search achieves optimal costs for all datasets, the GRASP algorithm has managed this without the use of complicated local search neighbourhoods and extensive use of problem specific information. Even without the use of feedback (Figure 4.17 (c)), the GRASP approach was able to find optimal solutions for all datasets for at least three of the ten runs.

4.4 Further testing of the GRASP algorithm.

Since the GRASP method presented here has been so successful in solving these 52 datasets, we present some discussion and analysis of the particular datasets as well as considering how the algorithm may be further tested. Appendix A gives the details of the 52 datasets supplied and on which all testing for this algorithm was performed. Originally, the intention was to select a small number of these datasets on which to experiment so that the remaining datasets could be used to test the robustness of the resulting algorithm. However, each dataset provides unique characteristics and so it was not possible to choose a small number of datasets which were representative of the entire base of problems. Although, as has been shown by the various results in the previous sections, some of the problem instances are more 'difficult' than others, there is usually a reason specific to that dataset why the algorithm struggled to find an optimal solution, such as a particular, anomalous nurse for whom a high-cost pattern

was required in order for solutions to be optimal or the optimal day/night partition of the nurses being particularly unlikely to emerge. Thus we were provided with 52 unique and interesting problem datasets and we tested our algorithm on all of them at each stage to see how successfully it was able to solve each of these cases.

In order to further test the algorithm, then, we artificially manufactured a further 52 datasets from the original data by redistributing the nurses into new groups and creating an artificial cover requirement for each, such that there was no slack for each problem. Nurse numbers in each new dataset were kept reasonable within each grade and such that the cumulative total night requirement at grades 1, 2 and 3 would stay at 7, 14 and 21 respectively as in all previous datasets and the day cover requirement at grades 1 and 2 would be 14 and 28 respectively, also. Thus only the total day requirement was altered between datasets, making each newly-formed dataset as realistic as possible. However, despite the care with which these new datasets were formed, the results obtained from 10 runs on each did not show as much variation as expected: each of the 52 manufactured datasets was easily solved to optimality in every run using the most enhanced GRASP approach. Since these datasets were carefully created using real nurses and tight covering constraints we can conclude that either the algorithm created is very robust, and clearly powerful enough to successfully solve new instances or that there is something else in the underlying structure of the original 52 datasets which makes them especially difficult to solve, in which case we are justified in testing our algorithms on all 52 datasets at each stage, rather than just a representative sample. In either case, we can conclude that the final GRASP approach is certainly a very successful method, producing optimal solutions over 99.5% of the time and it rivals the most successful method so far, that of Dowsland and Thompson's (2000) TS approach, while remaining relatively simple and without the use of complicated neighbourhoods.

4.5 Conclusions

As stated in the introduction in Chapter 1, the focus of this thesis is concerned with balancing different objectives using greedy, randomised constructive techniques within a metaheuristic framework. The nurse scheduling problem presents an excellent opportunity to investigate the balance between feasibility and optimality as

these are in direct conflict and it is often the case that the quality of solutions in terms of the preference cost must necessarily be compromised if the schedule is to be feasible. However, we have shown that it is essential not to focus entirely on either aspect and that optimal solutions arise when a sensible balance between these two costs is achieved.

However, the main feature of the nurse scheduling problem which enables this balance to be struck is the underlying feasibility structure consisting of the day/night balance among the nurses. Since we have introduced only simple neighbourhoods in the local search phase, the main obstacle to finding feasible solutions was ensuring that each construction was balanced in terms of this day/night distribution and since we were able to introduce an exact method, by hybridising with the knapsack model, to ensure all constructed solutions were balanced, this meant that enough consideration could be given to the preference cost to ensure a higher chance of optimality. Thus the introduction of the knapsack model was an essential part of the success of this algorithm. However, in order to fully exploit the structure of the problem, it was necessary to introduce further measures such as the Extended neighbourhood and plateau moves to prevent the local search from stagnating in near-optimal regions of the search space from which there were no improving moves.

The final enhancements, introducing feedback, provided further examples of developing improvements to the balance between feasibility and optimality. The preference cost threshold, for example ensured that as much advantage was given to the preference cost as possible, without jeopardising the likelihood of attaining feasibility. Every time the threshold proved to be biased too much in favour of the preference cost, the effect was reduced until the balance was restored. The effect of the knapsack in maintaining a favourable day/night allocation was the enabling factor in allowing this bias towards preference cost in the feedback mechanism. For datasets where the particular day/night partition required for optimality was less likely to be produced by the construction phase, the knapsack-based diversification again allowed the algorithm to explore different constructions sufficiently such that optimal solutions could be found using the same balance between the costs as in all other situations.

In terms of the aims of this thesis, then, we have succeeded in showing that the construction part of the heuristic provides an essential role and, although some element of local search is required to ensure high-quality solutions, it is the construction which initialises the local search with a suitable initial solution and enables it to improve to a high-quality local optimum. Furthermore, we have shown that a good balance may be achieved within the construction, such that the constructed solutions are reasonable in terms of both the cover and preference costs. The use of the knapsack model has also been shown to be an essential part of the algorithm, allowing the constructed solutions to be made feasible relatively easily by the application of local search; without the knapsack, the number of feasible solutions was very low, on average, and there were several datasets for which no feasible solutions were found at all.

Thus we have found a GRASP approach, hybridised with the two elements of an exact knapsack model and feedback, which was able to successfully solve the 52 instances of the nurse scheduling problem presented here to optimality and able to rival the best-known approach so far. However, neither of these approaches has been able to produce optimal solutions for all datasets in every run, and there is also room for improvement in terms of computational efficiency. One of the most common methods of enhancing GRASP is to use a path-relinking strategy and this is certainly a promising avenue for further research.

Given that the methods available for this problem are able to find optimal solutions for all datasets, another interesting subject for further research would be to experiment with slightly larger or more complex datasets. It would be interesting to see whether a GRASP approach would be able to find good solutions to a similar problem where the datasets involve substantially more nurses, for example, or a longer planning period. In Chapter 3, variations on the nurse scheduling problem were discussed and it would be interesting to develop the methods presented here to see whether they would be able to deal with the inclusion of additional constraints.

Chapter 5

Nurse scheduling with ACO

5.1 Introduction

The purpose of this chapter is to investigate similar questions using an Ant Colony Optimisation (ACO) approach as those explored using GRASP in the previous chapter, such as the required balance between the conflicting aspects of feasibility and optimality in the construction heuristic and the role of the knapsack in enhancing solution quality. However, in order to be able to carry out such an investigation, it is first necessary to find an ACO framework suitable for application to this problem. This means establishing suitable trail definitions and parameters as part of the research.

Given the similarity of the constructive ideas behind ACO and GRASP, the positive elements of the GRASP construction may be easily included in an ACO algorithm and the use of a population-based approach and the inclusion of memory within the pheromone trails is a natural next step to further improve solution quality.

An essential enhancement to the GRASP algorithm was the inclusion of the knapsack model, which provided a great deal of improvement to the quality of the constructions. This knapsack model can easily be included in an ACO approach due to the similarity of the solutions' constructions and we will show that its inclusion provides vital assistance with regard to finding feasible solutions. In order to further enhance the solutions using GRASP, it was necessary to use an element of feedback. The ACO's pheromone trails provide a built-in capability to learn from previous solutions and the population-based approach provides a neat framework for incorporating diversity.

In the literature review in Chapter 3 it was mentioned that ACO is most directly applicable to routing-type problems and that for other, non-permutation-type, problems finding a suitable trail definition is a particular challenge. For discrete routing problems, an 'adjacency' trail is easily defined using the original ant colony analogy, and the quality of a final solution can easily be judged using a single distance measure. For the nurse scheduling problem there is no obvious way to maximise the effectiveness of a feedback mechanism and two costs, for cover and preference, must both be taken into account. This will be covered later in the chapter.

Although, as has been discussed in the literature review in Chapter 3, there are many types of ACO approaches, we shall be concerned only with a simple Ant System (AS) approach. The aim of this research is to see whether or not an ACO approach is able to provide the necessary balance between the different constraints in order for good solutions to be found and to investigate which construction heuristics are most suited to provide this balance in conjunction with an ACO approach, as well as the necessity of incorporating the knapsack model within these constructions. If good solutions were produced, this would then provide the basis for a more detailed investigation using the available ACO approaches, so that the most suitable approach could be found. However, as shall be shown later in this chapter, the ACO approach presented here is not as successful as some other methods, even with an additional element of local search.

The rest of this chapter is organised as follows. In the next section, we describe how ACO may be applied to nurse scheduling, including a discussion of the different

possible trail definitions. This will be followed by an explanation of the experiments undertaken and their results.

We begin with a description of ACO applied to nurse scheduling.

5.2 Solution approach

This section is concerned with the various solution approaches with which we shall be solving the nurse scheduling problem. Since we aim to use an ACO approach, in particular Ant System (AS), this section will discuss the visibility and trail costs which will be utilised and the possible values for the many parameters which are present in an AS algorithm.

We begin with an overview of AS applied to nurse scheduling.

5.2.1 AS applied to nurse scheduling

Chapter 3 has already given an explanation of the AS algorithm; the aim here is to apply such an approach to nurse scheduling.

The basic algorithm may be described as follows. The population is initialised with a given number of ants, *nants*. Each ant proceeds to build a schedule in an iterative construction similar to that of GRASP. One difference between the GRASP construction and that of an ant is that at each stage of the construction, the ants may select from all available components, rather than just the best n. Another, more significant, difference between the construction by a single ant using an AS approach and that of a GRASP lies with the way the scores are calculated; each component has a score based partly on the visibility, which may involve utilising a heuristic such as those suggested in Chapter 4, and partly on the information contained within the trail matrix, τ. Thus a final score pertaining to the assignment of nurse i to shift pattern j would be calculated using

$$score(i,j) = \frac{t(i,j)^{w_t} v(i,j)^{w_v}}{\sum_{\gamma \in J} t(i,\gamma)^{w_t} v(i,\gamma)^{w_v}}, \tag{5.1}$$

where J is the set of all feasible shift patterns for nurse i, $t(i,j)$ is the score relating to the trail information, $v(i,j)$ is the visibility score and w_t and w_v are the weights corresponding to trail and visibility, respectively. Note that the notation used here is not standard as, in much of the literature, the trail and visibility weights are denoted by α and β, respectively, and the trail and visibility scores are given by $\tau(i,j)$ and $\eta(i,j)$, respectively. However, since throughout this thesis all weights are presented in the format suggested here, this notation is consistent with the rest of this thesis and also provides an indication of which aspect the weight relates to, when taken out of context of this formula. In terms of the trail and visibility scores, the trail score presented here indicates a score to be calculated in some way from the trail matrix elements, rather than indicating one such element of the trail matrix. For consistency, then, the notation for the visibility score is altered in line with this. A glossary of the notation for this chapter can be found in Appendix D.

As mentioned before, the visibility scores may be calculated using a heuristic and the visibility scores suggested here are taken from the GRASP approach in Chapter 4. The trail scores will be calculated from the information in the trail matrix, τ, at that time. In routing problems, such as the TSP, the elements of the trail matrix may be used directly as the trail score and for one of our suggested trail definitions, this will also be the case. However, for two of the three trail definitions which will be presented later in this section, the trail matrix cannot be used directly and a trail score must be calculated, from relevant values within it, first. Since the knapsack model proved so invaluable to the success of the GRASP constructions, it is likely to be a similarly necessary addition to the ACO. We include it in our experiments in the same way as for the GRASP approach in order to determine whether this extra bias towards feasibility is necessary when approaching the problem using ACO.

Once each ant in the population has built a complete solution, the elements in the trail matrix are updated. This takes the form of 'evaporating' pheromone from existing

trails by multiplying each element in the trail matrix by $(1-\rho)<1$ and by adding an amount to each element to reflect the quality of the schedules produced. For an ant a with solution cost C^a, the trail would be updated for all elements utilised within that schedule by an amount Q/C^a. So, where an ant has produced a low cost schedule, the trails representing this solution will become larger than for poorer solutions.

Let R be the set of nurses and R^+ be the set of nurses already allocated. Let *nants* be the total number of ants and $nants^+$ be the number of ants with a completed solution. Similarly, let *gen* be the number of cycles required and gen^+ the number of completed cycles. Then we may describe the process by which AS is used to schedule the nurses using the procedure detailed in Figure 5.1.

Procedure to solve the nurse scheduling problem using AS

- Set $gen^+ = 0$.
- Set $nants^+ = 0$.
- Set $R^+ = \varnothing$.
- Calculate the visibility score $v(i,j)$ associated with assigning pattern j to nurse i, for all feasible pairs (i,j) [§].
- Calculate trail score $t(i,j)$ associated with assigning pattern j to nurse i, for all feasible pairs (i,j) [§] using trail matrix τ.
- Calculate final score $f(i,j)$ for this pairing with $f(i,j) = \dfrac{t(i,j)^{w_t} v(i,j)^{w_v}}{\sum_{\forall \iota,\gamma} t(\iota,\gamma)^{w_t} v(\iota,\gamma)^{w_v}}$. [†]
- Select (i,j) using roulette wheel selection, i.e. each (i,j) is selected with a relative probability proportional to $f(i,j)$.
- Update the schedule with this allocation and set $R^+ = R^+ \cup \{i\}$.
- If $R^+ \neq R$ go to Step 4.
- Set $nants^+ = nants^+ + 1$.
- If $nants^+ < nants$, go to step 3.
- Update the trail matrix τ.
- Set $gen^+ = gen^+ + 1$
- If $gen^+ < gen$, go to Step 2.

[§] Note that the set of feasible allocations (i,j) may include all nurses not already allocated, or may relate to a single nurse i given by a predefined ordering.

[†] Here w_t is a weight associated with the trail score and w_v is a weight associated with the visibility score. These are constants which may be varied to assign more or less weight to each of the components.

Figure 5.1. Procedure to solve the nurse scheduling problem using an AS approach.

It is clear that there are several parameters associated with the AS metaheuristic, as has been mentioned earlier. Before we discuss the problem of defining a trail matrix or calculating visibility costs, we shall briefly examine each of these parameters and discuss their significance. These are: population size, *nants*; number of cycles, *gen*; evaporation rate ρ; the weights associated with the trail and visibility, w_t and w_v respectively; and the value of Q, the constant which initially determines the weight of chosen elements over those which have not been included in a solution.

5.2.2 Parameters

Population size

In a permutation-based problem such as the travelling salesman problem (TSP), the number of ants in the population, and therefore the number of solutions constructed in every cycle, is generally taken to be equal to the number of nodes in the original problem. In the case of the TSP this would be equal to the number of cities. This number gives a good spread of starting solutions, especially if each permutation is initialised with a different starting node. For example, in the TSP, each ant is generally assigned a different starting city. This gives balance within the ant population.

For the nurse scheduling problem, however, nurse-shift pattern pairs are being created, rather than a single permutation. It would be possible to initialise each ant with a different nurse, but, having seen when using the GRASP approach that a pre-ordering of the nurses was not very successful, it must be concluded that some nurses should be scheduled earlier in order for solutions to be feasible and that enforcing each ant to initialise with a different nurse would necessarily result in the inclusion of some less-desirable solutions.

A second option would be to initialise each ant with the same nurse, but each taking a different shift pattern. However, this again forces one nurse to be scheduled first, and given that some shift patterns are likely to be very high cost, this method is likely to be even less successful.

The only sensible option available is to give each ant the freedom to make the initial selection, but by choosing a sensible population size this should hopefully still give rise to a good range of solutions. It seems sensible to allow a population size similar to the number of choices each nurse has available so that each shift pattern could potentially be chosen once. Given that each nurse works a different number of shifts and has a different number of patterns available, there is no one value which will be equal to the number of shift patterns for every nurse, so we select a suitable compromise, which will be discussed in Section 5.3.1.

Clearly, with too large a population size, the computational expense required to produce solutions would be unreasonable. Further, the feedback will lack focus and it will be difficult for the trail matrix, and the solutions from each cycle, to converge.

Number of cycles

The number of cycles permitted must be large enough for the trail matrix to be useful in helping to find good solutions, but obviously there is no point in continuing the search once the trail has converged and the population contains little or no diversity. Given that we had success with a feedback function using GRASP over 100 cycles, we use this same figure as a suitable value for the number of ant colony cycles. Obviously, this may be modified in the future if necessary, but later experiments will show that 100 cycles is suitable.

Evaporation rate

The evaporation rate determines how lasting the effects of each trail update are; a low value of ρ means that each cycle feels the effects of the trail from previous cycles more strongly and the algorithm is likely to converge more quickly. A higher value for ρ, on the other hand, encourages diversity. This means that any undeserved success for a particular trail value is likely to be short-lived, but also that very good components with currently high trail values may easily be overlooked in the next cycle; getting a good balance is important. We therefore choose a value commonly used in the literature, discussed further in Section 5.3.1.

Visibility and trail weights

It is necessary to determine the weight which will be attributed to each of the components used to create the score function. Using GRASP we determined weights to assign to the cover cost and preference cost parts of the score. When using ACO, these weights are still in use, but the whole of this score must be weighted against a secondary cost determined by the trail matrix and it is important to strike the correct balance between the two.

We wish to ensure that the information from previous schedules is utilised to maximum effect. By assigning too little weight to the trail information it becomes redundant and past mistakes may be repeated. By assigning too much weight to the trail, however, there is the risk that schedule quality may decrease as decisions are based, mainly, not on what will improve the current partial schedule, but on which elements have been successful in previous schedules, regardless of the impact on the current one. It is very important, therefore, to ensure the correct weightings are applied.

In previous literature using ACO, it is generally the case that the weight for visibility is higher than the trail weight, which is usually set to 1 (Dorigo et al. 1991, Dorigo et al. 1996, Dorigo et al. 1999, Dorigo and Stützle 2004, Gutjahr and Rauner 2007). In general, therefore, the score function or heuristic calculating the visibility is adequate to give a good idea of which elements will be successful, but the trail matrix is used as an additional aid to choose between elements of similar cost or to steer solutions away from elements which have proven very poor and towards elements which have been particularly successful.

In the case of nurse scheduling, particularly, it is difficult to decide which elements may be better additions to a partial solution based only on previous success. For the TSP, for example, there is a linear relationship between each solution element and the final cost and so success of a particular ordered pair of cities in one solution is likely to be as successful in another. For the nurse scheduling problem, however, this is not the case. Since it is not easy to determine how the trail and visibility scores will interact, and so how these two may be combined successfully in order to create a good balance, we base the initial experiments on values commonly appearing in the

literature. Details of the specific values experimented with will be given in Section 5.3.1.

The value of the parameter Q

In ACO, the parameter Q is a constant and it is this value, divided by the cost or cost function of the final solution, which is added to successful elements in the trail matrix. According to Dorigo et al. (1996), this value is relatively unimportant with regards to final solution quality and so an arbitrary value of 1 or 100 is often assigned (Dorigo et al. 1991, Dorigo et al. 1999, for example). The only power the value of Q possesses is to determine how much importance is assigned to elements of the trail which have been initially selected for inclusion within a schedule, over those which have not. For example, a very large value of Q will lead all updated elements to have a much improved chance of being selected by the next cycle of ants, whereas a small value of Q will mean that despite some elements being updated, all will still have a roughly equal chance of being chosen by the next cycle. Obviously, as the number of cycles completed increases, the effect of the value of Q becomes negligible as the trail matrix converges and so it is only in the first stages that Q can have any impact upon solution quality.

However, the cost of solutions varies greatly between datasets and it is not sensible to carefully select a specific value of Q in order to produce a given effect upon solution quality. We too select an arbitrary value of $Q = 100$ with the reasoning that if the trail converges immediately we can lower this value if necessary. As shall be shown by later experimentation, however, the original supposition holds true and the impact of the value of the parameter Q appears negligible.

5.2.3 Calculating the cost of a schedule

Previously, using the GRASP approach in Chapter 4, there was no need to combine the cover and preference costs into a single score. Each schedule had a preference cost and was either feasible or infeasible. When applying local search a hierarchy was established for these two costs so that an improvement in cover would be classed as better than an improvement in preference cost, regardless of the actual figures involved. Using ACO, however, a single cost must be manufactured, demonstrating

the total success of a solution so that its components may be updated accordingly in the trail matrix.

There is the option to create a similar hierarchy, so that infeasible solutions are excluded from the feedback process, for example, or are judged negatively, however, given the difficulty of finding feasible solutions for this problem, the likelihood is that the majority of schedules will be infeasible, especially in the initial cycles. Thus by excluding or penalising components of infeasible schedules, it may be difficult for the trails to gain any useful information and so the use of a feedback mechanism becomes redundant.

Equally, scoring a schedule based on just the cover cost or just the preference cost is likely to result in extremely poor scores for whichever cost is ignored and alternating between the two costs would result in a lack of focus.

The most sensible way to proceed is therefore to attempt to create a single score for each schedule, incorporating both the cover and preference costs, weighting them so as to ensure that feasibility is given more consideration.

The formula used to calculate the score for a schedule created by ant a, T^a, is then

$$\frac{Q}{Wc^a + p^a + 1}, \tag{5.2}$$

where W is the weight demonstrating the importance of feasibility over preference cost, and c^a and p^a are the cover and preference costs of the schedule created by ant a, respectively.

Varying W will determine how much more importance is given to satisfying the cover constraints over the nurses' preferences. By setting the value of W suitably high, it is possible to ensure that the cover and preference cost hierarchy is maintained, with even small improvements to the cover cost valued more highly than any preference cost improvements. A lower value will obviously treat the two costs more equally.

Adding 1 to the denominator ensures that, for datasets where a zero-cost solution is possible, a division by zero can never occur. The initial value chosen for W will be discussed in Section 5.3.1.

Now that we have set up the basic AS algorithm as applied to nurse scheduling and have a means of calculating the relative success of each solution, we may discuss the particular trail definitions and ways of scoring the visibility which will give us the scores $t(i,j)$ and $v(i,j)$. We begin by discussing appropriate trail definitions.

5.2.4 Trail definitions

As has already been mentioned, the nurse scheduling problem has no obvious trail definition and so we must manufacture something suitable for the purposes required in order to make an ACO algorithm practical for application to this problem. Even once such a trail definition is set up, there is still the question of calculating how each schedule will feed back to the elements of the trail matrix from which it is constructed and, as has been discussed, the two conflicting costs relating to cover and preference must be combined to give a score representative of each schedule's quality.

Note that the option exists not to combine these costs and instead to utilise two separate trail matrices, one relating to feasibility and one to preference cost, but again these must be weighted within the scoring process to determine how much each of the two trail scores will contribute. Given that the two costs must therefore be weighted anyway, it seems much more sensible to keep to a single trail matrix, thus keeping the algorithm relatively simple and avoiding unnecessary computational expense.

There is no one obvious trail definition, and we present three possibilities for further study. Each of these feeds back a different type of information to be included in the shift patterns' scores.

The GRASP algorithm was improved by applying two types of feedback: information on how much the preference cost should be allowed to deteriorate for the sake of improving the cover cost and information regarding the day/night partition. The first of these is not really suitable for adaptation to an ACO trail as the feedback only alters

one value and this value for the preference cost threshold has no direct relation to the potential success of selecting a particular shift pattern during the construction. The second option may create a good trail definition, between each nurse and either 'days' or 'nights'. This way, the algorithm would automatically learn which type of shifts each nurse should work in order for good solutions to be found. However, although this may initially provide a means for obtaining feasible solutions and potentially perform a similar function as the knapsack model, the reason this type of feedback was introduced in Chapter 4 was in order to diversify in situations where the optimal solution lay in a particular day/night partition which was less likely to occur. Therefore, although this approach may be suitable for promoting the production of feasible solutions it is likely to be very bad at optimising solutions where these rarer day/night partitions of the nurses are required, intensifying the use of just one partition and having the opposite effect to the successful knapsack-based diversification introduced in Chapter 4.

We therefore consider new ways of incorporating feedback into the constructions; by using the more subtle ACO feedback we can discover whether or not this day/night information will be processed automatically and the correct mixture of days and nights found without explicit instruction.

The three trail definitions investigated are described in detail below.

5.2.4.1 Nurse-pattern trail

The first suggestion for a trail definition is to assign a trail value between each nurse and each potential shift pattern for that nurse. Since we intend to build the solution as before, by assigning each nurse with one of a pre-determined list of shift patterns, it is sensible to include this as an option for a trail definition. This way, while each shift pattern may be assigned a visibility score for a given nurse, that same pattern will also have a pre-defined trail value.

The trail matrix component $\tau(i,j)$ refers to the current trail value between nurse i and shift pattern j and hence the trail score $t(i,j)$ for assigning nurse i to pattern j would be

$$t(i,j)=\tau(i,j), \tag{5.3}$$

where $\tau(i,j)$ is the entry in the trail matrix relating to the assignment of nurse i to pattern j.

This is possibly the simplest way to define the trail and also simplifies calculation of the trail update.

Practically, the trail matrix can be set up as an $m\times 411$ matrix, since m is the number of nurses and there are 411 possible shift patterns in total. Infeasible elements would never be selected and in order to save computation time, only the feasible elements of the trail matrix would be evaporated at the end of a complete cycle.

Updating the trail matrix at the end of each cycle from the *nants* schedules created is also simple using this approach. An amount $\Delta\tau_{ij}^{a}$ is added to the trail between each nurse i and the shift pattern j allocated to that nurse in T^{a} for each ant a. The update therefore takes the form

$$\tau(i,j)=(1-\rho)\tau(i,j)+\sum_{a=1}^{nants}\Delta\tau_{ij}^{a}, \tag{5.4}$$

where $\Delta\tau_{ij}^{a}=\begin{cases}\dfrac{Q}{Wc^{a}+p^{a}+1} & \text{if nurse } i \text{ works pattern } j \text{ in } T^{a}\\ 0 & \text{otherwise}\end{cases}$.

One of the drawbacks of using the nurse-pattern trail is a problem mentioned previously: the cost of each trail element does not share a linear relationship with the final solution quality. A shift pattern which has previously been part of high-quality schedules and has a correspondingly high trail matrix value owes a large part of its success to the other nurse-pattern pairs present in the schedule. An individual pattern with a low preference cost, for example, may only be successful if the other choices made allow the cover to be satisfied and also have relatively low preference costs.

Thus the trail matrix may not always accurately recognise which schedule components are necessary to achieve a high solution quality and which have previously been included in a good solution, but may be easily replaced for the sake of feasibility. We try another tactic in the aim of relating the final costs involved more closely with the trail elements.

5.2.4.2 Nurse-shift trail

One of the oversights of the nurse-pattern trail is the way in which similar shift-patterns are dealt with. Usually, similar shift patterns will have a similar preference cost. Since the nurse-pattern trail only updates the trail relating to the particular shift pattern involved, it may be the case that a similar shift pattern is overlooked, even if this substitution would improve feasibility without increasing total preference cost. By regarding each separate shift independently, the nurse-shift trail aims to rectify this problem. Two similar patterns will automatically be regarded similarly by the trail matrix, since the information stored relates only to single shifts rather than the pattern as a whole. Therefore, two patterns differing only by one shift will be treated similarly, regardless of which actual pattern is selected.

The trail matrix element $\tau(i,k)$ refers to the current trail value between nurse i and shift k and trail score, $t(i,j)$, would be calculated as

$$t(i,j)=\frac{\sum_{k\in j}\tau(i,k)}{|j|}, \tag{5.5}$$

where $|j|$ denotes the number of shifts k worked in pattern j.

At the end of a completed cycle, the trail matrix is updated from the *nants* schedules by considering every shift which each nurse has worked. The update therefore takes the form

$$\tau(i,k)=(1-\rho)\tau(i,k)+\sum_{a=1}^{nants}\Delta\tau_{ik}^{a}, \tag{5.6}$$

where $\Delta\tau_{ik}^{a} = \begin{cases} \dfrac{Q}{Wc^{a} + p^{a} + 1} & \text{if nurse } i \text{ works shift } k \text{ in } T^{a} \\ 0 & \text{otherwise} \end{cases}$.

This method of calculating the trail score should give the algorithm more flexibility and prevent it from discarding potentially good shift patterns at an early stage. However, since each nurse will have several shifts in the trail updated for every schedule created and these values will support the selection of several related patterns for selection in the next cycle, it could also lead to a lack of focus.

5.2.4.3 Nurse-nurse trail

Given the difficulty of applying a feedback mechanism to this problem, a third trail definition is suggested, which is based more upon the nurse-ordering idea used in Aickelin's indirect GA (1999). Since good results have been found by scheduling the nurses with a simple heuristic once a good ordering has been ascertained, we have adopted this idea of ordering the nurses for use with an ant colony trail. Aickelin found that the nurse scheduling problem was a difficult one for a GA to cope with due to its complexity, but by using the GA to solve the much simpler problem of ordering the nurses and then using a heuristic to decode this ordering into a schedule, good results were obtained. In some cases, a nurse will only have one or two shift patterns which can be part of an optimal solution; if these nurses are not scheduled early, there is a risk that in creating a feasible schedule these nurses may be forced onto a high-cost shift pattern. Here we suggest using the trail definition to keep track of an ordering of the nurses, while the visibility alone determines which pattern each nurse should work.

The trail score, $t(i, j)$, is based solely on the nurse considered for selection, relative to the nurses which have not yet been selected. As each nurse is selected, we essentially create an ordered string of the nurses. The set of nurses which have not yet been scheduled is denoted by $R \setminus R^{+}$.

As the trail score does not use any pattern-specific information, we simply use the trail score $t(i)$ to calculate a score for scheduling nurse i next in the sequence, for each nurse not yet scheduled. Here the trail score $t(i)$ would be calculated as

$$t(i) = \frac{\sum_{i' \in R \setminus R^+} \tau(i,i')}{|R \setminus R^+|}. \tag{5.7}$$

To update the trail matrix at the end of each cycle, an amount $\Delta\tau_{il}^a$ is added to the trail between each ordered pair of nurses (i,i') where nurse i was scheduled before nurse i' in T^a. The updated trail therefore takes the form

$$\tau(i,i') = (1-\rho)\tau(i,i') + \sum_{a=1}^{nants} \Delta\tau_{ii'}^a, \tag{5.8}$$

where $\Delta\tau_{ii'}^a = \begin{cases} \dfrac{Q}{Wc^a + p^a + 1} & \text{if nurse } i \text{ was scheduled before nurse } i' \text{ in } T^a \\ 0 & \text{otherwise} \end{cases}$.

Since this approach does not use any information about particular shift patterns from previous solutions, but only creates an ordering of the nurses, it is similar to Aickelin's (1999) indirect GA. Aickelin selected the best shift pattern for each nurse, whereas this approach uses a roulette wheel selection to choose the nurses' patterns once the ordering has been decided. Aickelin's resultant schedules were of a high quality, but not able to rival the mostly optimal solutions obtained by Dowsland and Thompson (2000), and it is possible that this trail will produce results of a similar quality.

5.2.5 Visibility scores

As mentioned earlier, the visibility scores for an AS algorithm are roughly equivalent to the scores used in a GRASP construction. The heuristics implemented in the GRASP algorithm are therefore equally as applicable here and for the nurse-pattern and nurse-shift trails the *Cover*, *Combined*, *Holistic* and *LastChance* heuristics may be

easily integrated into the AS algorithm as they stand. The nurse-nurse trail, however, creates an ordering for the nurses and so is not compatible with the *Holistic* heuristic, which assumes no pre-ordering. The *Combined* and *Holistic* heuristics use the same approach, but where the *Combined* heuristic considers just one nurse, the *Holistic* approach considers all unscheduled nurses. Calculating the visibility score for each unscheduled nurse using the *Combined* heuristic is equivalent to the *Holistic* approach, since all unscheduled nurses will have their shift pattern scores enumerated during each construction phase. The *LastChance* heuristic is also compatible with a nurse-nurse trail, since it gives a score to each nurse, as does the trail. The three heuristics which are used in conjunction with the nurse-nurse trail are therefore *Cover*, *Combined* and *LastChance* only.

The three trail definitions and the visibility scores to be used in conjunction with each have now been defined. Given the difficulty of finding feasible solutions to this problem, however, it is likely that these basic approaches may struggle to obtain feasibility. We now discuss possible extensions to the algorithm which may help to overcome this problem.

5.2.6 Extensions to the AS construction method

As we have discussed, the basic framework laid down to solve the nurse scheduling problem using AS has been put in place, but the particular difficulty of finding feasible solutions to this problem may cause the algorithm to struggle in this respect. We have already suggested that the addition of the knapsack model may provide the constructions with vital assistance and the application of the knapsack model to an AS algorithm will be discussed in more detail later in this section.

The feedback mechanism of the nurse-pattern or nurse-shift trails is likely to work very well just for reducing the preference costs, since the cost of each solution element has a linear relationship with the total preference cost, but since the feasibility of each component within the solution is dependent on every other, the feedback mechanism may struggle to converge to good values. Although there may be some nurses for which a particular shift or shift pattern is essential in order for feasibility, this is not usually the case and so it is finding the correct combinations of elements,

rather than finding specific elements which is essential; this is also something the feedback mechanism may struggle with, since components which would be a vital addition to a current partial schedule, but have previously been part of a low-quality solution, would be overlooked. Even if there were specific assignments essential to the feasibility of solutions for a particular dataset, if these are not selected early, due to a relatively high preference cost, for example, the trail matrix entry for this choice would gradually be evaporated and its likelihood of being chosen would decrease further.

The AS with nurse-nurse trail is also likely to struggle with regards to finding feasible solutions. In this case, the actual construction of a solution has no information regarding the success of individual components in previous cycles; the only assistance is in the ordering of the nurses. For certain datasets, where some nurses have only one or two feasible shift patterns, ordering the nurses may provide valuable assistance. However, given that no information about which of their patterns should be assigned is stored, the algorithm becomes similar to that of a GRASP construction and so is likely to require added assistance if good-quality solutions are to be found.

As discussed earlier, the solution construction approach used by each ant bears a resemblance to the construction phase of a single GRASP cycle and the AS metaheuristic can be viewed almost as a population-based version of GRASP which incorporates learning, but no local search. Given the similarity between the GRASP construction phase and the ant constructions, in the first cycles of AS in particular, it is likely that similar results to the pre-descent GRASP constructions will be obtained using AS. During later cycles, the effect of the trail information is likely to have more impact, but especially in the first few cycles the difference in trail values for 'successful' and 'unsuccessful' components will not be very pronounced. Note that this can be enhanced by varying Q if necessary, but even with large Q the early cycles will not be as affected by the trail as later cycles; the first set of ants to construct a solution will not be affected by the trail matrix at all, since there have been no previous cycles and all trail values will be equal.

It is important for these early cycles to make good choices so that the trail matrix can learn effectively. Given that the solutions produced by the GRASP construction phase

were generally of a low quality, it seems necessary to enhance the ants' scheduling capabilities if the AS approach is to be successful. We therefore also suggest adding an element of local search as a possible way to enhance solution quality. The addition of the knapsack model, introduced earlier as a potentially successful enhancement, and the options for adding local search are now discussed in more detail.

5.2.6.1 Knapsack

The knapsack model was very successful in improving solution quality when incorporated into the GRASP construction and incorporating this same model into the AS approach is likely to be even more advantageous to solution quality than for the GRASP approach. As an AS algorithm will, at best, contain a limited amount of local search, the correct assignments must be made during the construction process. Without good-quality solutions to draw on, the trail matrix has no chance of recognizing good components and, as mentioned earlier, if necessary assignments are overlooked in the early stages, there is a strong chance that the trail matrix will ensure they are never selected. Thus, by utilising the knapsack model, the chances of making sensible choices, certainly in terms of feasibility, are improved from the very beginning and the trail matrix is more likely to converge to values encouraging good solutions.

The knapsack model can be implemented here in the same way as in Chapter 4. For each ant, the knapsack will produce an initial list of which types of nurses may work days and nights. The construction will then continue as normal, but before any assignment is confirmed, the allocation is checked with the knapsack solution. If the assignment and knapsack solution are in agreement, and this may mean producing a new knapsack solution, the assignment is made, otherwise it is discarded and a new assignment must be selected. Figure 4.4 in Chapter 4 is representative of the procedure by which each ant may create a schedule using the knapsack model.

5.2.6.2 Local search

The local search proved to be an essential part of the GRASP algorithm; pre-descent schedules created using the construction process alone were never optimal and the

difference in cost before and after the local search was implemented was often great. In the first cycle, the trail does not factor into assignment choices since all trail values are equal and in the first cycles its influence is still relatively minor, since there has not been time for a significant difference in the trail values to accrue. We have already discussed how the visibility scores are generally assigned more weight than the trail scores and this is in keeping with the theory that in the early cycles, the AS will produce similar solutions to that of a GRASP construction. Since these are unlikely to be locally optimal and are therefore of a relatively poor quality, compared with those given the benefit of a local search algorithm, it seems pertinent to discuss the possibility of including some element of local search within the AS algorithm.

An important consideration, however, is the computational expense required to produce these solutions. The local search phase of the GRASP algorithm accounts for a large part of the time required to create the solutions, often accounting for more than half of the computation time. Using the AS method, the computational expense is already likely to be larger than for GRASP, since rather than just completing one schedule per GRASP cycle, we are required to create schedules for a number of ants in each cycle. Unless the number of ants or the number of cycles is kept very small, there will be far more than 100 schedules being created in every run. If local search is incorporated into each of these, potentially more than doubling the computation time, it is likely that the computational effort required would be unreasonable.

In order to reduce computation time while still receiving the benefits of local search, a compromise is suggested: from each cycle, apply local search to just one of the completed schedules. It makes sense to select the best, or most promising, schedule for this purpose. This can be determined using the usual process of lowest feasibility cost with preference cost as a secondary consideration. By selecting an already good-quality schedule, it is likely that the local search will require fewer iterations in order to locally optimise, thus reducing the computational impact of the local search further (Li et al. 1994, Feo et al. 1991).

A final point to note is that if only one schedule receives the benefit of local search, especially if this was originally the best schedule of the cycle, the resulting schedule will be of a far better quality than schedules produced by the other ants of the cycle.

As such, when the process is complete and the trail matrix is updated, it makes sense to update only from this one ant. Otherwise, the benefits of having found a locally optimal schedule will be overshadowed by the large numbers of relatively poor-quality solutions also providing feedback. When experimenting with the inclusion of local search within the AS framework, all three neighbourhoods described in Chapter 4 shall be employed, since the Extended neighbourhood proved so beneficial. The precise details of the local search can be found in Section 5.3.5.

5.3 Experiments and results

These experiments are performed on the same 52 datasets as described in Chapter 4 and in Appendix A. All datasets have previously been solved using exact methods and so we are able to judge the success of the ACO approach for each dataset by a direct comparison.

Again, any solution will be judged firstly on its feasibility and schedules with zero cover cost will be judged according to how the preference cost compares with the optimal solution. Where the cover cost is non-zero, the schedule will be discarded, regardless of the preference cost. In the case of no feasible solutions being found for any method, two infeasible solutions could be compared based on their cover costs with the preference cost only considered in cases of equal cover cost.

5.3.1 Initial experiments

For the GRASP algorithm, the initial experiments were performed on a single GRASP run of 100 cycles. Since each cycle was independent, GRASP often having been referred to as a repetitive sampling technique (Resende 2001), this gave a good overview of the algorithm's performance; a further 100 cycles would be likely to give very similar results. For the AS algorithm, this is no longer the case; although each cycle produces a number of independent schedules in identical conditions, each cycle learns from the one before it and so the cycles cannot be examined individually.

This interaction is an important factor in the success of the AS metaheuristic and it is essential in these first experiments to determine not only which approach produces the

best results, but which, if any, are successfully able to utilise the feedback from the trail to gradually improve the overall quality of each successive cycle. Thus, we cannot consider the 100 cycles as a set of independent observations and even for the very basic experiments we will need to do several complete runs.

Given the large number of constructions which constitute one AS run, and the large number of such runs which must be performed initially, the number of repetitions performed must be reasonably small. We have chosen to repeat each run 5 times, since this should give a fair approximation of how the algorithm will perform in general. We now discuss the parameters which must be assigned values and the experiments which will be performed to test them.

Parameter testing

The AS algorithm relies on a wide range of parameters and some of these are more sensitive to change than others. We have already discussed in Section 5.2.2 the different parameters associated with the AS algorithm. Note that the particular visibility scores chosen, with the exception of the *Cover* heuristic, also require parameter tuning to balance the cover and preference cost weights, w_c and w_p, respectively. Although values for these were found using GRASP, they may not be suitable for this algorithm; it will no longer be only the best n from which the choice is made, and these scores will also be used in conjunction with the trail scores, so it cannot be assumed that the parameter combinations chosen for the GRASP algorithm would be the best in this case.

Thus, the parameters associated with this AS algorithm are as follows:

w_c	The weight associated with the cover part of the visibility score
w_p	The weight associated with the preference cost part of the visibility score
w_v	The weight associated with the visibility part of the score function
w_t	The weight associated with the trail part of the score function
nants	The number of ants per cycle
gen	The number of cycles per AS run

ρ The evaporation rate

Q The multiplier for scores added to trail values during the trail update

W The weight assigned to undercover when determining trail update scores $\Delta\tau_{ij}^{a}$

Each set of parameter combinations selected must be applied, five times, to each of 52 datasets, using three trail definitions, each with either five or seven visibility heuristics, and each of these with and without the knapsack routine. So for each parameter combination, there would be a minimum of 9,880 runs to complete. Clearly, an exhaustive examination of parameter combinations is impractical. We therefore carefully select sensible values for each parameter in an initial attempt to gauge which of the trail definitions is the most promising and to which of the visibility heuristics, with or without the knapsack, it is most suited.

However, at this stage, we aim just to provide a general overview of the methods suggested and the computational expense required necessitates minimising the number of runs performed. For most of the parameters, just one value is selected initially. Once it is determined which of the approaches are worth pursuing, parameter testing can be carried out without wasting computation time on tuning parameters for approaches which may never produce good solutions. Table 5.2 shows the values for each parameter during the initial experiments and a discussion of these choices will follow.

Parameter	Value(s) investigated
w_c	1, 3, 5
w_p	1
w_v	2
w_t	1
nants	50
gen	100
ρ	0.5
Q	100
W	10

Table 5.2. Values investigated for each parameter.

Cover and preference cost weights

For most of the parameters only one value has been selected, however, given the sensitivity of the heuristics in the past to the balance between cover and preference

cost, some variation will be included in these initial experiments. Due to the difficulty of finding feasible solutions and the fact that the AS will not have the assistance of a sophisticated local search algorithm, we restrict w_p to 1 and try three equal and higher values for w_c: 1, 3 and 5. We do not try all values from 1 to 5 because of the large amount of computational effort required; if the choice of these parameters proves to be of particular importance, further experimentation will enable closer inspection of the relationship between these parameters and the solution quality.

Visibility and trail weights

As mentioned previously, in much of the literature (Dorigo et al. 1991, Dorigo et al. 1996, Dorigo et al. 1999, Dorigo and Stützle 2004, Gutjahr and Rauner 2007) it has been found that ACO is more effective when visibility weight is at least that of the trail and when the trail weight is equal to 1. Thus we have selected values which are in keeping with this, but are not too exaggerated.

Number of ants per cycle

As discussed earlier the number of ants is usually based upon some suitable number relevant to the size of the problem, although it is difficult to determine upon which criterion this should be based for the nurse scheduling problem. It was decided that using a similar number of ants to the number of feasible shift patterns for each nurse was the most logical although, given that this number varies between nurses, one value would have to be chosen which would be acceptable for all. Common numbers of patterns are 56 and 70 for (5,4) and (4,3) nurses, although many types of nurse have far fewer options and some can have as many as 80. When nurses are prohibited from working either day or night shifts, their number of options will approximately halve, depending on their type.

We selected the value $nants = 50$ as a compromise. This figure is very similar to the number of patterns available to the full-time nurses, who often account for a large proportion of the staff on a given ward. We wanted to ensure that there would be enough diversity within each cycle to give a diverse range of solutions, but without adding unnecessary computational expense.

Number of cycles

The number of cycles is set at 100, equal to the number of GRASP cycles in a run. This should allow the trail matrix plenty of opportunity to aid the constructions and also gives an easy comparison with the GRASP. This number can be altered in later experiments if necessary, but it is a suitable number for initial testing and results will show later that there is no real need to alter this value.

Evaporation rate

In much of the literature (Dorigo et al. 1991, Dorigo et al. 1996, Costa and Hertz 1997, Dorigo et al. 1999, Dorigo and Stützle 2004), ρ is set to 0.5 although higher values may be found successful. We adopt the value 0.5 for these initial experiments.

Weight of the cover in judging solution cost

The value of W is difficult to determine. A very large cost will ensure that solution quality is judged primarily on the cover cost and the preference cost will only be considered when the cover cost is zero. However, although this is the way solution success is judged when determining which is the best schedule, for example, or to decide whether or not to accept a local search move, this is not necessarily a good idea for this purpose. We wish the trail matrix to reflect the quality of schedule components on both these counts and by allowing it to build up a picture only of the success of each element in terms of feasibility, there is the danger that the trail will tend to favour feasible, very high-cost solutions and it will be difficult to balance the two once feasibility has been attained and preference cost suddenly becomes important. For this reason, we decide to use a more reasonable value for W, so that although the trail is biased towards cover, larger differences in preference cost will be reflected in the trail. This way, the algorithm should be able to steer the search towards lower-cost feasible solutions slowly, rather than quickly finding feasibility at the expense of solution quality.

Weight of selected components, Q

As mentioned earlier, the parameter Q is generally found to have a negligible effect on solution quality. As such, we select $Q = 100$ for these experiments.

Initial trail values

The trail matrix is updated using information obtained from previous cycles and each element must be initialised with a non-zero value in the first cycle. These initial values, τ_0, should be of a suitable size relative to the amount by which they will be updated, although as the solution process progresses the value τ_0 becomes increasingly unimportant. For our problem, the cost of a solution obviously varies between datasets and whether or not the solution obtained is feasible will also impact on the amount fed back to the trail matrix. We choose $\tau_0 = 1.0$ for these experiments.

Thus we have just three sets of parameter combinations to test on each of the 38 solution approaches, one for each value of w_c, and each of these experiments will be repeated 5 times on each of the 52 datasets. The next section details the results obtained from these experiments.

5.3.2 Initial results

When examining the results of the nurse scheduling experiments using GRASP, both the feasibility and the preference costs of these feasible solutions were considered. However, for these initial AS approaches, the results show that for each run, comprising 5000 constructions in total, the percentage of these which were feasible was low enough to make analysing the feasible solutions' preference costs irrelevant; in some cases, not one feasible solution was produced during the run.

Thus, not only would it be difficult to compare the (feasible) preference costs fairly, given the low number of these available from each run, but since finding feasible solutions is crucial, we wish to ensure that any decisions based on these experiments will maximise the chances of producing feasible solutions in the future. If it were the case that no feasible solutions were obtained from any of the approaches, we could use the average or best cover costs produced in order to compare the algorithms. However, given that many do produce some feasible solutions, we shall use the percentage of solutions which are feasible as an initial measure of success.

Table 5.3 shows the breakdown of results according to which approach and which value for w_c was applied. In order to obtain these figures the mean percentage of feasible solutions found in each cycle for each dataset in each of the five runs was calculated. Thus, an approach which produced exactly 25 feasible solutions in each cycle from a possible 50, for all datasets and runs, would have an average feasibility of 50.00%. If these 25 feasible solutions were only produced for one dataset and all others yielded no feasible solutions in each of the five runs, this figure would be smaller by a factor of 52. Table 5.4 shows the number of datasets from 52 for which at least one feasible solution was found along with the average number of runs from 5 attempts for which at least one feasible solution was found. Thus 52/5.0 indicates that every run on each dataset produced feasible solutions, while 52/2.5 shows that for every dataset feasible solutions were found for around half of the runs, on average. Due to rounding, values of 52/5.0 arise in Table 5.4 where the number of feasible runs was slightly under the maximum. In order to distinguish these from methods producing feasible solutions in every run, the values indicating the maximum number of feasible runs are highlighted in bold.

Thus, Table 5.3 indicates the total number of feasible solutions found from 5000 attempts per run, whereas Table 5.4 gives more of an impression of the success of one run on any dataset in finding at least one feasible solution. Both of these measures are important in judging the success of each approach; it is important that a good number of feasible solutions are found in each run, so that the trail matrix may learn effectively, giving more chance of finding high quality solutions, but equally, we wish the solution methods to be robust and able to cope with all datasets rather than producing many feasible solutions for some and none for others. Note that the *Cover* heuristic does not balance the cover and preference costs within the score and so no weight for w_c is required.

	Without knapsack model								
Trail	Nurse-pattern			Nurse-shift			Nurse-nurse		
w_c	1	3	5	1	3	5	1	3	5
Cover	0.07	—	—	0.05	—	—	0.01	—	—
Combined$_a$	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Combined$_q$	1.11	2.50	1.84	0.00	0.00	0.09	0.00	0.00	0.00
Holistic$_a$	0.02	0.01	0.01	0.00	0.00	0.00	—	—	—
Holistic$_q$	2.15	4.51	12.21	0.03	1.11	5.04	—	—	—
LastChance$_a$	9.56	16.86	17.94	8.69	16.68	18.23	3.52	8.79	10.74
LastChance$_q$	5.12	12.23	20.23	3.77	10.63	17.92	5.61	8.54	12.19
	With knapsack model								
Trail	Nurse-pattern			Nurse-shift			Nurse-nurse		
w_c	1	3	5	1	3	5	1	3	5
Cover	0.17	—	—	0.07	—	—	0.02	—	—
Combined$_a$	0.09	0.06	0.05	0.00	0.00	0.00	0.00	0.00	0.00
Combined$_q$	9.14	16.58	23.93	0.00	0.26	2.73	0.00	0.00	0.00
Holistic$_a$	0.01	0.03	0.01	0.00	0.00	0.00	—	—	—
Holistic$_q$	8.89	20.16	31.15	0.03	1.42	9.79	—	—	—
LastChance$_a$	25.23	55.96	**62.77**	21.31	50.35	58.23	9.54	30.95	38.27
LastChance$_q$	12.36	24.95	46.00	9.80	22.37	42.64	6.35	19.18	42.91

Table 5.3. The percentage of feasible solutions produced by the different methods and with different values of w_c.

	Without knapsack model								
Trail	Nurse-pattern			Nurse-shift			Nurse-nurse		
w_c	1	3	5	1	3	5	1	3	5
Cover	6/0.3	—	—	6/0.3	—	—	3/0.2	—	—
Combined$_a$	3/0.1	2/0.1	3/0.1	0/0.0	0/0.0	0/0.0	2/0.1	2/0.1	2/0.1
Combined$_q$	2/0.1	7/0.4	11/0.6	0/0.0	7/0.4	13/0.7	0/0.0	0/0.0	0/0.0
Holistic$_a$	2/0.1	3/0.1	3/0.1	2/0.1	0/0.0	1/0.1	—	—	—
Holistic$_q$	11/0.7	31/1.9	44/3.5	7/0.4	34/2.4	45/4.0	—	—	—
LastChance$_a$	38/3.4	47/4.5	46/4.4	38/3.5	47/4.5	46/4.4	40/3.5	50/4.8	50/4.8
LastChance$_q$	17/1.5	35/3.1	44/4.0	18/1.6	37/3.3	47/4.2	41/3.8	49/4.5	50/4.6
	With knapsack model								
Trail	Nurse-pattern			Nurse-shift			Nurse-nurse		
w_c	1	3	5	1	3	5	1	3	5
Cover	44/2.4	—	—	31/1.8	—	—	16/0.6	—	—
Combined$_a$	25/0.9	20/0.7	27/0.9	1/0.1	2/0.1	2/0.1	1/0.0	1/0.0	1/0.0
Combined$_q$	18/1.2	40/2.9	49/4.3	9/0.4	49/3.9	51/4.9	1/0.0	1/0.0	1/0.0
Holistic$_a$	11/0.4	20/0.6	19/0.6	1/0.0	2/0.1	3/0.2	—	—	—
Holistic$_q$	29/2.1	47/3.9	52/4.9	13/0.6	50/4.4	**52/5.0**	—	—	—
LastChance$_a$	47/4.4	52/5.0	**52/5.0**	48/4.6	**52/5.0**	**52/5.0**	49/4.6	**52/5.0**	**52/5.0**
LastChance$_q$	27/2.4	48/4.5	50/4.8	28/2.6	50/4.6	50/4.8	21/2.0	50/4.6	50/4.8

Table 5.4. The number of datasets for which feasible solutions were found/the average number of runs for which feasible solutions were found for each approach, rounded to 1d.p.

There are several factors to consider when analysing these results. We must select the trail definition, heuristic, w_c value with the best performance and decide whether or not the knapsack model is a successful addition.

The value in bold in Table 5.3 highlights the particular approach which produces the highest average percentage of feasible solutions. This approach also produced feasible solutions for every dataset, as can be seen in Table 5.4. It is clear that of the three trails, the nurse-nurse trail is the least effective, although it produces reasonable results in conjunction with the *LastChance* heuristic, which produces good results for all three trails. The nurse-pattern trail produces the highest average percentage of feasible solutions in general and is the most robust of the three trails; it is clear that the addition of the knapsack is essential in all cases, vastly improving the feasibility of results, and that higher values of w_c are also imperative to the success of the algorithms.

Another interesting observation is that for the *Combined* and *Holistic* heuristics, the multiplicative version appears to be much more effective at producing feasible solutions; the reverse is true for the *LastChance* heuristic, although the difference is less marked. A possible explanation for the superiority of the multiplicative over the additive approaches is that the additive versions give a great deal of weight to the preference cost. Even with high values of w_c, the preference cost score for a pattern can be some magnitude higher than that of the cover, especially if it is a particularly low preference cost and several nurses have already been scheduled. This leads to assignments required for feasibility being based primarily on preference cost instead. The multiplicative version, however, uses the w_c value as an exponent, thus reinforcing the dominance of the cover score. The other important point to note is that any patterns which would not contribute to the feasibility at all are given a cost of zero using the multiplicative approach and so would not be considered.

We mentioned that for the *LastChance* heuristics, the multiplicative version was no longer superior and this is probably due to the different mechanism by which it functions; by assigning each chosen nurse with their best shift pattern, the likelihood of producing feasible solutions is increased. If good choices with regards to feasibility have been made earlier in the construction, as is especially likely when the knapsack

model is employed, then the remaining few nurses are likely to be put on the 'correct' patterns which will fill the gaps in cover. As was mentioned before, for the multiplicative versions, there are fewer choices since those not improving the cover will have zero cost. The additive version therefore gives a broader range of options for each nurse, even if some of these are less than ideal with respect to feasibility. However, it may be that the *LastChance* heuristic, which is already very suited to producing feasible solutions, is aided by this added diversity, which allows it to further explore the solution space and produce schedules which the multiplicative version would not be able to create.

Table 5.4 shows that there are six methods for which at least one feasible solution was found for every run. Since we know that the best preference cost from each will therefore be from a feasible solution, we may compare the average best preference costs from each approach fairly. This comparison is made in Table 5.5; optimal solutions have been subtracted in order to give an indication of how far above the optimal these costs are. Thus a value of zero in Table 5.5 would indicate that every run achieved the optimal value for that method.

	With knapsack model								
Trail	Nurse-pattern			Nurse-shift			Nurse-nurse		
w_c	1	3	5	1	3	5	1	3	5
$Holistic_q$	—	—	—	—	—	15.19	—	—	—
$Lastchance_a$	—	—	4.53	—	4.51	4.70	—	6.66	6.71

Table 5.5. The average best cost with optimal subtracted over each run of the 52 datasets for the six methods with the maximum feasibility performance as shown in Table 5.4.

From Table 5.5 we can see that the nurse-shift trail with *LastChance* additive heuristic and $w_c = 3$, gives the lowest average best preference cost. However, the nurse-pattern trail with *LastChance* additive and $w_c = 5$, was shown in Table 5.3 to be much more successful in terms of number of feasible solutions produced, while the difference between the two methods in terms of average best preference cost is negligible. Since the nurse-pattern trail was shown to be slightly more robust in general, and has been shown to be a suitable choice for further investigation, we therefore continue all investigations with this method in the aim of further improving solution quality. All

further results refer to experiments carried out with the nurse-pattern trail, with the *LastChance* additive heuristic and using $w_c = 5$.

5.3.3 Parameter experimentation

All experiments thus far have been performed using just one value for each parameter, shown in Table 5.2, except for w_c, which has now been set equal to 5. Each run takes just under a minute on average and so five runs on 52 datasets to evaluate a method takes several hours. It is therefore not computationally viable to evaluate every combination of parameters and instead each parameter is evaluated in turn.

Weight of the cover in judging solution cost

There is a distinction in Table 5.2 between the parameters associated with a generic AS algorithm and those which have been introduced specifically for AS applied to the nurse scheduling problem. It is sensible to first adequately equip the AS algorithm with the tools necessary to solve the problem, before going on to fine-tuning the AS parameters. Thus we have investigated the relationship between solution quality and the trail definitions, the use of the knapsack and the weights for the heuristics in the first stages of the investigation. The final parameter which has been introduced to the AS algorithm in order to adapt it to nurse scheduling is the weight with which the solution cover cost feeds back to the trail, W. In all previous experiments this had been set at 10, but we shall now vary W in the hope of improving solution quality and investigate values of $W = 5$, 20, 50 and 100 for comparison purposes.

Table 5.6 shows the results from these experiments. Note that the results for $W = 10$ have already been obtained in the previous investigation. The table shows the following results: the percentage of solutions created in each run which were feasible; the average preference cost of the best solution from each run, less the optimal cost; the average value of the cost functions from the best solution of each cycle; and the average number of solutions from each run which were optimal. Note that since the best solution from each run is feasible the average of the best preference costs from each run will be equal to the average cost function values of the best solutions for each run, less the average of the optimal solution costs.

W	% Feas.	Best Pref.	Av. Best CF	No. Opt.
5	62.18	4.43	31.67	16.89
10	62.77	4.53	31.60	16.21
20	63.92	4.53	31.70	16.55
50	65.61	4.61	31.76	17.13
100	66.85	4.49	31.88	17.56

Table 5.6. The percentage of feasible solutions per run, average best preference cost per run, average cost function for the best solution from each cycle and average number of optimal solutions found per run using different values of W.

Clearly, varying the parameter W does not create a great difference in the results produced. However, there are slight variations and we shall use these to choose suitable values of W with which to continue our investigation. The percentage of feasible solutions found, for example, does increase slightly with W. And although the average best preference cost and average best cost function values vary and increase slightly with W, higher values of W do also appear to produce a higher number of optimal solutions, on average. However, since the number of optimal solutions implies an average of around 17 optimal solutions from a possible 5000 for each run, the difference between each approach is negligible. Since higher values appear to produce slightly better results in terms of both feasibility and optimality we select $W =$ 100 to continue with and, mindful that a high value may be to the detriment of the preference cost in general, we therefore also select $W = 50$ as a compromise. Since none of the values for W appear to have a great deal of influence, however, either of these values should give respectable results.

Visibility and trail weights

So far we have been concerned with optimising the parameters which adapt the AS algorithm to the nurse scheduling problem. Now that these parameters have been selected, we may further improve the results obtained by ensuring the visibility and trail weights are suitable. As discussed earlier, attempting all parameter combinations would be too computationally expensive and so, given that it is usual for the visibility power to exceed that of the trail in order to achieve good results, we attempt to vary only the visibility power, whilst keeping the trail power at a value of 1. We have

already tried $w_v = 2$ and now investigate values of $w_v = 1, 3, 4, 5$, including $w_v = 1$ for completeness.

Table 5.7 shows the results of this investigation.

W	w_v	% Feas.	Best Pref.	Av. Best CF	No. Opt.
50	1	64.44	3.74	29.52	22.55
50	2	65.61	4.61	31.76	17.13
50	3	65.88	5.48	33.24	12.64
50	4	65.30	6.03	34.10	10.69
50	5	65.09	6.54	34.57	9.68
100	1	65.61	3.81	29.51	23.62
100	2	66.85	4.49	31.88	17.56
100	3	67.13	5.76	34.16	12.83
100	4	67.06	6.00	35.06	11.12
100	5	66.20	6.53	35.56	9.85

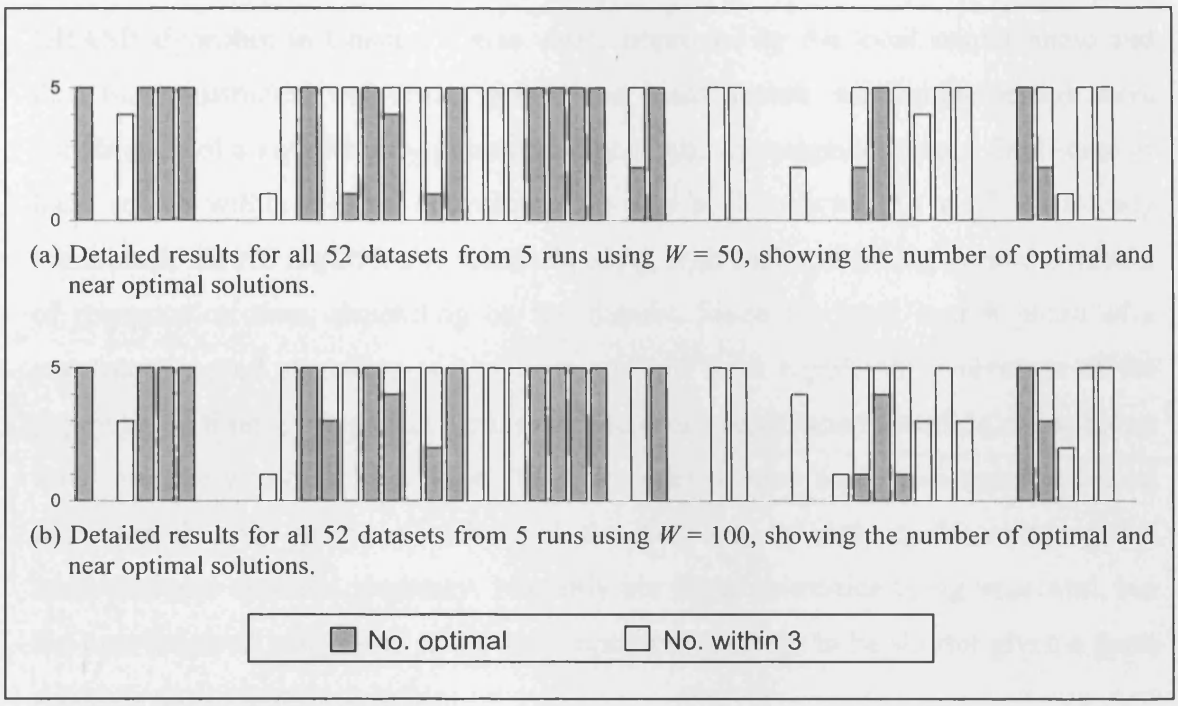
Table 5.7. Results for different values of w_v using $W = 50$ and 100.

From Table 5.7 it is possible to see the difference in solution quality between results obtained using the five trialled values of w_v with the two choices for W. In terms of feasibility, all w_v values perform similarly, although $w_v = 3$ gives a slightly higher return of feasible results for both W values; it is also clear that $W = 100$ has a slight advantage in percentage of feasible solutions obtained, as was originally indicated by the results in Table 5.6. This difference between the two W values therefore has more of an effect on the feasibility than varying w_v, as would be expected, since the two are directly related. In terms of the quality of the arising solutions, however, the value of w_v has a much greater impact and it is clear that for either W value, $w_v = 1$ is certainly the best performing of those tested. Given that the difference in feasibility between $w_v = 1$ and $w_v = 3$ is insubstantial in comparison to the cost benefits, we may accept $w_v = 1$ as the superior value and use this in the next experiments.

5.3.4 Results by individual dataset

Thus far the interpretation of results has treated the 52 datasets collectively. However it is unlikely that the figures pertaining to the group as a whole will be reflected equally in the results for each dataset and, as for the GRASP approach, the algorithm

will perform better on some datasets than others. Thus we present a breakdown of the results obtained by dataset in Figure 5.8.



(a) Detailed results for all 52 datasets from 5 runs using $W = 50$, showing the number of optimal and near optimal solutions.

(b) Detailed results for all 52 datasets from 5 runs using $W = 100$, showing the number of optimal and near optimal solutions.

Figure 5.8. Detailed results for all 52 datasets from 5 runs using $w_v = 1$ for $W = 50$ and $W = 100$.

Figure 5.8 shows the results, by dataset, for $w_v = 1$ for both $W = 50$ and $W = 100$. Clearly, the difference in solution quality between the two choices of W is small and, although $W = 100$ shows a slight advantage, neither one significantly outperforms the other. Although these two approaches are the best of those using AS, their results are still relatively poor quality. All runs successfully find at least one feasible solution, but, for around a quarter of the datasets, no optimal or near optimal solutions are produced from the 5 runs.

Since these results are vastly inferior in comparison to those produced using the GRASP approach in Chapter 4, we add a final stage of local search to the algorithm, as suggested in 5.2.6.2, in order to increase solution quality and make the two approaches more comparable.

5.3.5 AS with local search

As was mentioned in Section 5.2.6.2, the quality of the solutions produced using the GRASP algorithm in Chapter 4 was vastly improved by the local search phase and that the constructed solutions, before the local search was implemented were consistently of a significantly poorer quality. Thus, we suggested that a final stage of local search within the AS algorithm may also be beneficial. As we have already mentioned, the AS algorithm is relatively slow, with each run taking around a minute of computation time, depending on the dataset. Since the local search phase of a construct/descend algorithm is known to account for a significant percentage of the computation time, by applying local search to every constructed schedule, the solution times are likely to double, at least. Thus, in order to save time, we suggest applying local search only to the schedule of the best ant in each cycle, reducing the computational expense necessary. Not only are fewer schedules being improved, but the time required to descend to the local optimum is likely to be shorter given a good schedule as the starting solution.

A complication with this approach is that during the trail update, if all the schedules of all ants are considered, the superior attributes of the one locally optimal schedule are likely to be diluted by the inclusion of the other, much poorer quality, schedules so that any benefit to the trail from this solution is lost. Instead, therefore, only the solution to which local search has been applied will contribute to the trail update. This is likely to mean that the trail matrix will converge more quickly, but the aim is that it will quickly converge to much better values as it is being guided only by solutions of a much higher quality. Further, by applying local search to a completed schedule, this goes some way to overcoming the problem that the patterns are not linear with respect to feasibility cost since unsuitable patterns assigned on the basis of trail value may be replaced by the local search.

In the case of the GRASP approach in Chapter 4, the local search was an essential part of the solution process and an integral part of the algorithm. We therefore spent a great deal of attention selecting suitable neighbourhoods and ensuring that these were as effective as possible. When adapting this local search for use with the AS algorithm, we may apply the knowledge gained from the GRASP approach; the

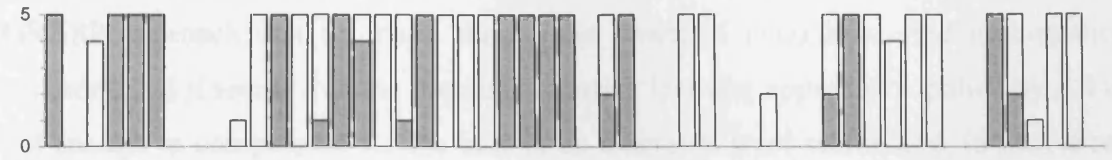
particular neighbourhoods used for GRASP may also be applied in this case. However, we wish to keep the local search as simple as possible; the aim is to investigate whether an AS algorithm, with the additional help supplied by a small amount of local search, is capable of producing good results for this problem, rather than creating an AS-local search hybrid. One of the ways in which the local search is kept simple is by the fact that only one solution per cycle will have local search applied. However, it is important to note that in the case of the non-population-based method, GRASP, only one solution is created in each cycle and so the two methods produce the same number of locally optimised solutions in each run. The three neighbourhoods applied in Chapter 4 will all be applied here; the extended neighbourhood was particularly effective. However, for the GRASP algorithm, a number of plateau moves were included to allow a more extensive exploration of the solution space. For the AS approach, we shall keep to the basic neighbourhood definitions, and allow no more than 10,000 local search iterations.

A final point to note is that since the trail matrix is updated only from the best solution, the parameter W becomes largely irrelevant, since the best solution is always likely to be feasible. Indeed, the results produced using $W = 50$ and $W = 100$ were identical in all cases and so only one set of results incorporating local search (LS) are given, shown in Table 5.9.

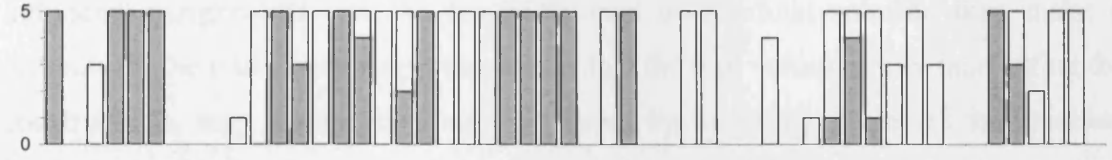
LS	W	% Feas.	Best Pref.	Av. Best CF	No. Opt.
No	50	64.44	3.74	29.52	22.55
No	100	65.61	3.81	29.51	23.62
Yes	50	67.07	3.07	26.63	262.23

Table 5.9. Results obtained using AS with local search compared with those previously attained.

Clearly, the results produced using local search are vastly superior in terms of the number of optimal solutions found, although the other differences are less significant. Figure 5.10 shows the breakdown by dataset of this approach and compares these results with those obtained without the use of local search using $W = 50$ and $W = 100$ in Figure 5.8, as well as the most comparable GRASP results, utilising the *LastChance* heuristic with the knapsack model and the three neighbourhoods in their simplest forms. Note that the GRASP implementation shows the results from 10 runs, rather than 5.

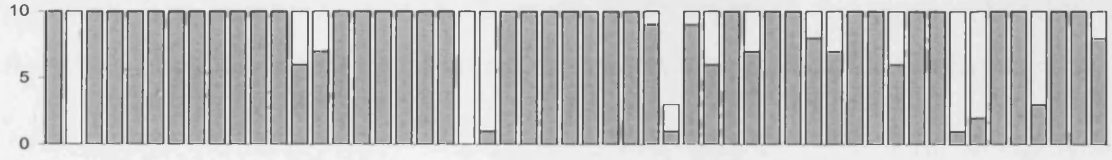


(a) Detailed results for all 52 datasets from 5 runs using $W = 50$, showing the number of optimal and near optimal solutions.



(b) Detailed results for all 52 datasets from 5 runs using $W = 100$, showing the number of optimal and near optimal solutions.

(c) Detailed results for all 52 datasets from 5 runs incorporating local search, showing the number of optimal and near optimal solutions.



(d) Detailed results for all 52 datasets from 10 runs using GRASP with the *LastChance* heuristic, knapsack and the three standard neighbourhoods.

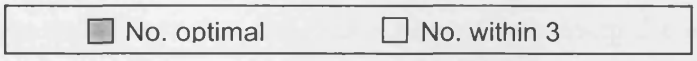


Figure 5.10. Detailed results for all 52 datasets from 5 ACO runs with and without local search, compared with 10 runs obtained from a basic GRASP using all three neighbourhoods.

Table 5.9 showed that, with local search, the algorithm was the most successful of the AS approaches so far. However, the breakdown by dataset in Figure 5.10 shows that although this approach finds high quality solutions for a slightly wider range of datasets, there are still several datasets for which no near optimal solutions are found and the algorithm is certainly not robust enough to cope with the wide range of datasets available. The GRASP approach, whose solutions are given in Figure 5.10 (d), is very similar to the AS approach relating to (c); both use the *LastChance* heuristic as the basis for construction and both apply all three neighbourhood definitions to a total of 100 solutions. However, it is clear that the GRASP outperforms the AS method to such an extent so as to call into question the suitability

of an AS algorithm for this problem. We have already shown, in Chapter 4, that a GRASP approach can be made robust and powerful enough to find high-quality solutions, but it seems that the population-based, learning approach supplied by AS is not enough to compensate for the lack of an extensive local search and, in fact, must even be hindering the process. As we have suggested, the lack of linearity between the individual assignments and the feasibility cost of the final solution may make it difficult for the trail to converge; this means that the trail values, rather than aiding the constructions, may instead be hindering them, by including irrelevant information. Furthermore, the roulette wheel selection process used by the ACO constructions includes all possible solution components, whereas the GRASP selects only from the best n and it was shown in Chapter 4 that the GRASP constructions produced the best results for small values of n. Note that, although ACO approaches do not generally employ a restricted candidate list (RCL), the Ant Colony System approach introduced by Dorigo and Gambardella (1996) shows the benefits of using an RCL with an ACO approach. The inclusion of an RCL in an ACO approach for nurse scheduling is therefore something which could be put forward as a suggestion for further research.

Figure 5.11 shows a plot of the average and best values of the cost function for each cycle, without local search and with $W = 100$, for dataset 52. This shape is typical among all datasets and shows that the trail is not really helping the ants to learn from previous cycles, although there is some evidence of improvement in the first few cycles. This is typical of the datasets for which no good solutions were found. The average value of the cost function is lowered after the first one or two cycles, but the quality of the best solution is largely unaffected by the number of previous cycles and the best cycle of the run is just as likely to be produced in earlier cycles as later ones. For datasets which do find optimal solutions, such as dataset 4, it is possible to see slightly more benefit from the trail in the first few cycles, but again, it is clear that the trail is unable to successfully guide the constructions.

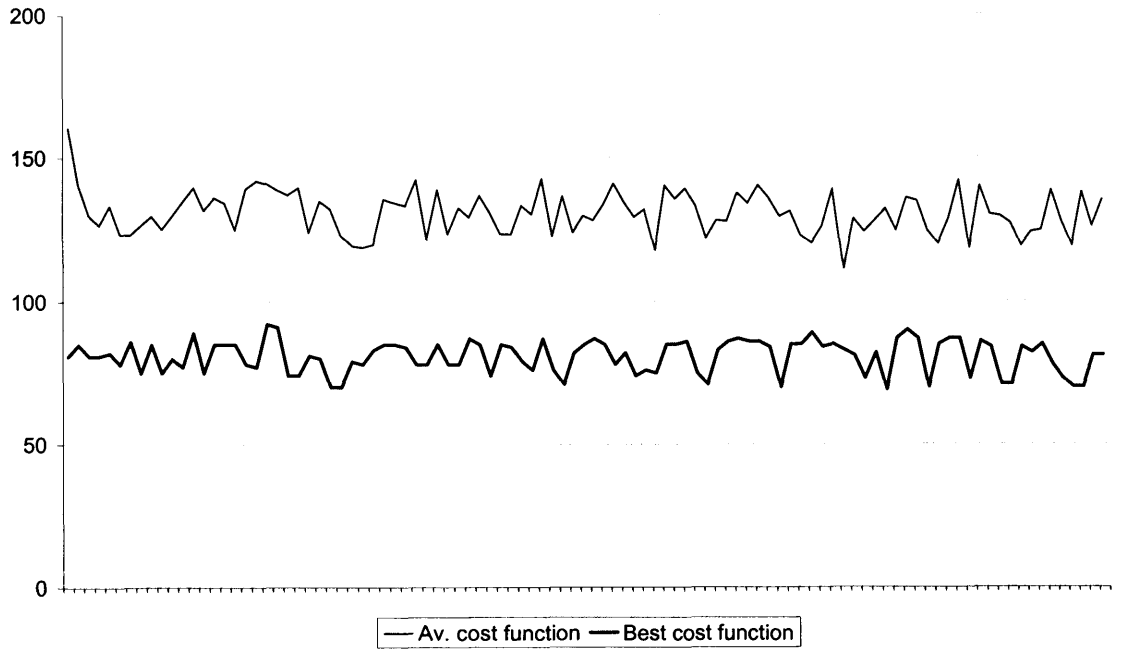


Figure 5.11. Plot of average and best cost function values against AS cycle for dataset 52, without local search and with $W = 100$.

Figure 5.12 shows a similar plot for one run of dataset 52 with the addition of local search.

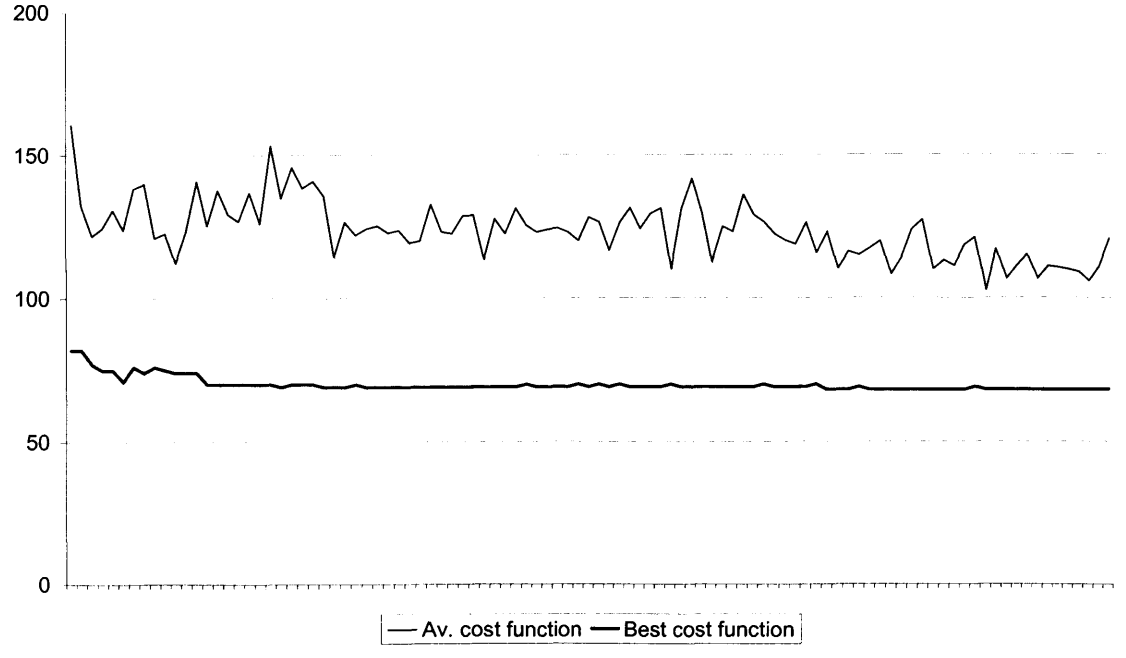


Figure 5.12. Plot of average and best cost function values against AS cycle for dataset 52, using local search.

It seems that with only one ant contributing, the trail is more able to focus and it can be seen that there is an overall tendency for gradual improvement as the number of cycles increases. This could potentially imply that the number of ants, *nants*, may have impacted negatively on the solution quality before the local search was applied and that, in particular, fewer ants may have been more successful. Indeed the *LastChance* heuristic selects from a number of nurses each time and so it is possible that setting *nants* equal to the number of nurses may produce better results. The *Combined* heuristic, for which the number *nants* = 50 represents a similar number to the number of assignments available in the form of the number of shift patterns per nurse, does show the trail playing more of a successful role in improving solution quality from cycle to cycle, as can be seen in Figure 5.13, even though the actual costs involved are substantially worse. Thus the nurse-pattern trail is capable of providing assistance to the constructions and an optimised value for *nants* may help the algorithm. Note that with local search, however, only one ant contributes to the trail and so solution quality can only increase with *nants*. However, with the results presented here, and given that even with the benefit of a good local search the algorithm is unable to compete with the GRASP approach, it is unlikely that merely changing *nants* will have enough of an impact on solution quality to make further investigation worthwhile. Another possible enhancement would be to use local search and apply local search to the best few ants' schedules, updating the trail from all of these, but, again, the poor quality of the solutions produced by AS thus far and the computational expense required to include more local search suggests little advantage would be gained by pursuing these ideas further.

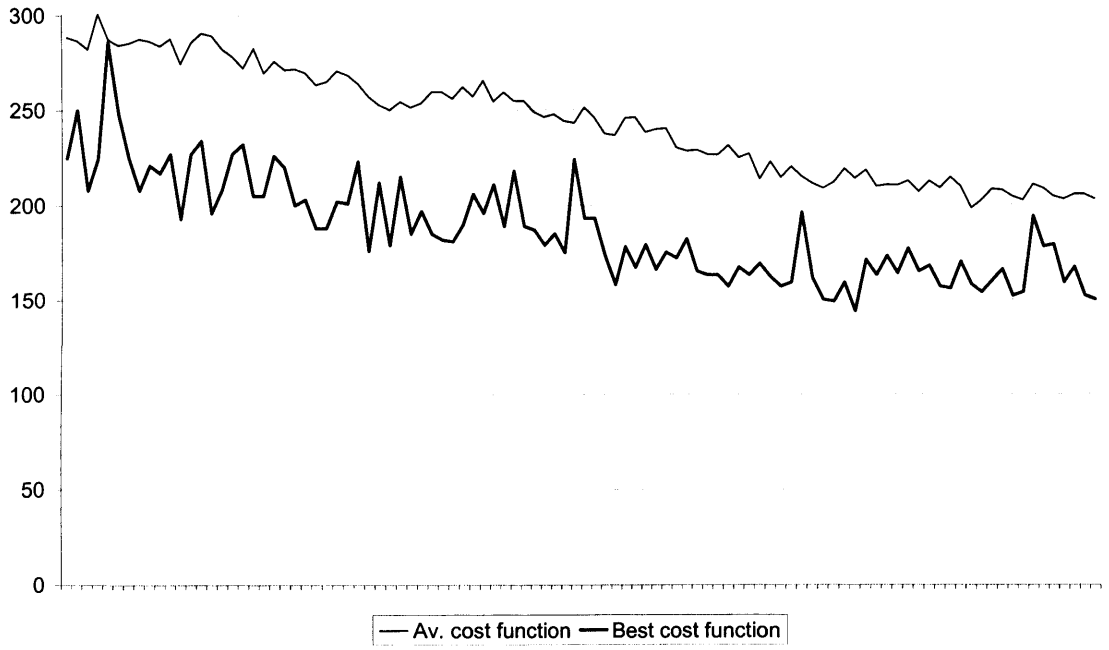


Figure 5.13. Plot of average and best cost function values against AS cycle for dataset 52, using the nurse pattern trail with the additive *Combined* heuristic, with $w_c = 5$, $w_v = 2$ and $W = 10$, as in the initial experiments.

5.3.6 Discussion of untested parameters

In Section 5.3.1, we listed the nine parameters associated with the AS algorithm for nurse scheduling and discussed their significance. Of those, the weights w_c, w_p, w_v, w_t and W have all been investigated and suitable values found. The remaining parameters have been assigned a single value for each experiment and we discuss each of these now.

Number of ants per cycle

The value of *nants* and its potential impact on solution quality has already been discussed; for the *LastChance* heuristic it is likely that a smaller number of ants may be more suitable and, in particular, setting *nants* equal to the number of nurses is likely to give better results. However, when an element of descent is included, the particular value of *nants* becomes largely irrelevant as, in our case, only one ant is used to update the trail matrix; instead, it is really the number of ants to which local search is applied and which affect the trail that becomes important.

Number of cycles

As we have discussed, the number of cycles, *gen*, was initially set to equal that of the GRASP approach so as to provide a direct comparison between the two. For the runs where the trail does not provide adequate support and the solution quality fluctuates seemingly randomly from cycle to cycle, the number of cycles used is not very important. Fewer runs would result in a reduced computation time, but would not be likely to affect solution quality. Thus a smaller value of *gen* is not worth investigating at this stage. From the runs with and without local search for which the trail matrix was beneficial, the value *gen* = 100 seems adequate and it is unlikely that solutions would be much improved by allowing further cycles. Further, by increasing the number of cycles significantly, solution times would also be significantly increased.

Weight of selected components, Q

We have already discussed this parameter; its role within the algorithm is minimal and ceases to affect solution quality after a number of cycles have been completed. Since it is widely accepted that the value for Q has a negligible impact on the effectiveness of the algorithm, it is not necessary to experiment with this parameter further.

Evaporation rate

The evaporation rate is integral to the success of the AS algorithm, as we discussed earlier, determining how lasting the effects of the trail updates are. A high value indicates the results of several previous cycles are contributing to the trail score, while a low value means that only very recent cycles will impact on the assignment choices of the next. In much of the literature a value of $\rho = 0.5$ is adopted with some papers choosing a lower value. There is little evidence of higher values of ρ being successfully adopted. In the original experiments, performed without the use of local search, the value of ρ chosen resulted in the trail matrix values converging until just a few potential shift patterns were being considered for each nurse by the final cycle. In some ways this produces a similar effect as the GRASP heuristic choosing only from the best n, although in this case there is no way to determine whether these remaining patterns are actually the best. However, the value of ρ is suitable, since the number of non-zero trail matrix values after several cycles still provides enough choice to allow a diverse range of solutions, while distinctly promoting the

components deemed from previous cycles to be the most advantageous. However, this situation is slightly altered once the local search is implemented. Ordinarily, a lower value of ρ is used in conjunction with a small population of ants to ensure that the trail still represents information from a wide range of solutions. For our problem, when local search is included and only one ant is used to update the trail, it is important that the *nants* in the following cycle are based mainly around this solution as opposed to the components of several solutions to avoid the problem where components from several good solutions are combined, but form a solution which is not feasible. However, it can be seen that the trail matrix values do converge much more quickly, as is to be expected, and there could be an argument for using a lower value of ρ to prevent the convergence of trail values to a single shift pattern for each nurse, as does happen for a few datasets. However, although this may improve solution quality, it is unlikely to have enough impact to make further testing worthwhile at this stage.

5.4 Conclusions

The aim of this thesis is to investigate balancing different constraints within the framework of a constructive metaheuristic approach. In the last chapter, a GRASP approach was utilised and found to be successful in this respect. Combining the knapsack model alongside a heuristic employing appropriately tuned weights in the construction phase and combining this with a relatively simple local search proved to be a powerful method; this simple method was powerful enough to deal effectively with the conflict between feasibility and optimality, finding high-quality feasible solutions in all cases. Although this approach was later improved with slightly more problem-specific information, it was clear that a simple GRASP framework was suitable for dealing with the conflicting constraint information and able to balance the bias towards each problem aspect such that both costs were reduced sufficiently. This chapter has investigated the same questions using an AS framework. However, while the GRASP approach had a relatively straightforward implementation, the initial implementation of an ACO approach was complicated by the necessity of finding a suitable trail definition, as well as values for the large number of inbuilt parameters.

The experiments carried out in Section 5.3 have provided a sensible trail definition and tested the algorithm with a reasonable range of parameters. Results showed similarities with the GRASP approach with regards to which constructive approaches were successful; the *Holistic* and *LastChance* heuristics were again shown to be superior and the inclusion of the knapsack model once again proved vital to finding feasible solutions. However, the AS method proved less able to successfully balance the conflicting constraints of feasibility and optimality. Even with an element of local search, the number of optimal solutions found showed the method to be much less robust in this respect than its GRASP counterpart and the possible reasons for this have already been discussed. The impracticality of applying trail matrix information to a non-linear problem is one of the possible reasons for the poorer quality of the solutions; by giving each assignment an overall score based on both feasibility and preference cost no regard is given to the fact that the feasibility of the solution rests with the successful combination of solution components, rather than with the necessity of including particular assignments. Another possible reason put forward for the lack of success of the AS algorithm was that the heuristic was allowed to choose from all possible assignments at each stage, rather than from just the best few, as was the case with the GRASP. The suitability of the weights within the GRASP heuristic in balancing the conflicting constraints was shown in Chapter 4, but the AS was allowed to select assignments which would necessarily have been excluded from the GRASP constructions. This could be countered by altering the heuristic to allow more distinction between high and low scores, by increasing the weights w_c and w_p using the multiplicative heuristics, for example, but this would alter the balance of importance assigned to each type of constraint.

However, it must be noted that the AS approach applied in this chapter, whilst not providing the level of solution quality or robustness exhibited by the GRASP, has shown that with the bias in the construction provided by the application of the knapsack model and the small element of local search, it is capable of providing feasible solutions for all datasets, something which not all previous methods in the literature can claim, and although the preference costs have not been optimised in most cases, for several datasets the quality of the solutions obtained is reasonable.

We can therefore conclude that, while the practicality of applying and ACO approach to other problems with conflicting constraints may be questionable, due to the large number of possible parameter choices and the difficulty of finding a suitable trail definition, the ACO metaheuristic provides a framework in which a reasonable balance may be obtained.

Doubtless, there are many ways in which solution quality may be improved. We have already discussed how altering parameters such as *nants* and ρ may provide some improvement and, of course, there are several variants of ACO which have not been applied and which may be more able to produce good solutions. Ant Colony System, for example, provides a slightly different framework: firstly, only the best solution created so far in the run is used to add pheromone to the trail matrix; secondly, each solution component is chosen probabilistically as for AS for $q > q_0$, where q_0 is a parameter in the range [0,1] and q is a random variable in the range [0,1], but the best-scoring choice is made whenever $q \leq q_0$; and thirdly, the trail matrix is updated after each assignment has been made in order to diversify, by altering the trail of the solution component which has just been utilised. Clearly, some of these additions to the basic AS algorithm may be beneficial to the success of an ACO approach as applied to this problem; the fact that in a random proportion of cases the best-scoring assignment is made would certainly be likely to increase solution quality. Another possible way to improve solution quality is to enhance the local search or to apply local search to more than one solution per cycle.

However, the aim of this chapter was to determine whether a simple ACO approach would be able to handle the conflicting constraint information and create a suitable balance between the feasibility and optimality costs in the construction, rather than finding solutions of the highest possible quality. Improving the algorithm further, in order to enable an ACO approach to rival that of the final GRASP produced in Chapter 4, is a possible subject for further research.

Chapter 6

Medical student scheduling with GRASP

6.1 Introduction

In Chapter 2 we introduced the medical student scheduling problem as one which provides an interesting array of conflicting objectives which must be successfully balanced during any solution approach if high-quality solutions are to be found. We mentioned that, unlike for the nurse scheduling problem, it is possible to guarantee feasible solutions; however, the feasibility issue is not so trivial that it can be ignored completely and so a trade-off still exists between the hard and soft constraints. Further, where the nurse scheduling problem had just one set of costs relating to the soft constraints, the medical student scheduling problem presents three separate costs relating to the soft constraints and this allows further investigation of the compromise necessary in order to achieve a good balance. Despite the many differences between the two problems, they are similar enough to allow the techniques applied to nurse scheduling to

be applied to medical student scheduling as well and the constructive heuristics developed for medical student scheduling in Section 6.2.1 are based on the same ideas as those developed for nurse scheduling in Chapter 4. The exploitation of the underlying feasibility structure, which allowed the knapsack model to be applied to the nurse scheduling problem, is also mirrored in this chapter, where a network flow model is introduced in order to satisfy the hard constraints. We shall later show how this model can be developed into a full construction heuristic in its own right and the experiments using this approach, detailed later in the chapter, will be shown to provide interesting results.

We showed in Chapter 5 that an ACO algorithm was unable to provide very good solutions to the nurse scheduling problem due to the lack of linearity between the cost of solution components and the final solution cost. The fact that each nurse's pattern was dependent on every other for feasibility meant that, in terms of feasibility, a good component of one solution may be very poor in another and an ACO approach was not able to consolidate the information from the previous cycles in order to provide reliable information about the likely value of each assignment. These same potential problems are present for medical student scheduling since, even if a complete schedule were considered for each student at each stage, the student pair constraints create an automatic interdependency between solution components. However, beyond that, the underlying symmetry of the problem, mentioned in Chapter 2, means that it would be impossible even to identify which component from one solution is equivalent to a specific component from another. Thus, even if ACO had been successful when applied to nurse scheduling, it may not be sensible to apply it here. However, although this traditional form of feedback may not be realistically applied, we do manage to introduce a form of learning between cycles, by storing part of the full schedule from the previous cycle. The details of this memory approach will be described later in the chapter.

We therefore begin our investigation into the medical student scheduling problem by applying a straightforward GRASP algorithm, incorporating ideas from the GRASP approach for nurse scheduling in Chapter 4. The rest of the chapter is organised as

follows. In the next section we discuss how a GRASP approach may be applied to medical student scheduling and present the construction heuristics which will be investigated, discussing how a network flow model may be incorporated to ensure feasibility. The section concludes by introducing suitable neighbourhoods for the improvement phase. Section 6.3 presents a discussion of the 48 datasets investigated, providing details of how these were manufactured from the original two provided and Section 6.4 gives details of the experiments carried out using the basic construction heuristics and local search neighbourhoods detailed in Section 6.2 and the results obtained. Section 6.5 then gives details of further enhancements to the algorithm and, finally, Section 6.6 provides conclusions and suggestions for further research. Note that the notation for this chapter is as defined in Chapter 2 and is summarised in Appendix D.

6.2 Solution approach

This section is concerned with the solution approaches with which we shall tackle the medical student scheduling problem. Since we aim to solve this problem using GRASP, we shall be discussing the construction algorithms with which it may be sensible to proceed along with the possibility of ensuring feasible solutions at the end of the construction phase. The latter part of this section will then detail some of the local search neighbourhoods available for use in the improvement phase.

We begin with an examination of the construction phase.

6.2.1 Construction

The GRASP method consists of both a construction phase and an improvement phase, the construction phase gradually building a solution from either a partial or, more usually, an empty starting solution. The manner in which the full schedule will then be assembled is not pre-determined, but may be approached in several ways; the particular heuristics used to score and select components to add to the schedule must be chosen such that they create the careful balance between the different constraints necessary to arrive at good

solutions. This section describes the particular heuristics and construction approaches we have decided to investigate, with explanations of why these choices have been made.

Since we are dealing with such a large scheduling problem, it makes sense to use a systematic approach in order to solve it. This leads us to two options for how to proceed: either we can approach the problem one timeslot at a time and make sure all students are scheduled before proceeding to the next timeslot or we can proceed student by student and give each student a full schedule of five placements before continuing on to the next student. The option to pick both student and timeslot simultaneously was discarded for two reasons. The first is that by doing this, it will take much longer to incur any costs and, as such, we are more likely to be forced into making very poor choices when scheduling the last few placements. The second reason is that there is no obvious way to ensure feasibility when scheduling in this way. As will be discussed in Section 6.2.2, creating the schedules in a more orderly fashion, either student by student or timeslot by timeslot will permit feasible solutions to be guaranteed.

Now our two approaches have been decided upon, we will detail the four construction heuristics we have chosen to investigate, based on the *Cover*, *Combined*, *Holistic* and *LastChance* heuristics used to tackle the nurse scheduling problem. Note that for the nurse scheduling problem, both additive and multiplicative methods were introduced to combine the feasibility and optimality parts of the scores. However, results showed that, while there were significant differences between heuristics, there was generally little difference between the results obtained using an additive or a multiplicative approach. We therefore use only one such approach in this chapter, selecting the multiplicative approach as it proved to be the more robust of the two. This section gives details of each heuristic applied.

Note that, at the start of the construction, students form a homogenous group and only become distinct once they have been assigned a firm for their first placement in timeslot 1. In this problem, for which the exact characteristics of the 48 datasets presented are discussed in Section 6.3, the first timeslot is tight, and so always has the same number of

places available as students to be scheduled. Given that every place in each firm in timeslot 1 must be filled by a student, each student must take exactly one place and all students are identical when no assignments have been made, we may assign each student a place in timeslot 1 and use the construction algorithm to schedule only the remaining four timeslots. Even if students are to be assigned one at a time, rather than timeslot by timeslot, all students may still be assigned their first placement without affecting the effectiveness of such a method.

To assign each student a firm in timeslot 1, we merely fill the places in each firm in turn, student by student. This results in a complete schedule for this timeslot.

Let S be the set of students and S^+ the set of students who have been assigned their first placement. Here s is a student in S, f is a firm offering speciality 1, C_f is the capacity of firm f and C_{fl}^+ is the number of students currently placed in firm f in timeslot t. Then we proceed as in Figure 6.1.

Procedure to allocate first timeslot

Step 1: Set $S^+ = \varnothing$.

Step 2: Set $f = 1$, $s = 0$ and $C_{f1}^+ = 0$, $\forall f$.

Step 3: Set $s = s+1$.

Step 4: If $C_{f1}^+ = C_f$, set $f = f+1$.

Step 5: Make the allocation $x_{sf1} = 1$ and set $S^+ = S^+ \cup \{s\}$, $C_{f1}^+ = C_{f1}^+ + 1$.

Step 6: If $S^+ \neq S$ Go to Step 3.

Figure 6.1. Procedure to allocate the first timeslot.

We will not consider taking each student in turn and giving them a full schedule at this stage, but concentrate instead on creating schedules timeslot by timeslot. The reasons for this will be explained in Section 6.2.1.5. We now describe the construction phase using a

score function $y(s,f,t)$ for the case where timeslots 2-5 are scheduled one at a time. Let t be the current timeslot being scheduled. Here, S^+ is the set of students who have been assigned a placement in timeslot t. We introduce the notation D_q to represent the set of all students who have already been assigned to a firm covering speciality q. The procedure is shown in Figure 6.2.

Procedure to allocate timeslots 2-5

Step 1: Set $t = 2$.

Step 2: Set $S^+ = \varnothing$ and $D_q = \varnothing$, for $q = 2, \ldots, 5$.

Step 3: Calculate the score $y(s,f,t)$ associated with allocating student s to firm f in the current timeslot for all pairs (s,f) §, with $s \notin D_q$ if $a_{fq} = 1$.

Step 4: Let L be the candidate list of the n highest scoring options.

Step 5: Select $(s,f) \in L$ using roulette wheel selection, i.e. each (s,f) is selected with a relative probability proportional to $y(s,f,t)$.

Step 6: Update the schedule with this allocation and set $S^+ = S^+ \cup \{s\}$ and $D_q = D_q \cup \{s\}$ for q such that $a_{fq} = 1$.

Step 7: If $S^+ \neq S$ go to Step 3.

Step 8: If $t \neq 5$, $t = t + 1$. Go to Step 2.

§ Note that the set of feasible allocations (s,f) may include all students not already allocated, or may relate to a single student s given by a predefined ordering.

Figure 6.2. Procedure to allocate timeslots 2-5.

As for the nurse scheduling heuristics, we have two scores for the feasibility. Let us define a value for the number of places remaining in a speciality q at time t, d_{qt}, as

$$d_{qt} = \sum_{f:a_{fq}=1} \max\{C_f - C_{ft}^+, 0\}. \qquad (6.1)$$

If we set $q' = \arg\max\{d_{qt}\}$, we can define the first of these two feasibility scores, $fscore1_{ft}$ to be

$$fscore1_{ft} = \begin{cases} \max\{C_f - C^+_{ft}, 0\} & \text{if } a_{fq'} = 1 \\ -1 & \text{otherwise} \end{cases}. \qquad (6.2)$$

We choose q' as the speciality with the highest number of remaining spaces and only give a score to a firm f if it offers this speciality. The score given to a candidate firm f, $fscore1_{ft}$, is then simply a measure of the remaining number of available places in this firm in the current timeslot. The value -1 assigned to firms not belonging to q' is an arbitrary negative value to ensure firms not offering speciality q' are given a lower score than firms which offer q', but have no places remaining and any firm with a negative score is automatically not included in the roulette wheel selection process.

Since each student must cover each speciality, it is important that we do not schedule too many students to do each speciality in the early stages as we may have difficulty in obtaining feasible schedules in the later stages of the construction. In the data provided, firms offering the same speciality tend to have a similar number of places, but different specialities will have different numbers of firms with different numbers of places available. If only the remaining number of spaces in each firm were considered, then, near the beginning of the construction, specialities with a smaller number of higher capacity firms would be consistently selected before any other specialities were considered. By restricting the choice of firm to one from a speciality where fewer people have already been placed, we hope to ensure a more even distribution of students across the different specialities from an early stage, in order to improve the chance of being able to directly construct feasible solutions. Note that the logic behind this idea uses a similar approach to that of the *Cover* heuristic applied to the nurse scheduling problem, where choices were limited to just days or nights.

To create our second feasibility score, we allow firms from all specialities to be considered. By doing this we hope to direct the search to finding high quality solutions, rather than just feasible ones. In Section 6.2.2 we are able to introduce an additional algorithm to guarantee feasible solutions. Given this, we wish to ensure that our constructions are still concerned with finding low-cost solutions; it would be undesirable to direct the search towards feasible solutions to the detriment of final solution quality.

We thus define our second feasibility score, $fscore2_{ft}$ to be

$$fscore2_{ft} = \max\{C_f - C^+_{ft}, 0\}, \tag{6.3}$$

for all firms f.

We now define the optimality score for a given firm f in timeslot t, based on the fulfilment of the three soft constraints. The score comprises three components: one from each of the three soft constraints. The part of the score associated with the consultant clashes, O1, is given by equation (6.4). The part of the score from hospital clashes, O2, is given by (6.5) and (6.6) gives the score from the number of student pairs, O3.

$$oscore1_{sft} = \sum_{u=1}^{t-1}\sum_{g=1}^{v} c_{fg}x_{sgu} \tag{6.4}$$

$$oscore2_{sft} = \sum_{u=1}^{t-1}\sum_{g=1}^{v} h_{fg}x_{sgu} \tag{6.5}$$

$$oscore3_{sft} = \sum_{u=1}^{t-1}\sum_{\substack{z=1 \\ z\neq s}}^{r}\sum_{g=1}^{v} x_{sgu}x_{zgu}x_{zft} \tag{6.6}$$

The way in which (6.4) and (6.5) are calculated is straightforward; we calculate the number of times in the previous timeslots where student s has been in either the same

hospital or with the same consultant as firm f. We have essentially the same scores as (2.4) and (2.5), put forward in Chapter 2, but we ignore all irrelevant costs and concentrate solely on those which would be affected by the new assignment.

When we calculate (6.6), we need to calculate the effect on the student pairs cost, of adding student s to firm f. We cannot use the same type of calculation as before, since we are counting the occurrence of pairs, where a single pair is not to be penalised. When investigating hospital and consultant clashes, every pair incurs a cost. Instead, we calculate this score, $oscore3_{sft}$, as the number of times students s and z have been placed together in previous timeslots, where z has already been placed in f in timeslot t. We use this score because if students s and z are placed together in timeslot t, the student pairs part of the cost will increase by the number of times these students have been placed together in the previous timeslots. This is demonstrated in Table 6.3, where N_{szt} represent the number of times students s and z have been placed together in timeslots 1 to $t-1$.

N_{szt}	Cost due to N_{szt}	Cost if $N_{szt} = N_{szt}+1$	Increase in cost
0	0	0	0
1	0	1	1
2	1	3	2
3	3	6	3
4	6	10	4

Table 6.3. Explanation for equation (6.6).

We can therefore use the value of N_{szt} to calculate the change in cost achieved by allocating student s to firm f. Note that $N_{szt} = 5$ indicates that student s (and z) has already been given a full schedule and so would not be considered.

Combining these three scores into a single optimality score, $oscore_{sft}$, and using the weights we decided upon previously, we obtain

$$oscore_{sft} = \sum_{u=1}^{t-1}\sum_{g=1}^{v}\left(50c_{fg}x_{sgu} + 10h_{fg}x_{sgu}\right) + \sum_{u=1}^{t-1}\sum_{\substack{z=1 \\ z\neq s}}^{r}\sum_{g=1}^{v} x_{sgu}x_{zgu}x_{zft} \,. \qquad (6.7)$$

We now demonstrate how these three scores are used to create the four heuristics we will base our experiments on.

6.2.1.1 Feasibility

We define the score function for the *Feasibility* heuristic to be

$$Feasibility: g_1(f,t) = fscore1_{ft}. \tag{6.8}$$

This heuristic is similar to the *cover* heuristic used to solve the nurse scheduling problem; only the feasibility is taken into account, with the soft constraints disregarded. For initial experiments we use this heuristic with a predefined ordering of the students. We therefore calculate $g_1(f,t)$ where t is the current timeslot being considered, for all firms f which cover a speciality the particular student under consideration has not yet been assigned to.

6.2.1.2 Combined

The score for the *Combined* heuristic is defined as

$$Combined: g_2(s,f,t) = \frac{fscore2_{ft}^{\ w_f}}{oscore_{sft}^{\ w_o} + 1}, \tag{6.9}$$

where w_f and w_o are constants. This heuristic uses a pre-defined ordering of the students. Thus $g_2(s,f,t)$ is used to calculate the scores for each firm f covering a speciality student s has not yet covered, where s is the particular student being scheduled and t is the current timeslot being considered.

This heuristic, as indicated by the name, is based on the *Combined* heuristic used for the nurse scheduling problem. Multiplying by $fscore2_{ft}$ will result in a zero score for all firms f already filled to capacity in timeslot t, irrespective of how few other costs would be incurred. We divide by $oscore_{sft}$ since a larger cost here is undesirable. The larger the optimality costs, the smaller our overall score. We add 1 to this denominator to prevent division by zero.

6.2.1.3 Holistic

The *Holistic* heuristic for the medical student scheduling problem is, once again, a variant on the heuristic of the same name produced for the nurse scheduling problem. Here we use the same basic scoring method as for the *Combined* heuristic, but, rather than taking all the students in a set order, we select the student and firm at the same time. This heuristic only makes sense if we schedule one full timeslot before tackling the next. If we were to create a full timetable for each student before moving on to the next, there would be little advantage in taking the students out of order. There is the option to select both the timeslot and firm for a pre-selected student, but given the symmetry between timeslots at all stages of the construction, again, there seems little purpose to this endeavour.

The scores for the *Holistic* heuristic are therefore

$$Holistic: g_3(s,f,t) = \frac{fscore2_{ft}^{\ w_f}}{oscore_{sft}^{\ w_o} + 1}, \tag{6.10}$$

but in this case the scores are calculated for all s, and all f relating to s in timeslot t. Again, w_f and w_o are parameters to be decided.

6.2.1.4 LastChance

We relate this heuristic back to the nurse scheduling heuristic of the same name. Let G_{st} be the set of all firms f, such that it is feasible for student s to be allocated to f in timeslot t.

Then let $f^*(s,t) = \arg\max_{f \in G_{st}} \{g_3(s,f,t)\}$.

We then calculate the scores for the *LastChance* heuristic as

$$LastChance: g_4(s,t) = \max_{f \in G_{st}} \{g_3(s,f,t)\} - \max_{\substack{f \in G_{st} \\ f \neq f^*(s,t)}} \{g_3(s,f,t)\}. \tag{6.11}$$

As for the *Holistic* heuristic, it makes little sense to apply this, as it stands, to the case where we schedule by student rather than by timeslot, since in each phase of the construction this algorithm would lead us to choose a student to schedule. We would then assign student s to firm $f^*(s,t)$.

To make this heuristic fit with the student by student approach we would have to work out the scores for complete student timetables. We would then pick the student with the largest difference between their best and next best schedules and assign them so as to achieve the lowest possible cost. Obviously, this is a much more time-consuming enterprise than the original scheme.

6.2.1.5 Choosing between scheduling approaches

Since two of our four construction heuristics must be scheduled by timeslot rather than by student, we shall proceed with the timeslot by timeslot method of construction. There is no reason to suppose that scheduling students one at a time would be a more successful construction technique and, given that the two heuristics which cannot be used in this way are based on the more successful of the heuristics applied to nurse scheduling, this seems a sensible approach to take. The final decision on which approach to use will be discussed once the approaches to ensure feasibility have been analysed for both types of construction.

6.2.2 Ensuring feasibility

As discussed previously, we are able to present a method by which we can guarantee all solutions will be feasible at the end of the construction. Once feasibility constraints are satisfied, the improvement phase may concentrate solely on lowering the costs relating to the soft constraints. This section discusses the manner in which we may produce feasible solutions.

Since each speciality has the same number of places available in each timeslot, we already have a trivial method for creating feasible schedules, by allowing each group of four students to cycle through the specialities such that each speciality is covered exactly once in each timeslot. This would distribute the students evenly across the specialities over the four timeslots and ensure that the feasibility criteria are met. However, by adhering to such a constricting system, we inevitably lose the required flexibility to build good quality solutions as was demonstrated by the example in Figure 2.3 in Chapter 2. As such, this type of cyclic scheduling will not be explored further.

Since we are creating the schedules timeslot by timeslot we need to guarantee that at the end of each timeslot i, each speciality q will have accepted a number of students, A_{qi}, such that the number of students yet to study speciality q does not exceed the number of spaces available. This will be achieved if we satisfy conditions (6.12) and (6.13).

Inequality (6.12) prevents the number of students assigned to speciality q in the current timeslot from exceeding the number of spaces available in q while inequality (6.13) says that the number of students who must still study q at the end of timeslot i, given by the number who needed to study q at the beginning of timeslot i less those who have been allocated to study it in i, must be less than the capacity of q in the remaining timeslots. For a general case with N timeslots, this gives

$$A_{qi} \leq \sum_{f=1}^{v} C_f a_{fq} \ , \tag{6.12}$$

$$r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} - A_{qi} \leq \sum_{f=1}^{v} \sum_{t=i+1}^{N} C_f a_{fq} \tag{6.13}$$

where

$$r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq}$$

represents the number of students who have not yet studied q and

$$\sum_{f=1}^{v} \sum_{t=i+1}^{N} C_f a_{fq}$$

represents the number of places available in q in the remaining timeslots. Then, by rearranging (6.13) we obtain a lower bound for A_{qi}, given by inequality (6.14).

$$A_{qi} \geq r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} - \sum_{f=1}^{v} \sum_{t=i+1}^{N} C_f a_{fq} \tag{6.14}$$

Thus (6.12) and (6.13) impose necessary conditions on the upper and lower bounds for A_{qi}, respectively, for the solution to be feasible. However, although these two conditions are necessary for a feasible solution to exist, they are not necessarily sufficient. Consider the counterexample given in Figure 6.4.

Counterexample showing (6.12) and (6.13) are not sufficient for feasibility

Consider a situation with 16 students, where the following numbers of places are available in each speciality in each timeslot.

		Timeslot				
		2	3	4	5	Total
Spec.	2	3	4	5	5	17
	3	4	3	5	5	17
	4	5	5	3	4	17
	5	5	5	3	4	17
	Total	17	17	16	18	

There is enough space in each timeslot and each speciality for a feasible solution to be possible; the following permutations each worked by the given numbers of students: (2,3,4,5) ×3, (3,2,5,4) ×3, (4,5,2,3) ×5 and (5,4,3,2) ×5, give an example of a feasible allocation of the students' speciality permutations.

However, by obeying (6.12) and (6.14), such feasibility is not guaranteed. Allocating students to specialities in the first two timeslots thus: (2,3) ×3, (3,2) ×4, (4,5) ×5, (5,4) ×4 would leave the correct number of places remaining for each student to study their remaining specialities in timeslots 4 and 5 and so (6.12) and (6.15) would be satisfied. However, we would be left with the situation where 7 students must study specialities 4 and 5 in the last two timeslots and although there are enough places in total, there is not enough space in timeslot 4 for 7 students to study these two specialities.

Figure 6.4. Counterexample showing (6.12) and (6.13) are not sufficient for feasibility.

This counterexample shows that (6.12) and (6.13) are not sufficient for feasibility in the general case. However, in the particular cases investigated in this research, the same number of spaces are available for each speciality in each timeslot. We now prove that, when equal numbers of spaces in each speciality are available in each timeslot, conditions

(6.12) and (6.13) are sufficient for feasibility. We first define a number of terms necessary for the proof:

Bipartite graph:

A graph $G(V,E)$ whose vertices V may be partitioned into two disjoint subsets with no edge $e \in E$ connecting two vertices within the same subset.

A Matching:

A set of edges such that no two edges have a vertex in common.

A perfect matching:

A matching of a graph $G(V,E)$ containing $|V|/2$ edges.

Hall's Marriage Theorem:

Let $G(V_1,V_2,E)$ be a bipartite graph with partitions V_1 and V_2 and $|V_1| \leq |V_2|$. Then there exists a matching such that every vertex in V_1 is connected iff, for every subset V' of V_1, $|\cup V'| \geq |V'|$, where $|\cup V'|$ is the number of vertices in V_2 adjacent to vertices in V'.

6.2.2.1 Proof of feasibility

We prove by induction that (6.12) and (6.13) are sufficient in order for a feasible solution to exist for the case where there are N timeslots needing to be scheduled.

First, consider inequality (6.13). At the beginning of timeslot i, the number of students who will study speciality q in timeslot i may be incorporated into the right hand side of the equation, leaving the more general inequality

$$r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \leq \sum_{f=1}^{v} \sum_{t=i}^{N} C_f a_{fq}, \qquad (6.13')$$

which indicates that the number of students who are left to study q in the remaining timeslots does not exceed the number of places available.

Consider the situation where just one timeslot remains to be scheduled, timeslot N. Assuming that (6.12) and (6.13') have held throughout timeslots 1 to $i - 1$, each student will have just one speciality left to cover and, if (6.13') holds at the beginning of this timeslot, a feasible solution exists, since this tells us there are enough places for each student to study their remaining speciality.

Let us now consider an arbitrary timeslot i and assume that inequalities (6.12) and (6.13') have been satisfied for timeslots 2 to $i - 1$. We must show that if we start i with (6.12) and (6.13') satisfied, we can find an assignment that will then allow us to start timeslot $i + 1$ with (6.12) and (6.13) still satisfied.

Let us set up a bipartite graph $G(V_1,V_2,E)$ as follows. Let V_2 be the set of all the places available in timeslot i and let V_1 comprise two subsets: V_a and V_b. Let V_a be the set of all r students and let V_b be a set containing one 'dummy student' for each extra place in V_2 which exceeds the size of the subset V_a. Thus V_b is set up to ensure that the number of places available in V_2 is equal to the number of 'students' in V_1. If we think of this in terms of the specialities, this gives

$$|V_b| = \frac{(N-i+1)\sum_{f=1}^{v} C_f - \sum_{q}\left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft}a_{fq}\right]}{N-i+1} \tag{6.15}$$

which can be rewritten as

$$|V_b| = \frac{\sum_{q}\left[(N-i+1)\sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft}a_{fq}\right]\right]}{N-i+1} \tag{6.16}$$

where

$$(N-i+1)\sum_{f=1}^{v} C_f a_{fq}$$

is the total remaining capacity in speciality q, and

$$r-\sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq}$$

is the number of students still needing to do speciality q. Essentially, $|V_b|$ is given by summing the capacity of all firms in a given timeslot and subtracting the total number of students, r; by considering that all students still need to study $N - i + 1$ specialities in the $N - i + 1$ remaining timeslots, the size of $|V_b|$ summed over all remaining timeslots is obtained. We therefore divide by the $N - i + 1$ timeslots remaining to obtain the value for $|V_b|$ in a single timeslot. Note that $|V_b|$ is constant throughout since the value of $|V_b|$ is defined as the total capacity less the number of students and, in the problem considered here, the total capacity of each speciality is the same in each timeslot.

Now consider a variable $slack_q$, which is defined as the maximum number of spare places we can allow in q in timeslot i. This is given by

$$slack_q = \sum_{f=1}^{v} C_f a_{fq} - \min(A_{qi}) \tag{6.17}$$

$$= \sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} - (N-i)\sum_{f=1}^{v} C_f a_{fq} \right] \tag{6.18}$$

$$= (N-i+1)\sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \tag{6.19}$$

Note that (6.18) is obtained from making the substitution given by (6.13). The constant '$N - 1$' in (6.18) is given by the fact that this part of the sum is over timeslots i+1 to N in inequality (6.13) whereas, until this point, we have been referring to (6.13'), which incorporates the A_{qi} values and is therefore summed over timeslots i to N.

For each q then pick $slack_q$ variables in V_2 corresponding to speciality q and denote these sets V_q.

Now define an edge between each vertex in V_a and each vertex in V_2 which corresponds to a place in a speciality this student has not yet covered. Further, define an edge between every vertex in V_b and every vertex belonging to a place in a set V_q.

Proposition:
Every perfect matching in $G(V_1,V_2,E)$ corresponds to a feasible assignment for period i satisfying (6.12) and (6.13).

This will be true because every student vertex in V_a will be matched with a vertex in V_2 corresponding to a speciality the student has not yet covered, given that only edges relating to feasible allocations were allowed and given that in a perfect matching each student vertex will have exactly one such edge. Furthermore, (6.13) will be satisfied because a maximum of $slack_q$ spaces in any speciality q will be unassigned in timeslot i and $slack_q$ is defined as the permissible number of unassigned places in q such that (6.13) will hold.

We therefore just need to prove that a perfect matching exists.

Take a subset V' of size k of vertices in V_1.

First let us assume that V' does not contain any vertices from V_b. Then this subset is adjacent to all vertices in a subset S' containing all places covering a speciality still required by any vertex in V'. We therefore have

$$|S'| = \sum_{q \in S_{V'}} \sum_{f=1}^{v} C_f a_{fq}\,, \tag{6.20}$$

where $S_{V'}$ is defined as the set of specialities required by at least one student vertex in V'. Since each element of V_a is adjacent to $N - i + 1$ specialities, we have the result

$$|V'| \le \frac{\sum_{q \in S_{V'}} \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right]}{N-i+1} \tag{6.21}$$

$$\le \sum_{q \in S_{V'}} \sum_{f=1}^{v} C_f a_{fq} \tag{6.22}$$

since

$$\sum_{q} \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \le (N-i+1) \sum_{q} \sum_{f=1}^{v} C_f a_{fq}$$

by (6.13').

We have therefore shown that for the case where V' consists only of vertices from V_a, $|V'| \le |S'|$ and so, by Hall's Marriage Theorem, a perfect matching exists.

Now let us assume that V' contains k_b vertices from V_b. We therefore have

$$k_b \le |V_b| = \frac{\sum_{q} \left[(N-i+1) \sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \right]}{N-i+1} \tag{6.23}$$

$$= \frac{\sum_{q \in S_{V'}} \left[(N-i+1) \sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \right]}{N-i+1} + \tag{6.24}$$

$$\frac{\sum_{q \notin S_{V'}} \left[(N-i+1) \sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \right]}{N-i+1}$$

We know that the size of V' can now be written as

$$|V'| = k \le |V_b| + \frac{\sum_{q \in S_{V'}} \left[r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} \right]}{N-i+1} \tag{6.25}$$

$$= \sum_{q}\sum_{f=1}^{v} C_f a_{fq} + \frac{\sum_{q \notin S_{V'}} \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} \right]}{N-i+1} \qquad (6.26)$$

by substituting the value of $|V_b|$ from (6.24) and simplifying. But V' is also adjacent to all V_q for $q \notin S_{V'}$, since each vertex in V_b has an edge to all vertices in V_q for all q, and since

$$|V_q| = slack_q = (N-i+1)\sum_{f=1}^{v} C_f a_{fq} - \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} \right], \qquad (6.27)$$

we therefore have

$$|V'| \le \sum_{q}\sum_{f=1}^{v} C_f a_{fq} + \frac{\sum_{q \notin S_{V'}} \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} \right]}{N-i+1} \qquad (6.28)$$

from (6.26) and we obtain

$$|S'| = (N-i+1)\sum_{q}\sum_{f=1}^{v} C_f a_{fq} + \sum_{q \notin S_{V'}} \left[r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft} a_{fq} \right] \qquad (6.29)$$

by adding the total number of nodes adjacent to $|V_b|$.

Comparing the right hand sides of (6.28) and (6.29) shows we have the result $|V'| \le |S'|$ as required, proving that a matching, and hence a feasible allocation of the students, exists for timeslot *i*. We have shown that if (6.12) and (6.13) hold at the beginning of timeslot *i*, then we can start timeslot *i*+1 with (6.12) and (6.13) still satisfied and we know that, if (6.12) and (6.13) are satisfied at the beginning of timeslot *N*, we find a feasible allocation for the remaining students. Furthermore, we know that (6.12) and (6.13) hold at the beginning of timeslot 2 because each problem has sufficient capacity for the problem to be feasible. It has therefore been shown that (6.12) and (6.13) are both necessary and sufficient conditions for feasibility, given that all timeslots have equal capacities for each speciality.

We have shown the bounds for the total number of students who must do each speciality, but we also know that only certain students may be included in these numbers; if a student has already done speciality q in a previous timeslot, they must not be one of the students to be assigned to q in the current timeslot.

We now introduce a new variable, by designating sixteen student 'types'. Note that these should not be confused with the nurses' 'types' introduced in Chapter 2, since the nurses were classified into types according to their contracts and remained the same type throughout the solution process. Although the student types also relate to feasibility, the difference here is that a student's type changes dynamically as the construction progresses and is related to the current construction rather than the details of the problem instance being considered. The type p of a student is determined by the specialities the student must still cover in order for the solution to be feasible. A student's type is therefore largely determined by the current timeslot, since as each student is scheduled, the number of specialities remaining for this student decreases and the student type will change. The student types are outlined in Table 6.6.

p	Specialities previously assigned
1	None
2-5	{2}, {3}, {4} or {5}
6-11	{2,3}, {2,4}, {2,5}, {3,4}, {3,5} or {4,5}
12-15	{2,3,4}, {2,3,5}, {2,4,5} or {3,4,5}
16	{2,3,4,5}

Table 6.6. A list of the specialities covered by each of the student types, categorised by timeslot.

Note that we ignore speciality 1, since this is covered by all students in timeslot 1 and consider only the remaining four specialities when referring to student type. Therefore $p = 1$ refers to a student who has been assigned to speciality 1 only. After a student has been assigned a place in timeslot 2, the student's type will be in the range [2,5], since in timeslot 2 they must have been assigned to cover exactly one speciality. Similarly, a student scheduled in timeslot 3 will have covered two of the four specialities and their type will therefore be in the range [6,11]. Once all assignments have been made, the student will be type 16. If we now look at the types, p, of student who may be eligible to

study speciality q, we can easily set limits on the number of students B_{pq} from each type p who may study speciality q. The only possible values for B_{pq} are given by

$$0 \leq B_{pq} \leq |p| \tag{6.30}$$

where $|p|$ is the number of students of type p. Note also that

$$A_{qi} = \sum_{p} B_{pq} \tag{6.31}$$

We may now express our problem as finding a way to allocate numbers of students from the types p to the specialities, q, in such a way as to ensure the upper and lower limits are adhered to. We therefore wish to find whether there are suitable values B_{pq} such that constraints (6.12) and (6.14) are met and every student is allocated a place in a firm they have not yet studied. More formally, we can say that for timeslot i we wish to find a solution to

$$B_{pq} \leq |p| \quad \forall p, q \tag{6.32}$$

$$\sum_{p} B_{pq} \leq \sum_{f=1}^{v} C_f a_{fq} \quad \forall q \tag{6.33}$$

$$\sum_{p} B_{pq} \geq r - \sum_{s=1}^{r} \sum_{f=1}^{v} \sum_{t=2}^{i-1} x_{sft} a_{fq} - \sum_{f=1}^{v} \sum_{t=i+1}^{5} C_f a_{fq} \quad \forall q \tag{6.34}$$

$$\sum_{q} B_{pq} = D, \text{ where } D \text{ is a constant.} \tag{6.35}$$

We are now left with a problem which may be modelled as a network flow and in the next section we explain this process.

6.2.2.2 Applying a network flow model to the medical student scheduling problem

Figure 6.7 demonstrates how a feasibility test for the medical student scheduling problem may now be modelled as flow through a network. The main flow variables relate to the variables B_{pq} representing students of type p being allocated to speciality q in the current time period. This diagram shows the situation during a particular timeslot i; note that the number of nodes for student type p and the number of arcs from nodes p to q will vary

depending on the timeslot being considered. Other relevant arcs and source and sink nodes have been added in order to model the situation as a minimum cost flow problem in a closed network.

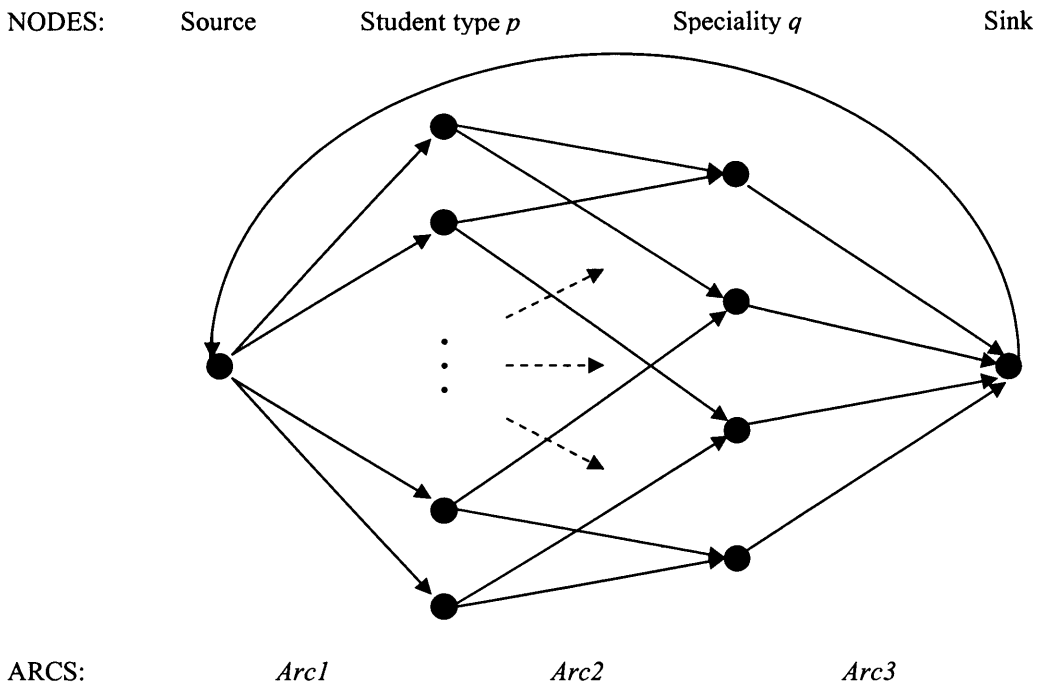


Figure 6.7. Network flow model representing the situation during an arbitrary timeslot i of the medical student scheduling problem.

All costs in the network are zero as they have no meaning in this context; the lower and upper bounds on each of the arcs, represented as l and u respectively, can be specified as follows.

Arc1

An arc (Source,p), connecting the source with a particular student type, p, has

$$l = u = |p|,$$

Arc2

An arc (p,q), connecting a student type node to a speciality node, q, has

$$l = 0,\ u = |p|$$

as given by inequalities (6.30), previously.

The flow along these arcs represents the number of students of each type who proceed to study each of the possible specialities in timeslot i. If no students of a given type p^* are assigned to a particular speciality q^*, the flow along arc (p^*,q^*) will be zero. Similarly if all students of type p^* were assigned to q^* the flow along this arc would be $|p^*|$.

Arc3

An arc (q,Sink), connecting a speciality with the sink node, has

$$l = r - \sum_{s=1}^{r}\sum_{f=1}^{v}\sum_{t=2}^{i-1} x_{sft}a_{fq} - \sum_{f=1}^{v}\sum_{t=i+1}^{5} C_f a_{fq},\ u = \sum_{f=1}^{v} C_f a_{fq},$$

as given by (6.12) and (6.14), previously.

Finally, an arc connects the sink node to the source node in order to complete the flow, with bounds given by

$$l = u = r.$$

This network flow problem can be solved relatively quickly using the out-of-kilter algorithm, detailed in Dowsland (2005). It is known that the minimum cost network flow problem can be solved in polynomial time, but, for the general case where there are N timeslots and specialities to consider, the time required for the network flow feasibility check increases exponentially with N.

Finding the existence of a feasible flow in this network for each of the timeslots means there exists a way in which we can allocate students so that the schedule will be feasible at the end of the construction. Of course in order to then create such a feasible schedule, we allocate students in a feasible manner throughout the construction phase of the GRASP cycle.

When using the knapsack algorithm to improve the feasibility of constructions for the nurse scheduling problem, we had a similar approach. Solving the knapsack model of the problem would tell us whether a feasible solution was possible, but then we had to check

each of our assignments with the model and update it accordingly after each addition to the schedule. We approach the use of the network flow model similarly.

We start by solving the problem in the above network to give an initial feasible solution and a corresponding set of flows F_{ij} along each arc (i,j). Let u_{pq} and l_{pq} denote the upper and lower bounds on arc (p,q), from type p to speciality q, respectively. Then as we progress through the construction, each time a student is assigned a firm, we test the feasibility of the flow by increasing the lower bound on the relevant arc by 1 unit. If this does not violate the upper bound and a feasible flow can be found, the assignment is allowed and the flow is altered accordingly. These lower bounds are maintained to ensure any future solution includes the current and all previous assignments. Otherwise, if no feasible flow exists with this assignment, the lower bound is reduced again and no future assignments are considered which would send flow through the arcs which are already at capacity. Essentially, each attempted assignment tells us whether or not a student of type p can study speciality q in this timeslot.

The process is summarised in Figure 6.8.

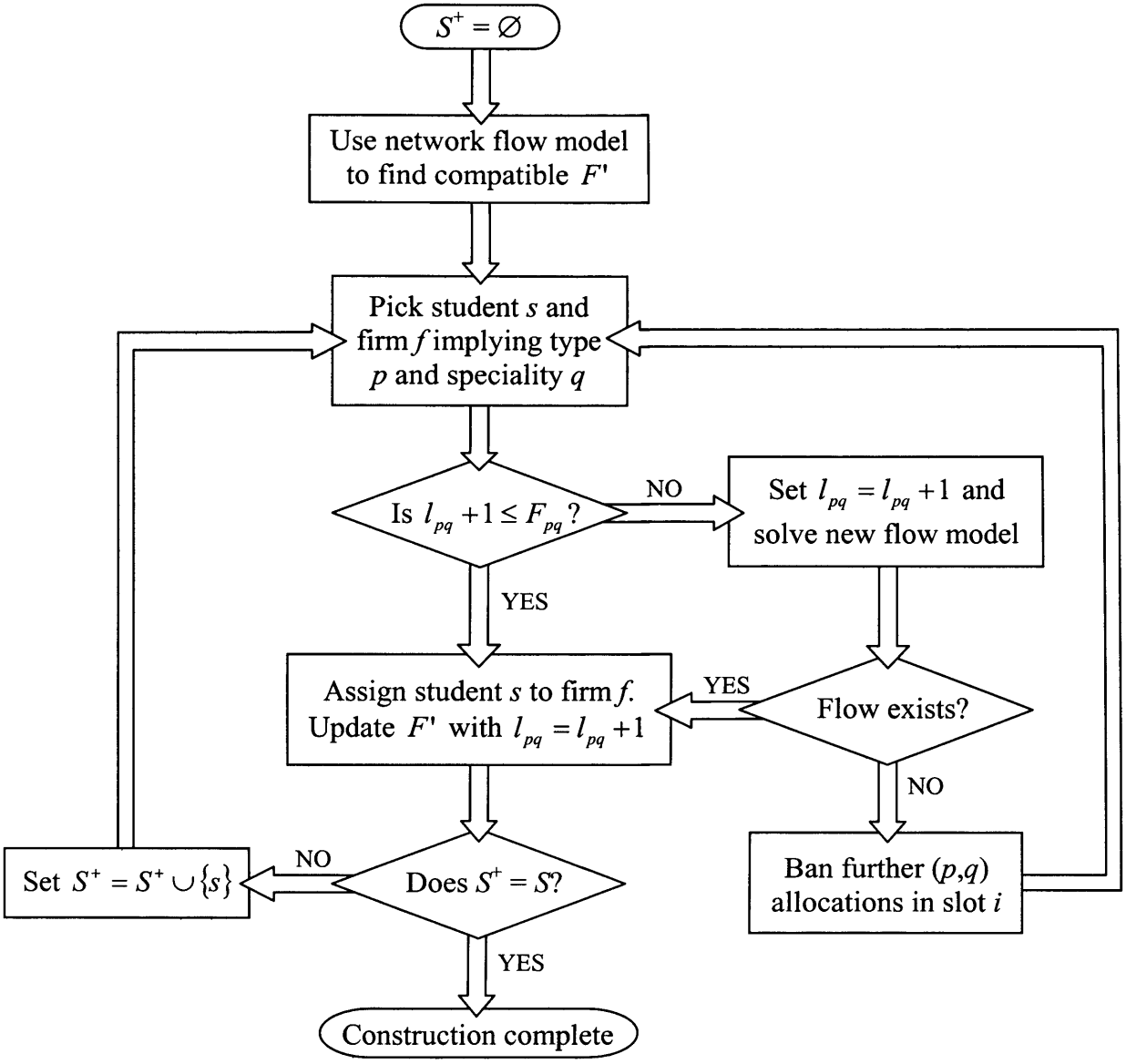


Figure 6.8. Flow chart showing implementation of the network flow algorithm.

It is known that the minimum cost network flow problem can be solved in polynomial time, but, for the general case where there are N timeslots and specialities to consider, the time required for the network flow feasibility check increases exponentially with N.

6.2.2.3 Ensuring feasibility for student by student construction

If we are creating the schedules student by student, rather than timeslot by timeslot, clearly the same logic cannot be applied. In this case, setting bounds on the number of assignments to each speciality is not sufficient. Although guaranteeing that there is enough space left in each speciality for all students still to be scheduled, by adhering to

$$\sum_{t=2}^{5}\sum_{f=1}^{v}\left(C_f a_{fq} - \sum_{s=1}^{i} x_{sft} a_{fq}\right) \geq r - i, \quad \forall q, \tag{6.36}$$

is a necessary condition for a feasible solution to exist, it is not sufficient, as the counterexamples in Figures 6.9 and 6.10 show. Each counterexample demonstrates a different type of problem: *Example 1* in Figure 6.9 shows that it is possible, when allocating a student's schedule one timeslot at a time, to reach a situation where it is not possible to find a feasible allocation for the rest of the schedule; *Example 2* in Figure 6.10 shows that it is possible, once a subset of students has been allocated, to reach a situation where there is no feasible allocation for the rest of the students, due to the fact that remaining spaces in a speciality will vary between timeslots.

Example 1

Consider a situation with 2 students, where the following numbers of places are available in each speciality in each timeslot.

		Timeslot				
		2	3	4	5	Total
Spec.	2	1	0	0	1	2
	3	0	1	1	0	2
	4	0	0	1	1	2
	5	1	1	0	0	2
	Total	2	2	2	2	

There is enough space in each timeslot and each speciality for a feasible solution to be possible; (2,5,3,4) and (5,3,4,2) would be a feasible allocation of the students' speciality permutations. However, if the first student is allocated (2,3) in the first two timeslots, there is no way to complete the current student's allocation, despite the correct number of spaces being available in total.

Figure 6.9. Counterexample showing inequality (6.36) is not sufficient for feasibility when scheduling student by student.

Example 2

Consider a situation with 3 students, where the following numbers of places are available in each speciality in each timeslot.

		Timeslot				
		2	3	4	5	Total
Spec.	2	*1*	2	2	2	7
	3	2	*1*	0	0	3
	4	0	1	*1*	1	3
	5	1	0	1	*1*	3
	Total	4	4	4	4	

By allocating the first student permutation (2,3,4,5), we have at least 2 spaces left in each speciality, and (6.36) is upheld, however, there is then no feasible way to make the remaining allocations since both students will be forced to cover speciality 3 in timeslot 2 and therefore only one student will be able to cover speciality 5.

Figure 6.10. Second counterexample showing inequality (6.36) is not sufficient for feasibility when scheduling student by student.

We must therefore develop necessary and sufficient conditions for a feasible allocation to exist. We begin with the situation in *Example 2*.

Suppose there are s students left to allocate. Let the number of spaces available in speciality q in timeslot t be denoted $Space_{qt}$. The number of places required overall in q is s, as given by (6.36). Set up the bipartite graph $G(V_1,V_2,E)$ such that V_1 is the set of specialities and V_2 is the set of timeslots and define an edge $e \in E$ between $q \in V_1$ and $t \in V_2$ for every available place in speciality q in timeslot t. There will therefore be $Space_{qt}$ edges between vertices q and t.

A feasible allocation will consist of a subset of edges E', such that $Deg_{E'}(v') = s$ $\forall v' \in (V_1 \cup V_2)$. This provides an edge relating to each speciality and each timeslot for each student. Thus, the existence of such a subgraph is a necessary condition for feasibility. We now show it is also sufficient.

An Edge Colouring:
An assignment of colours to the edges of a graph, such that no two edges with the same colour have a vertex in common.

Graph colouring theorem:
The edges of a bipartite graph with maximum degree D_{max} can always be coloured in D_{max} colours. Furthermore, this is the optimal edge-colouring.

Consider the subgraph formed by E'. The number of edges is given by

$$|E'| = \frac{(V_1 \cup V_2)s}{2} = \text{number of timeslots} \times s \ (= \text{number of specialities} \times s), \quad (6.37)$$

since the number of edges is equal to half the sum of the degrees. Thus the maximum degree is s and the graph can be coloured in s colours and each colour must be used on N edges, as all edges are included, where N is the number of timeslots (and specialities).

Thus each set of edges in a single colour form a feasible allocation for one student and so s students can be allocated.

We can represent the above conditions as a network flow problem. Let x_{qt} equal the number of edges from q to t in E'. Then there exists a suitable subgraph $G'(V_1, V_2, E')$ if we can find x_{qt} such that:

$$\sum_{q=2}^{5} x_{qt} = s \; \forall t \tag{6.38}$$

$$\sum_{t=2}^{5} x_{qt} = s \; \forall q \tag{6.39}$$

where the summations would be taken from 1 to N in the general case.

These are simple network flow constraints and will be satisfied if there is a feasible flow in the network with nodes $(S, V_1, V_2 T)$ with the lower and upper bounds, represented by LB and UB, on each arc $\forall q, t$ shown in Table 6.11.

Arc	LB	UB
(S,q)	s	s
(q,t)	0	$Space_{qt}$
(t,T)	s	s
(T,S)	Ns	Ns

Table 6.11. Lower and upper bounds required for a feasible flow in the network.

Figure 6.12 shows the setup of this network flow model.

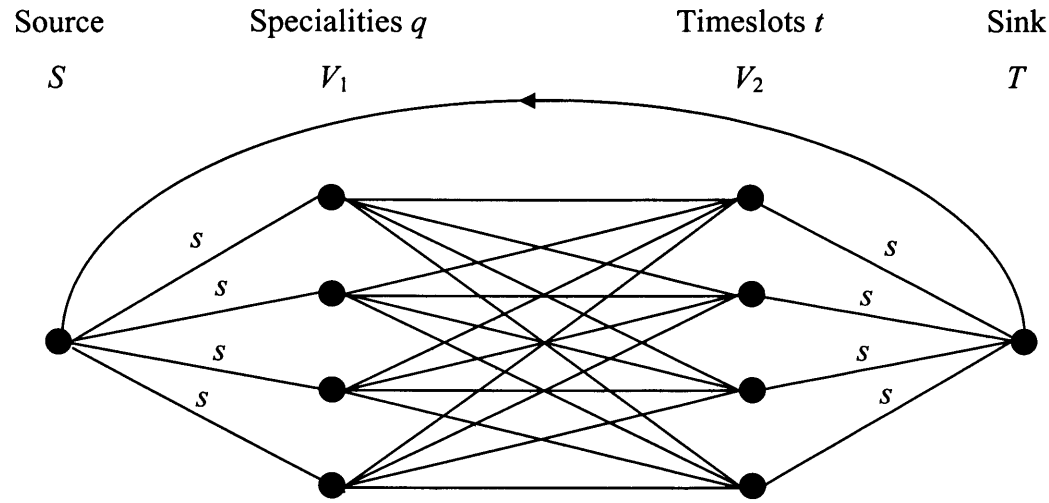


Figure 6.12. Network flow model for which a feasible flow designates a feasible allocation of specialities to timeslots which scheduling student by student.

Then if we can find a feasible flow in this network, we have found a suitable allocation of specialities to timeslots. The network flow model would not need to be set up for each student, but may be initialised at the start of the schedule with $s = r$ and the lower bounds increased as each student is scheduled.

This could then be used in conjunction with any other solution approach to check that an appropriate balance of each speciality is being maintained within each timeslot in the same way as the network flow model for checking the schedules timeslot by timeslot and the knapsack model was used to assess the feasibility of the nurses' assignments. Note that this only guarantees feasibility of the remaining students' full permutations, but would not avoid situations such as that in *Example 1* in Figure 6.9. To ensure that the current student being assigned may complete their allocation, it would be necessary to cycle through the permutations for the remaining timeslots and temporarily impose these on the network flow model until an assignment is found which will allow a feasible complete of the student's schedule.

Note that the complexity of the feasibility check using the student by student construction is also exponential relative to problem size, with up to $N!$ calls to the network flow

subroutine are required for each student. Since both scheduling approaches, student by student and timeslot by timeslot require feasibility checks whose time requirements are exponential with respect to N, there is no reason to prefer the student by student approach; the timeslot by timeslot approach will therefore provide the basis for all future investigation.

6.2.3 Local Search Neighbourhoods

Having examined the construction phase of the GRASP algorithm, we now give details of the improvement phase. This section explains the neighbourhoods which we will employ and the structure of the landscape we aim to explore.

The solution space considered in the improvement phase is dependant on the construction. If no network flow model is used to guarantee feasible solutions to use as a starting point for the local search, we must allow some infeasible solutions to be considered. However, if the network flow algorithm has been implemented, we may deduce all our solutions are already feasible and thus there is no reason to include infeasible timetables in the search space.

When the timetable is not yet feasible, a neighbouring solution is accepted as being an improvement on the current one if it lowers the feasibility cost or if the feasibility cost is unchanged, but the sum of the optimality costs is lowered.

When choosing suitable neighbourhoods for the improvement phase for nurse scheduling, we concerned ourselves only with changing whole patterns for a nurse; at no point did we change just a single shift. In the case of medical student scheduling, however, we are unable to use this technique. It would be too computationally expensive to search the whole neighbourhood based on selecting, for each student, a 'pattern' detailing which five firms they will attend in each timeslot, due to the number of such combinations which must be considered. By avoiding such an approach, we also increase the number of possible neighbourhoods; by dealing with each student, firm and timeslot individually,

there are many neighbourhood definitions available and we must ensure that the neighbourhoods we select are able to fully explore the landscape in the most efficient manner possible.

We investigate three possible neighbourhoods; two different 1-opt neighbourhoods and a swap neighbourhood. These are defined as follows.

6.2.3.1 Change Neighbourhood

$M_C(\sigma)$ is the set of solutions obtained from solution σ by changing one student's assignment in one timeslot. Only timeslots 2-5 will be considered; since students are homogenous and timeslot 1 is tight, changing all students' options in timeslots 2-5 would be equivalent to changing their timeslot 1 allocation. Note that if we only change the firm in one timeslot, we must ensure that the new firm we select belongs to the same speciality as the original firm in order to avoid the student studying the same speciality twice.

To achieve this, we select a random student s and select for that student a random pair (t,f), such that firm f covers the same speciality the student is currently timetabled to study in timeslot t. That is

$$a_{fq_0} = a_{f_0q_0}, \tag{6.40}$$

where f_0 is the firm which student s is currently assigned to in timeslot t and f_0 belongs to speciality q_0. If $f \neq f_0$, we check whether the change would result in an improvement, and accept it if this is the case.

Note that when selecting a student and firm, several checks must be performed. Not only must the new firm be different from the old firm, but the new firm must cover the same speciality as the old firm and have space remaining in order for the move to have a chance of being accepted. Thus, by allowing all choices of firm for each student, there is the chance that many unsuitable firms may be tested before one which meets these basic criteria is found. Note that when implementing this type of 1-opt neighbourhood for the nurses, it was practical to create a standard list of patterns for each nurse which could be

sampled from throughout the process, since the potentially successful patterns for each nurse were not so variable or so numerous. However, for the medical student scheduling problem, this would be a very time-consuming approach. Note that use of the permutation neighbourhood, described below, means that for each student, the speciality studied in a timeslot may change and so the list of potential firms for this neighbourhood will change also. In order to speed up the selection process, the local search creates an initial list of feasible triplets (s,f,t) which pass these checks, where s is a student and f is a firms which this student could potentially work in timeslot t. Then, every time a neighbourhood move is accepted, this list is updated to reflect the changes, removing triplets from the list pointing to firms which are now full, adding firms which have had places created and exchanging the timeslots in triplets where a student's permutation of specialities has changed. By maintaining the list in this manner, the change neighbourhood can easily sample from it and thus will automatically test an allocation which has the necessary basic requirements, speeding up the process overall.

Changing a student's firm gives us a large neighbourhood; each solution has in the order of vr neighbours. Note that without allowing students to change speciality in a given timeslot, we would not be able to reach all possible solutions using this neighbourhood alone, even allowing for symmetry. Another concern is that once a feasible solution is found, if the dataset is tight, there will be no feasible moves in this neighbourhood. Given this, we introduce a new neighbourhood to address some of these issues.

6.2.3.2 Permutation neighbourhood

$M_P(\sigma)$ is the set of solutions obtained from a solution σ by giving one student a new permutation of specialities for timeslots 2-5 and then allocating them a new set of firms accordingly. By allowing students to move between firms as well as changing their speciality, we increase the number of feasible moves available using this neighbourhood.

To achieve this, we select a student s along with a random permutation of specialities 2-5, (q_2,q_3,q_4,q_5). A check is performed to ensure the permutation differs from that of the

student's current timetable and that each speciality has at least one firm with places available in the given timeslot. That is, (6.41) and (6.42) must be satisfied, where q_t represents the speciality associated with timeslot t in the given permutation.

$$\sum_{t=2}^{5}\sum_{f=1}^{v} a_{fq_t} x_{sft} < 4 \tag{6.41}$$

$$\sum_{f=1}^{v} a_{fq_t} \cdot \max\left(\sum_{s=1}^{r} x_{sft} - C_f, 0\right) \geq 1 \quad \forall t \tag{6.42}$$

We cycle through each combination of 4 firms within this permutation, $\{(f_2, f_3, f_4, f_5) : a_{f_i q_i} = 1\}$, calculating the new cost these firms would provide, and at each stage storing the best combination found so far. Here, the 'best' is recognised by the same criteria by which we judge an improving move. We then accept this best combination if it is an improvement on the current schedule. Note that in timeslots where the speciality in the randomly generated permutation does not differ from the speciality the student is already assigned to, movement between firms is still allowed. By disallowing such moves we may miss out on potential improvements to our timetable and in cases where we do not force a feasible construction, we may even improve feasibility with such moves. However, if the randomly generated permutation does not differ in any timeslot from that already assigned to the student, no moves will be considered as this would be equivalent to a number of moves in $M_C(\sigma)$.

Obviously calculating costs for every firm combination is computationally expensive, so we reduce the work needing to be done by intelligently removing combinations which cannot result in an improvement. By ignoring all combinations involving a firm which is already full, we can drastically reduce the work needed to find an improving move.

This is a large neighbourhood with a size in the order of $v^4 r$ as we select the best set of firms or $v^N r$ in the general case and, in conjunction with the *Change* neighbourhood, it theoretically covers the whole landscape of solutions. However, when faced with a tight dataset, it is possible that no feasible moves will be available in this neighbourhood when the current timetable is feasible or close to feasible. We therefore introduce a third neighbourhood, which will have feasible moves in this situation.

6.2.3.3 Swap neighbourhood

$M_S(\sigma)$ is the set of solutions obtained from a solution σ by swapping the firms of two students in some or all of timeslots 2-5, in such a way that the feasibility of the schedule is not compromised. Note that since students are homogeneous before they are assigned, there is little point in allowing all timeslots to be swapped, since the same solution can be reached by a serious of moves in $M_S(\sigma)$.

We select two students s_1 and s_2 and perform two quick checks: first, that $s_1 \neq s_2$ and secondly, that the part of the cost function ζ contributed by these two students is not equal to the cost from them being paired together. That is, we only select students s_1 and s_2 if we have:

$$\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{f=1}^{v}\sum_{g=1}^{v}\left(c_{fg}+h_{fg}\right)\left(x_{s_1ft}x_{s_1gu}+x_{s_2ft}x_{s_2gu}\right)+$$

$$\sum_{t=1}^{4}\sum_{u=t+1}^{5}\sum_{\substack{z=1\\ z\neq s_1\\ z\neq s_2}}^{r}\sum_{f=1}^{v}\sum_{g=1}^{v}\left(x_{s_1ft}x_{zft}x_{s_1gu}x_{zgu}+x_{s_2ft}x_{zft}x_{s_2gu}x_{zgu}\right)>0 \qquad (6.43)$$

Obviously if their cost contributions are just their mutual pair costs, swapping their firms will not affect the overall cost of the solution and we may discard this selection without further investigation.

We then calculate the possible sets of timeslots whose firms may be swapped. From the set of timeslots {2,3,4,5} it will not usually be feasible to swap all possible subsets. For example, if the two students study different specialities in a particular timeslot, swapping the firms in this timeslot alone would leave each of the students' timetables with one duplicated speciality and one incomplete. We therefore only consider swapping subsets which would not interfere with the feasibility of the final schedule. From these possible subsets, we keep the best of these if it results in an improvement. Again, the 'best' is defined by the same criteria by which we judge an improving move. By only considering swapping sets of firms which will result in a feasible schedule for each student, the time

required to check for an improving move in this neighbourhood is reduced; although students with the same permutations of specialities will require the complete check of all 15 subsets ($15 = {}^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4$), students whose specialities differ in every timeslot, for example, will require all firms to be swapped in order for their schedules to be feasible and so only this one check needs to be performed.

$M_S(\sigma)$, then is defined as the set of solutions which may be obtained from a solution σ, by swapping a feasible subset of firms between two students. The number of neighbouring solutions from each solution, σ, is of order r^2.

This neighbourhood does not allow for new firms to be introduced into the solution. However, in tight datasets, this neighbourhood will always have feasible moves and so is very useful in combination with the first two neighbourhoods to fully explore the search space.

At each iteration, the local search chooses randomly between the change and permutation neighbourhoods until the solution becomes feasible; once a feasible solution is obtained, all three neighbourhoods have an equal chance of being selected. Note that for all three neighbourhoods, sampling is without replacement and the check for the new solution cost consists only of altering the existing cost with regards to the suggested move, rather than recalculating the entire cost of the schedule.

6.3 Data

We were supplied with two sets of data; one from the academic year starting in 2005 and one from a previous year. The smaller of these had 190 students and 69 firms and the larger had 280 students and 112 firms.

Each dataset came in the following format:

E1. A list of the capacities for each firm.

E2. A list of which hospital each firm belongs to.

E3. A list of which speciality each firm offers.

E4. A list of which consultants are associated with each firm.

E5. The number of students in the year.

Although we were able to experiment with just the two real datasets supplied, we wish our algorithm to be as robust as possible and so, to that end, we created several more datasets on which to base our results. The aim was to create datasets which were realistic, based on the information given, and which would vary slightly each of the different variables in a way which is plausible in the real world.

Given that our nurse scheduling parameter choices were based on analysis of results from 52 datasets, we ideally wish to experiment on a similar number of datasets when tackling this problem. By experimenting on only two datasets we run the risk of producing a data-specific algorithm, capable of solving these two datasets, but unable to cope well with any further medical student scheduling problems.

By modifying each of the two real datasets in a number of different ways as explained later in this section, we were able to construct a further 46 datasets, giving us a total of 48 with which to experiment. With this larger number of datasets we will be able to deduce more effectively how successful our algorithms are at solving the medical student scheduling problem; any algorithm now tested will have to be robust in order to successfully solve a wider range of data.

As mentioned previously, the real data supplied came in a format which split the relevant information into 5 sections: firm capacities (E1), firm-hospital grouping (E2), firm-speciality grouping (E3), consultant lists for each firm (E4) and number of students to be scheduled (E5). We decided to alter three of these five in order to create the new datasets. Table 6.13 provides details of the data for E1, E2 and E3 for the two original datasets as well as the modified versions of E1 and E2 which were used to create some of the new datasets. The key provides an explanation of each of the columns. We now give details of how we created the 46 new datasets.

Firm	E1	E1'	E2	E2'	E3
1	5	4	1	1	1
2	5	6	1	1	1
3	5	4	1	1	1
4	5	6	1	1	1
5	5	4	1	1	1
6	5	6	1	1	1
7	5	4	1	1	1
8	5	6	1	1	1
9	5	4	1	1	1
10	5	6	1	1	1
11	5	4	1	1	1
12	5	6	2	2	1
13	5	4	2	2	1
14	5	6	2	2	1
15	5	4	2	2	1
16	5	6	2	2	1
17	5	4	2	2	1
18	5	6	3	3	1
19	5	4	3	3	1
20	5	6	3	3	1
21	5	4	3	3	1
22	5	6	3	3	1
23	5	4	3	3	1
24	5	6	3	3	1
25	5	5	4	4	1
26	5	5	4	4	1
27	5	5	4	4	1
28	5	5	4	4	1
29	5	5	4	4	1
30	5	5	5	5	1
31	5	5	5	5	1
32	5	5	5	5	1
33	5	5	5	5	1
34	5	5	6	1	1
35	5	5	6	1	1
36	5	5	7	6	1
37	5	5	7	6	1
38	5	5	7	6	1
39	4	4	1	1	2
40	4	4	1	1	2
41	4	4	2	7	2
42	4	5	2	2	2
43	4	5	2	2	2
44	8	6	3	3	2
45	8	6	3	3	2
46	8	8	5	4	2
47	8	10	4	5	2
48	8	4	2	2	3
49	8	4	4	5	3
50	8	6	1	1	3
51	8	10	1	1	3
52	8	12	3	3	3
53	8	12	5	4	3
54	4	2	1	1	4
55	4	2	1	1	4
56	4	2	1	1	4
57	4	2	1	2	4
58	4	4	2	2	4
59	4	4	2	3	4
60	4	4	3	3	4
61	4	4	3	3	4
62	4	4	5	5	4
63	4	6	5	5	4
64	4	6	4	4	4
65	4	8	4	4	4
66	12	10	8	5	5
67	12	12	9	6	5
68	12	12	10	7	5
69	12	14	11	8	5

(a) Dataset 1: 190 students

Firm	E1	E1'	E2	E2'	E3
1	5	6	1	1	1
2	5	4	1	1	1
3	5	6	1	1	1
4	5	4	1	1	1
5	5	6	1	1	1
6	5	4	1	1	1
7	5	6	1	1	1
8	5	4	1	1	1
9	5	6	1	1	1
10	5	4	1	1	1
11	5	6	1	1	1
12	5	4	1	1	1
13	5	6	1	1	1
14	5	4	1	1	1
15	5	6	2	2	1
16	5	4	2	2	1
17	5	6	2	2	1
18	5	4	2	2	1
19	5	6	2	2	1
20	5	4	2	2	1
21	5	6	2	2	1
22	5	4	2	2	1
23	5	6	2	2	1
24	5	4	2	2	1
25	5	6	2	2	1
26	5	4	2	2	1
27	5	6	3	3	1
28	5	4	3	3	1
29	5	6	3	3	1
30	5	4	3	3	1
31	5	6	3	3	1
32	5	4	3	3	1
33	5	6	3	3	1
34	5	4	3	3	1
35	5	6	3	3	1
36	5	4	4	4	1
37	5	6	4	4	1
38	5	4	4	4	1
39	5	6	4	4	1
40	5	4	4	4	1
41	5	6	4	4	1
42	5	4	4	4	1
43	5	6	4	4	1
44	5	4	5	4	1
45	5	6	5	5	1
46	5	4	5	5	1
47	5	6	5	5	1
48	5	4	5	5	1
49	5	6	5	5	1
50	5	4	5	5	1
51	5	6	6	6	1
52	5	4	6	6	1
53	5	6	6	6	1
54	5	4	6	6	1
55	5	4	6	6	1
56	5	6	7	6	1

Firm	E1	E1'	E2	E2'	E3
57	5	8	1	1	2
58	5	3	1	1	2
59	5	5	1	1	2
60	5	4	1	1	2
61	5	4	1	1	2
62	5	4	1	1	2
63	5	6	1	1	2
64	5	6	5	2	2
65	5	6	5	3	2
66	5	4	5	3	2
67	5	2	5	5	2
68	5	7	6	5	2
69	5	5	6	5	2
70	5	6	6	5	2
71	5	8	2	2	3
72	5	3	2	2	3
73	5	6	2	2	3
74	5	4	2	2	3
75	5	4	2	2	3
76	5	4	2	2	3
77	5	5	3	3	3
78	5	5	3	3	3
79	5	6	3	3	3
80	5	4	3	4	3
81	5	2	4	4	3
82	5	7	4	4	3
83	5	6	4	1	3
84	5	6	7	1	3
85	5	8	1	1	4
86	5	3	1	1	4
87	5	5	1	1	4
88	5	4	1	1	4
89	5	4	2	2	4
90	5	4	2	2	4
91	5	5	2	2	4
92	5	6	3	3	4
93	5	6	3	3	4
94	5	2	3	3	4
95	5	4	4	4	4
96	5	7	4	4	4
97	5	6	4	5	4
98	5	6	6	5	4
99	5	8	1	1	5
100	5	3	1	1	5
101	5	6	1	1	5
102	5	4	2	2	5
103	5	6	2	2	5
104	5	4	2	2	5
105	5	5	3	2	5
106	5	5	3	3	5
107	5	6	4	3	5
108	5	4	4	3	5
109	5	6	5	5	5
110	5	4	5	5	5
111	5	2	5	5	5
112	5	7	6	5	5

(b) Dataset 2: 280 students

KEY to Table 6.13
E1 : Original capacity
E1' : Modified capacity
E2 : Original hospital
E2' : Modified hospital
E3 : Speciality

Tables 6.13 (a) and (b) Details of datasets 1 and 2, respectively.

6.3.1 E1 – Firm capacity

It is conceivable that firm capacities may be variable from year to year. In order to ensure that our algorithm is able to cope with such variations, we now explain how the original data was used to provide new data in which the firm capacities are altered.

Dataset 1 has 190 students to be scheduled and dataset 2 has 280 students. Referring to Tables 6.13, we see that in each case we have space for all students to do the introductory module in the first timeslot, with no places spare. The second dataset also has no slack in any of the further specialities; in each timeslot there will be the correct number of spaces to ensure each student may study one of the four outstanding specialities in rotation in each of the four remaining timeslots with no places unfilled. In the first dataset, however, there are more places in each of the four specialities in each timeslot than are necessary. Obviously the 190 students cannot divide equally into the four specialities in each timeslot as this would result in the practical impossibility of 47.5 students in each group. However, we can see that since we have 48 places in three specialities and 52 in the remaining one, there is some slack here; we may assign all the students and still have some firms filled to less than maximum capacity in some timeslots.

When altering the capacities of the firms, we allowed the total capacity of each speciality to remain constant. The new capacities for the firms were not generated randomly, but were chosen to create diversity without unduly unbalancing the total capacity of the firms in each hospital excessively. This process resulted in one new list of capacities for each of the two datasets.

Table 6.13 shows the original and modified capacities for each of the firms in columns E1 and E1', respectively.

6.3.2 E2 – Hospitals

To give us a broader range of data, we altered the number and types of firms in each hospital, merging some hospitals in the process. By merging hospitals, and broadening the number of specialities offered by each, we increase the difficulty of the problem by making hospital clashes harder to avoid. It is certainly within the realms of possibility that the number of hospitals in which students can do placements may decrease. While the reverse is true, we did not think this a sensible option to investigate, since more available hospitals would reduce the likelihood of hospital clash costs occurring. The possibility that a hospital may offer new firms and new specialities is also conceivable which led us to explore this option.

Again, the changes made were not generated randomly; we carefully selected the new hospitals' structure so that the resulting compositions were realistic considering the current setup. Overall, we were left with a reduced number of hospitals in both cases, with at least one hospital offering all five specialities. This resulted in one new list of hospitals for each of the two datasets. These changes are detailed in Table 6.13, with columns E2 detailing the original data and E2' detailing the modified data.

6.3.3 E3 – Specialities

It was decided that we would not alter the speciality with which each firm was associated. It seemed unlikely that a firm would switch from one speciality to another in a real-world situation, since all aspects of the firm, such as consultant and equipment would be affected by the change. Furthermore, all students have to cover each speciality once and so the ratio of the numbers of each type of firm is unlikely to change much, year by year. Varying all the other aspects of the problem definition individually, a more likely scenario, is enough to ensure a diversification in problem specification and so exploring this eventuality may be considered redundant. We therefore leave the firm-speciality pairings unchanged. The specialities for each firm can be seen in Table 6.13, in columns E3.

6.3.4 E4 – Consultants

The list of which consultants worked with each firm was used to generate the score, c_{fg}, for each pair of firms, (f,g). We calculated a value for 'clash density', for each of the two datasets, where we define the clash density, ρ, to be the probability that a randomly selected pair of firms (f,g) has $c_{fg} = 1$. The original data gave $\rho \approx 0.03538$ for dataset 1 and $\rho \approx 0.01802$ for dataset 2.

To create variation, we assigned a c_{fg} value of '1' to each element if RAND[0,1) $< 1.25\rho$, where RAND[0,1) was a randomly generated number in the range [0,1). This way we had a new, random consultant clash list for each of the two datasets, with a clash density of approximately 25% greater than the original.

The main observation with the new lists is that while, originally, consultants would only clash between firms if they belonged to the same hospitals, now there may be consultant clashes between hospitals, thus making the problem more difficult as avoiding consultant clashes must now balance avoiding hospital clashes. Rather than the hospital and consultant constraints reinforcing one another, they are now conflicting and so we have created datasets which, essentially, have three conflicting sets of constraints rather than two.

6.3.5 E5 – Number of students

There is the option to vary the number of students in a given year. Decreasing the number of students will increase slack and potentially reduce problem difficulty although does allow for more variation in the solutions. As outlined in Section 6.2.2 it is the way in which the spare capacity is used which determines the feasibility. Increasing the number of students, given that the introductory module, for example, provides exactly the required number of places, would result in all solutions being infeasible, unless more places were created.

We decided not to alter the number of students in this way, since the problem would not be made more interesting by a variation of this sort.

6.3.6 Compiling the new datasets

With the new data elements, we created eight datasets from each of the two original datasets. Table 6.14 shows the characteristics of each of these eight

Capacities (E1)	Hospitals (E2)	Consultants (E4)
Original	Original	Original
Modified	Original	Original
Original	Modified	Original
Modified	Modified	Original
Original	Original	Modified
Modified	Original	Modified
Original	Modified	Modified
Modified	Modified	Modified

Table 6.14. Characteristics of eight datasets created from each original dataset.

Applying these variations to both of the original datasets, we now have a total of sixteen datasets – eight based on the first dataset and eight on the second.

As mentioned earlier, the first of the original datasets was slightly slack in terms of firm capacity. It is known that there are 190 students, all studying the first speciality in time period 1. Since the total capacity of all firms offering this module is 190, this part of the problem is already tight. Since the other four specialities are to be studied simultaneously during time periods 2-5, for this part of the problem to be tight, exactly a quarter of the students must study each of the four specialities at any time. With three of the specialities having 48 places and the fourth 52 places available, all the four remaining specialities have slack. We can tighten the problem by adding two students and adding two more spaces to the total capacity of the firms offering speciality 1. In this way, the first timeslot remains tight and the three specialities with capacity 48 become tight. Only the speciality with total capacity 52 contains an element of slack. By making this modification for each

of the eight datasets based on this dataset, we now have sixteen datasets based on the first dataset.

To complete the tightening of this data we then remove four spaces from the remaining slack speciality. Applying this to each of the eight modified datasets, we now have 24 datasets, with eight of these being completely tight, eight with some slack and eight slacker still. Let us introduce the notation $m_\lambda(\delta)$ to represent a dataset with λ modifications and original dataset δ, We therefore now have 24 datasets comprising eight of the form $m_0(1)$, eight $m_1(1)$ and eight $m_2(1)$.

Since dataset 2 is already tight, we cannot tighten it further, but we may make similar modifications in the backwards direction. We already have eight datasets of the form $m_0(2)$ and we can remove two students and two spaces from the speciality 1 firms to create some slack in time periods 2-5 to create eight slightly slack datasets, designated $m_1(2)$. To obtain $m_2(2)$, we add more places to one of the specialities 2-5 to make more slack in the problem, giving a similar situation as the original dataset 1. Thus we have 24 datasets based on this dataset alone and 48 datasets in total. This gives us a broader range of realistic data to experiment with.

6.4 Experiments and results

As described in Section 6.3, we have 48 sets of data, each representing one timetabling problem. All datasets, including, to our knowledge, the two originals, have not previously been solved using any optimisation technique and we therefore have no benchmark solutions on which to base the success of our results.

Rather than comparing the results obtained with the optimal values as was possible for the nurse scheduling results in Chapter 4, we can only compare the results produced from different approaches and surmise that those approaches creating solutions of lower cost must be the more successful. However, we may also put forward the idea that methods

which produce solution costs with a high standard deviation are not robust and are certainly not producing the optimal solution in every run.

With this in mind, the following sections detail the particular experiments undertaken and their comparative success.

6.4.1 Initial experiments

Parameter testing

We have suggested four heuristics: *Feasibility*, *Combined*, *Holistic* and *LastChance*. For each of these the values of the parameters n, w_f and w_o are varied. As for the nurse scheduling problem, we selected four potentially suitable values of n for each heuristic. These values, along with those tested for w_f and w_o, are shown in Table 6.16.

Heuristic	n	w_f	w_o
Feasibility	3, 6, 10, 15	—	—
Combined	3, 6, 10, 15	1 – 5	1 – 5
Holistic	3, 10, 60, 100	1 – 5	1 – 5
LastChance	3, 6, 10, 50	1 – 5	1 – 5

Table 6.16. The parameter values investigated for each heuristic.

For each of these parameter and heuristic combinations, 100 GRASP cycles will be performed. If we consider these 100 cycles to constitute one run of the algorithm, we are therefore producing 304 runs, each with a different parameter combination.

Neighbourhood testing

Together, the *Change* and *Permutation* neighbourhoods theoretically cover the entire landscape of solutions. However, it has been noted that once a feasible solution is encountered, there may not be any feasible moves in either neighbour if only improving moves are permitted. The *Swap* neighbourhood gives further movement within the

landscape while maintaining feasibility. However, since the inclusion of the more versatile *Swap* neighbourhood will be likely to increase the total time taken for each GRASP cycle, initial tests will be performed to determine its usefulness.

Two local search approaches will therefore be utilised for each parameter combination: the first will use only the Change and Permutation neighbourhoods, while the second will include the Swap neighbourhood as well. Thus, 608 runs will be performed, half of these with two neighbourhoods and half with all three.

The Network Flow model

A further enhancement to the algorithm, suggested in Section 6.2.2.2, is the use of a network flow algorithm to ensure all solutions are feasible at the end of the construction phase. For the nurse scheduling problem in Chapter 4, it was found that without added help directing the construction towards feasibility, schedules were unlikely to be feasible even after the implementation of the local search. Experiments will be done here to test whether this is also the case for the medical student scheduling problem. If the algorithm struggles to find feasible solutions, then the addition of the network flow model may play an important part in creating good solutions. However, if finding feasible solutions is not an issue, it may save time and allow the algorithm to find lower cost solutions, by excluding such an enhancement.

Each variant of the algorithm suggested so far will be tested both with and without the network flow model. Thus we have 1216 variations to be tested on the 48 datasets. We only perform one run on dataset with each of the variants at this stage since it would be too computationally expensive to do more than this. Since one run comprises 100 independent GRASP cycles, this gives us enough information to choose parameters with which to continue experimentation.

6.4.2 Initial results

As was mentioned in the introduction to this section, we have no way to measure the actual success of these methods, since the optimal solutions are not known. Since we are not able to compare the solution costs obtained with the optimal, as we were for the nurse scheduling problem, we instead compare the different approaches and decide which of these is the most successful, even though we are not able to judge this success in definitive terms.

Normalising the results

For each dataset and each variant, we have produced 100 schedules. Unlike the nurse scheduling problem, where the algorithm struggled to find feasible solutions for all datasets, all algorithms tested on the medical student scheduling problem found at least 19 feasible solutions from 100 for every dataset. Furthermore, three quarters of the datasets yielded feasible solutions at least 70% of the time for all algorithms and so, unlike for the nurse scheduling problem, we will analyse the success of each algorithm based purely on the costs obtained relating to the soft constraints and not on its ability to find feasible solutions. There are consequently two main ways to judge the success of such a run: either the best cost obtained in the run or the average of the feasible costs obtained. There are benefits and drawbacks associated with both approaches. If we were to use the algorithm to create a schedule in the real world, one run would be performed and the output would certainly be the best cost solution. Thus, this approach is more realistic as it is really only the best solution which is of interest in the long term. Using the average of all feasible solutions would include information from poorer schedules which would never actually be selected for use. However, given that we are performing preliminary experiments and producing only one run at this stage, it is not sensible to rely solely on one solution for each method; the best solution for a particular method may be an outlier, unlikely to be repeated in subsequent runs and exaggerating its success. Using the averages, however, it is possible to compare two methods knowing that similar results would be likely if the experiments were repeated.

Given this deliberation, we elect to use both methods for the following analysis, and use only the best costs when we are comparing more than one run. So, for each dataset and each of the approaches, we have two values: the best feasible cost obtained from the 100 cycles and the average cost of all feasible solutions. The simplest way to compare the results, since the costs vary greatly between datasets, is to normalise the results across the datasets. For both average and best costs, we normalise the data so that variations in particular datasets will not skew the perceived success of any method.

To normalise the data, we calculate the mean and the standard deviation of the results for each dataset across all methods. Here, 'results' refers to either the best or average costs obtained. Each cost, $\chi_{\mu\delta}$, associated with a method μ and a dataset δ is then normalised. The normalised cost, $\hat{\chi}_{\mu\delta}$, is calculated using

$$\hat{\chi}_{\mu\delta} = \frac{\chi_{\mu\delta} - \overline{\chi}_{\delta}}{\sigma_{\delta}}, \tag{6.44}$$

where $\overline{\chi}_{\delta}$ is the mean of all costs pertaining to dataset δ and σ_{δ} is their standard deviation. This means that for each dataset the normalised costs have a mean of zero and a standard deviation of 1 and the methods may be compared fairly by simply summing these normalised costs across all datasets.

Neighbourhood testing

Once the initial results had been normalised, there was an obvious conclusion to be drawn regarding the use of the different neighbourhoods. Figure 6.17 shows the results obtained with and without the *Swap* neighbourhood using the best solution from each cycle.

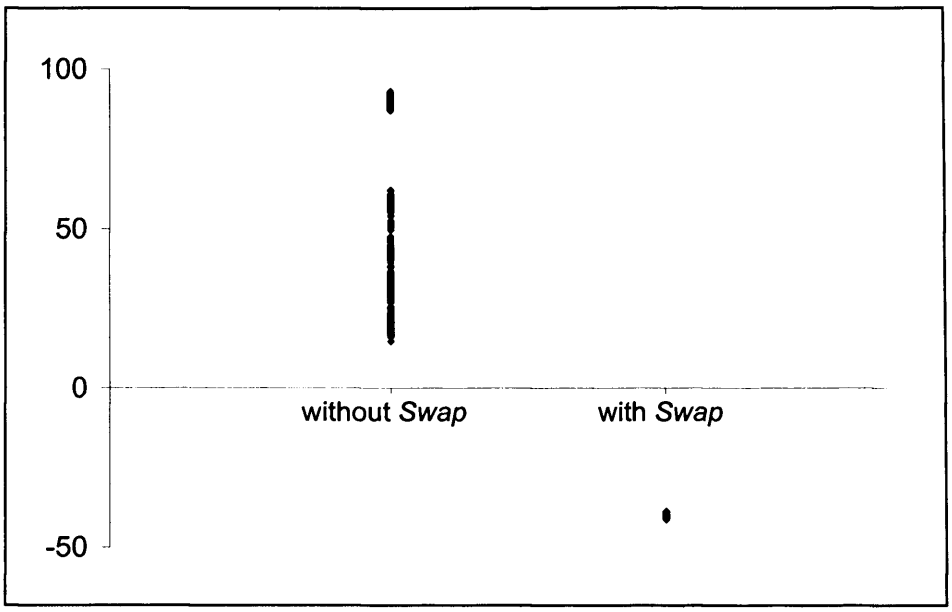


Figure 6.17. Graph of normalised costs with and without the *Swap* neighbourhood, summed across all datasets.

Each point in Figure 6.17 represents the total normalised best costs across all datasets for one of the parameter combinations. The same graph produced using the average costs shows a very similar pattern and it is clear that with the *Swap* neighbourhood, the algorithm produces results of a far superior quality. From this point onwards, all experimentation carried out will include the *Swap* neighbourhood.

From Figure 6.17, we can see that the difference between the variants which do include the *Swap* neighbourhood is negligible in comparison with the difference in cost of including or not including it. In order that the array of poor quality solutions arising from these discarded experiments will not skew further interpretation of the results, we renormalise the data at this point, ignoring these unwanted results.

Parameter testing

Once the results were normalised, there were no further obviously superior variants, and so values for the parameters n, w_f and w_o were selected for each heuristic, with and without the addition of the network flow model. In many cases, there was no one choice

which clearly outperformed the others, but this very lack of distinction meant that as long as sensible choices were made, the algorithm should not be much affected by the selection. The selected parameters are shown in Table 6.18.

	w/o network flow			with network flow		
	w_f	w_o	n	w_f	w_o	n
Feasibility	–	–	10	–	–	10
Combined	2	1	10	1	2	10
Holistic	3	3	10	2	5	10
LastChance	1	5	3	1	3	3

Table 6.18. The parameter values selected for each heuristic.

6.4.3 Further experiments and results

For each of the parameter combinations shown in Table 6.18, a further ten runs were performed, with one run consisting of 100 GRASP cycles and the results of these runs were normalised to provide a fair comparison. Now that ten runs are being performed on each variant, the best solution attained in each run becomes of more interest as we will have ten such best results for each. Whereas previously, where only one run was performed, the best solution could potentially be an outlier and misleading as to the success of the algorithm in general; when ten bests are produced for each, the overall comparison of such values is more likely to be a more accurate reflection of the algorithms' respective success. However, the average results will always provide a reasonably robust comparative technique. Other considerations for each approach's success are the number of feasible solutions obtained and the time taken for each complete run. Tables 6.19-6.22 detail these results for our analysis.

Heuristic	*Feas.*	*Comb.*	*Holis.*	*Last C.*	*Feas.*	*Comb.*	*Holis.*	*Last C.*
Net. flow?	No	No	No	No	Yes	Yes	Yes	Yes
	4.04	7.46	-9.06	-5.75	7.16	12.77	-13.86	-8.57
	-1.02	2.28	-11.47	-3.15	9.42	3.08	-6.93	-6.10
	9.83	3.90	-5.11	0.69	10.78	13.39	-19.38	-15.83
	11.88	11.12	-14.30	-3.89	-0.57	0.64	-8.09	-6.41
	9.68	1.45	-5.96	3.72	15.77	2.23	-4.98	-10.93
	9.63	-8.26	-6.11	-10.31	5.09	10.09	-7.97	-5.78
	12.51	0.23	-2.66	-10.37	10.31	13.47	-6.29	-4.36
	2.59	2.08	-5.84	-18.90	5.66	2.20	-3.45	-3.92
	4.32	8.94	-0.18	-2.04	7.23	5.67	-17.24	-9.91
	15.58	9.54	5.45	4.26	17.90	6.01	-1.58	-13.53
Mean	7.90	3.88	-5.52	-4.57	8.87	6.96	-8.98	-8.53

Table 6.19. The normalised best results for each heuristic over ten runs and all datasets.

Heuristic	*Feas.*	*Comb.*	*Holis.*	*Last C.*	*Feas.*	*Comb.*	*Holis.*	*Last C.*
Net. flow?	No	No	No	No	Yes	Yes	Yes	Yes
	-1.46	6.78	-25.18	-12.59	11.02	28.03	-0.35	-0.61
	1.55	-2.75	-12.10	-18.36	13.43	29.64	-10.13	-5.44
	-2.26	-3.57	-15.23	-13.99	8.28	21.11	-16.27	-11.26
	3.33	-0.93	-10.71	-5.02	9.13	27.57	-7.70	-3.44
	3.24	-5.22	-24.99	-6.27	15.80	19.65	-3.45	-10.44
	5.87	10.28	-4.97	-16.95	22.34	10.48	-12.25	-0.95
	-2.51	-5.00	-17.06	-8.06	23.26	21.51	-5.10	-0.38
	-6.49	9.96	-9.04	-12.05	12.29	24.05	-11.79	-3.42
	4.40	-0.33	-14.15	-11.47	23.64	26.25	-5.31	-3.81
	0.62	2.33	-16.20	-15.73	11.57	25.11	-1.23	-18.58
Mean	0.63	1.16	-14.96	-12.05	15.08	23.34	-7.36	-5.83

Table 6.20. The normalised average results for each heuristic over ten runs and all datasets.

Heuristic	*Feasibility*			*Combined*			*Holistic*			*Last Chance*		
	Max.	Min.	Ave.	Max.	Min.	Ave.	Max.	Min.	Ave.	Max.	Min.	Ave.
	100	95	99.42	100	72	93.69	100	33	89.10	100	26	89.19
	100	92	99.44	100	73	94.40	100	50	89.71	100	22	89.29
	100	96	99.46	100	73	93.94	100	37	89.42	100	33	88.46
	100	94	99.44	100	70	94.25	100	44	89.83	100	33	89.29
	100	95	99.42	100	73	94.17	100	48	89.54	100	36	89.17
	100	95	99.56	100	67	93.60	100	40	88.71	100	25	89.08
	100	94	99.50	100	74	94.17	100	47	89.69	100	28	89.71
	100	94	99.44	100	71	93.67	100	45	89.29	100	24	88.96
	100	94	99.42	100	70	93.94	100	44	89.63	100	20	88.54
	100	96	99.52	100	72	94.13	100	42	89.46	100	21	89.19

Table 6.21. The maximum, minimum and average number of feasible solutions from 100 over all datasets for each run for approaches without the network flow model.

Heuristic	*Feas.*	*Comb.*	*Holis.*	*Last C.*	*Feas.*	*Comb.*	*Holis.*	*Last C.*
Net. flow?	No	No	No	No	Yes	Yes	Yes	Yes
	-113.03	-345.83	205.12	99.96	-72.36	-231.27	289.06	178.34
	-111.22	-269.89	171.03	82.51	-60.36	-187.36	232.84	145.00
	-107.10	-276.64	170.06	78.34	-61.58	-188.63	234.94	143.88
	-106.04	-275.36	171.15	83.22	-60.69	-188.23	232.45	143.60
	-110.47	-274.17	170.48	83.93	-59.25	-186.95	230.50	144.85
	-99.55	-278.05	163.81	81.11	-53.76	-188.72	233.11	144.94
	-107.11	-274.21	170.42	84.15	-54.81	-188.90	233.22	146.59
	-109.23	-279.20	169.42	81.99	-59.75	-186.90	229.97	144.23
	-110.41	-276.34	170.73	82.86	-62.81	-189.12	233.10	143.63
	-108.05	-273.91	169.49	84.23	-61.48	-187.70	232.40	145.75
Mean	-108.22	-282.36	173.17	84.23	-60.68	-192.38	238.16	148.08

Table 6.22. The normalised times for each heuristic over ten runs and all datasets.

6.4.4 Analysis of further results

The summaries of the results obtained from this further set of experiments are shown in Tables 6.19 – 6.22. Given the difficulty of drawing obvious conclusions from them, this section provides a discussion and offers explanations for any unexpected results arising.

Table 6.19 shows the normalised best results for each approach. These results do not give a clear indication of any one approach significantly surpassing the others. It appears that the *Holistic* and *LastChance* heuristics outperform *Feasibility* and *Combined*, but other conclusions are not so easy to draw. At first glance it seems that the algorithm works best when utilising the network flow model to aid feasibility, and also that the *Holistic* heuristic may be achieving slightly better results, on average. However, when looking at the ten individual results attained for each run, it is clear that there is a great deal of variation within each category. Given that the results for the number of feasible solutions found, shown in Table 6.21, indicate that when the network flow model is not utilised, the algorithm still produces plenty of feasible solutions, there is no need to incorporate it solely for the purposes of feasibility, although it cannot be discounted it if does assist in improving solution cost.

Since these results are not conclusive and do not show a clear advantage of one algorithm over another, we also consider the average results. From the normalised results for the average solution costs in Table 6.20, it is possible to see that when considering the average results, rather than the best, it is the algorithms which do not incorporate network flow which are now superior. Again, the *Holistic* and *LastChance* heuristics produce the better results, with *Holistic* proving slightly more successful on average.

Grouping the datasets by consultant clash matrix

However, these runs have been performed on many different types of data. The datasets may be classified according to whether they were manufactured from the first or second original datasets, whether they have the original or denser consultant clash matrix, or by

how much slack is present. When considering each of these data types individually, the same patterns generally emerge, with one noticeable exception. When the datasets are considered in two separate groups, according to whether the consultant clash matrix used was the original or the modified version, it is the *LastChance* heuristics which are superior using the original matrices, while the *Holistic* heuristic performs best using the modified version. Since the altered matrix generally gives rise to higher overall costs and also greater variations within the costs, it is understandable that it is the results from the datasets with this denser matrix which would be more influential when the two types of data are considered together. Tables 6.23 show the mean of the normalised best results for each approach split into these two categories.

Heuristic	*Feas.*	*Comb.*	*Holis.*	*Last C.*	*Feas.*	*Comb.*	*Holis.*	*Last C.*
Net. flow?	No	No	No	No	Yes	Yes	Yes	Yes
Mean	7.88	3.40	-2.08	-3.45	7.06	1.80	-5.59	-9.03

Original Matrix

Heuristic	*Feas.*	*Comb.*	*Holis.*	*Last C.*	*Feas.*	*Comb.*	*Holis.*	*Last C.*
Net. flow?	No	No	No	No	Yes	Yes	Yes	Yes
Mean	0.02	0.48	-3.45	-1.13	1.82	5.16	-3.39	0.50

Denser Matrix

Tables 6.23. The mean of the normalised best results for each heuristic over ten runs for datasets with the original and modified consultant clash matrices, respectively.

The tables obtained using the average costs show a similar result and so it is difficult to choose between the *Holistic* and *LastChance* heuristics. It is to be expected that the *Feasibility* and *Combined* approaches would not be as successful since they both consider each student in turn and so reduce the flexibility of the algorithm. The *Feasibility* heuristic, in particular, pays no heed to the cost of assignments in the construction which means that the local search will generally start from a much higher cost solution. However, looking at the normalised average costs in Table 6.20, it seems that *Feasibility* is outperforming *Combined* on average. This situation arises when the local search is actually restricted by the lower cost solutions constructed by the *Combined* heuristic, which prevents the search from adequately searching the search space. If the constructed solution is relatively low-cost, but is not able to be optimised using the available neighbourhoods with only improving moves, then the *Combined* approach may

eventually result in poorer cost solutions than *Feasibility*, even though *Feasibility* must make more neighbourhood moves than *Combined* in order to achieve this result. The solution times given in Table 6.22 show that *Combined* does indeed seem to be converging more quickly than *Feasibility*. This situation does not arise in all cases, however, and Table 6.19 shows that the best solutions attained using *Combined* are better than those utilising the *Feasibility* heuristic, on average.

The performance of both the *Feasibility* and *Combined* heuristics is nonetheless inferior to that of the *Holistic* and *LastChance* heuristics and so these approaches are discarded, both with and without the network flow models. The remaining heuristics are still regarded as potentially good solution methods and so are used as a base of comparison for the experiments detailed in the following section.

6.5 Enhancements to the basic algorithm

The experiments in the previous sections were performed using the four basic construction heuristics detailed in Section 6.2.1 as well as four further heuristics which incorporated an addition to the construction to force the satisfaction of the feasibility constraints. Along with these constructions we used three local search neighbourhoods to explore the solution space. Although the optimal solutions were not known, we were able to deduce that the *Holistic* and *LastChance* heuristics were the best performing. However, the range of quality in the solutions produced indicates that these algorithms are not consistently producing optimal solutions and so we now turn to other constructive ideas with the aim of producing a more robust algorithm.

6.5.1 Using the network flow as a construction heuristic

This section introduces the idea of using the network flow model, not just as an aide to the heuristic to determine whether the heuristic's student-firm selection will be feasible, but as a heuristic in its own right. Thus far, the performance of the GRASP construction phase has been relatively poor and although results have been able to show that some

heuristic approaches are performing better than others, a closer inspection of the quality of solutions before and after the local search phase shows that the GRASP is heavily reliant on the local search in order to improve solution quality. On average, the cost of a constructed solution is improved by 91.6% of the initial value with the application of local search, equating to an average improvement of 8649.5. This cost-improvement obviously varies greatly depending on the dataset and heuristic approach used, but in all cases the improvement is similarly marked. The number of local search moves accepted is 680.8 on average, which is significant, given the number of assignments constituting a solution and indicates that the local search is having to 'correct' a significant proportion of the assignments made. This suggests that looking for improved construction techniques would be worthwhile. Furthermore, the number of assignments required for each construction is itself very large: up to 1400 for a schedule with 280 students, with each assignment requiring a large number of calculations regarding the feasibility and cost. Since the first assignments are made by selecting from many zero-cost options it is not surprising that these choices are later found to be poor. In general, it is only towards the end of the scheduling process for timeslot 2 when the first cost is incurred and this means that over two hundred assignments have been made before any idea of solution quality is considered. For the larger datasets this value is closer to four hundred. There is therefore good reason to try a new construction approach, based on the network flow model, to see whether these difficulties can be limited. A network flow approach to the construction would certainly be beneficial in terms of the number of assignments required, since solving one network flow model would assign a full schedule for one timeslot. However, this would not necessarily improve the problem of choosing between zero-cost assignments in the early stages, which is why a network flow construction incorporating look-ahead is also introduced in this section.

This section describes how the network flow model can be modified to include student-firm selection information and gives details of two possible ways of approaching this, by varying the arc-costs.

In its present form, the network flow model simply identifies the minimum and maximum numbers of each type of student who must cover each speciality in each timeslot for the solution to be feasible. We now describe how it is possible to add more detail to this flow model in order to assign students to specific firms by taking into account the specific costs involved. Obviously to do this, new nodes must be added to the model representing individual students and firms. Figure 6.24 shows the new setup. Note that although it is possible to associate each student-firm pair with a cost based on the hospital and consultant clashes, it is not possible to add the student pair cost into this model.

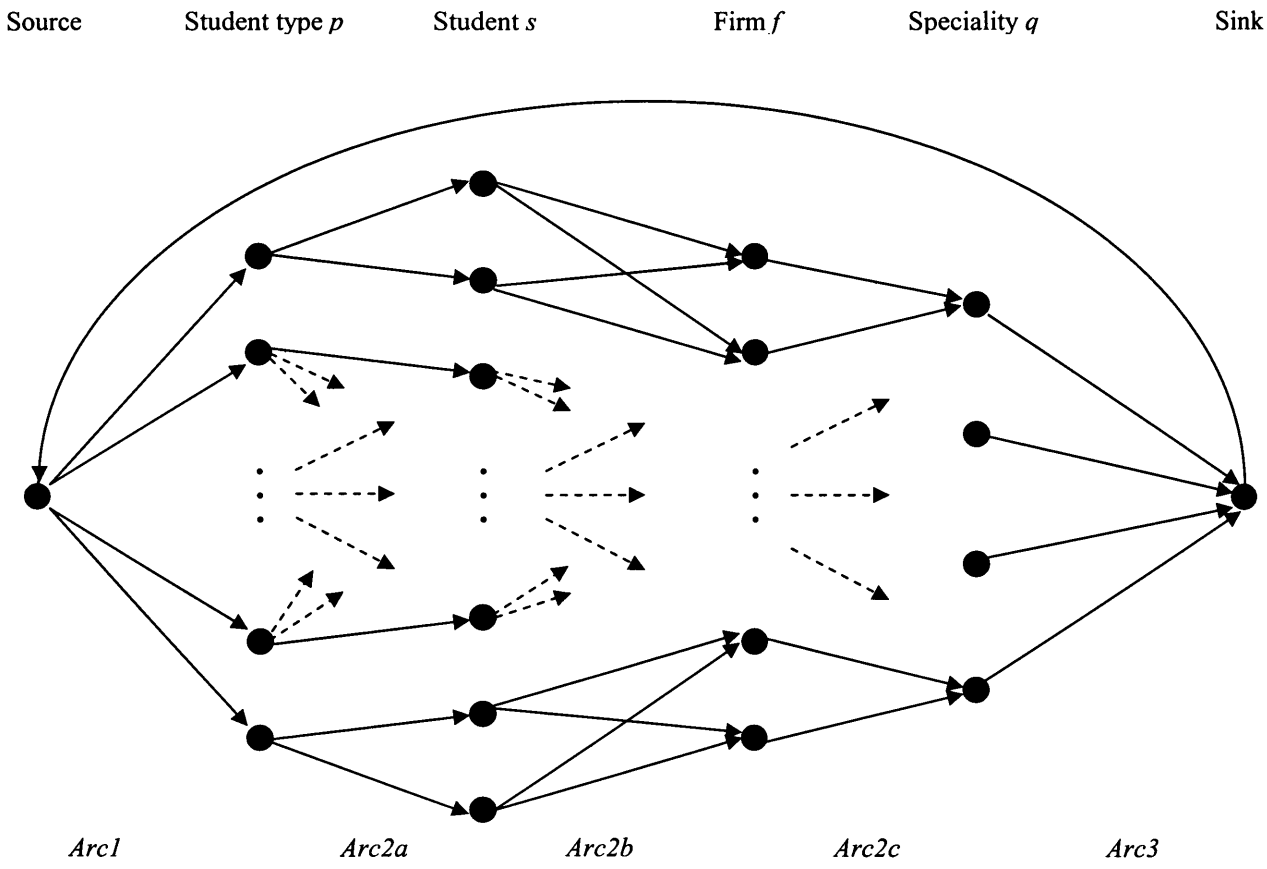


Figure 6.24. Network flow model detailing the situation during an arbitrary timeslot i of the medical student scheduling problem.

Again, the model represents the situation during one particular timeslot i. Arcs 1 and 3 are as in the previous model, but the central section of the flow diagram now incorporates details of the students and firms involved in the scheduling process. We therefore have

two new sets of nodes: student nodes and firm nodes. Figure 6.24 shows some of the types of flows which will transpire from this scheduling setup.

The lack of student pair cost in this modified network flow model means that higher than necessary student pair costs may arise. In datasets where the student pair costs may be more of a factor in finding optimal solutions, especially, we may find that this approach is less suitable. In order to prevent higher than necessary student costs being generated, the arcs are taken in a random order allowing students to prevent students from just following each other around the network. Consider the out-of-kilter algorithm, described in Figure 6.25.

Out-of-kilter algorithm

1. Set up all arcs and nodes.
2. Calculate costs and upper and lower bounds for each arc.
3. Set all flows and node potentials to zero.
4. Calculate y coordinates for each arc.
5. Create a list of arcs, $list_1$, to keep track of which arcs are out-of-kilter.
6. Select an arc from $list_1$, (i,j). If no arcs remain go to step 15.
7. If arc is in kilter, remove from the list and go to step 6, otherwise go to step 8.
8. Work out whether flow in (i,j) needs to be increased or decreased and label nodes i and j accordingly.
9. Create a list of arcs, $list_2$, to keep track of which arcs are labelled.
10. Select an arc from $list_2$, (k,l). If no arcs remain go to step 14.
11. If arc is labelled at only one end and the flow can be altered in the required direction, go to step 12, otherwise remove arc from the list and go to step 10.
12. Label the unlabelled node on arc (k,l) appropriately.
13. If the required terminal node has been labelled update all flows in the chain and go to step 6, otherwise go to step 9.
14. No flow augmenting chain has been found, so update node potentials. If the node potentials cannot be altered further, no feasible flow is possible.
15. All arcs are in kilter and a feasible flow has been found.

Figure 6.25. Description of the out-of-kilter algorithm.

Steps 6 and 10 require the selection of an arc. In the solution of the network flow model introduced in Section 6.2.2, each arc on the list was tested in order, but for this model, where individual students are considered, it is pertinent to randomise the arc selection and

this was done by sampling randomly without replacement from both $list_1$ and $list_2$ as defined in Figure 6.25. Initial testing without random sampling indicated significantly increased student pair constraint violations in the final solutions compared with the random approach. Note also that by randomly selecting arcs, the construction is still very much in keeping with a GRASP approach, since the solution produced will be a random choice of one of the 'n' optimal solutions.

Two types of construction will be considered.

6.5.1.1 Net-construct

Since arcs 1, 3 and the sink-source arc remain unchanged, the details of only the remaining arcs will now be clarified.

Arc2a

An arc (p,s), connecting a student type with a particular student, merely splits each set p into its constituent students. Obviously, each type node p may have one or more arcs leaving from it, but all student nodes s must have exactly one arc arriving from a node p. We therefore have

$$l = u = 1.$$

Since every student must have a type and the particular type holds no penalty, there is zero cost on all arcs (p,s).

Arc2b

The arcs (s,f), connecting each student to his or her possible firms, are now the basis of our construction heuristic. The associated costs with these arcs will direct the flows through arcs which should give lower solution costs. One difficulty with our original construction is the fact that no costs are incurred until many assignments have been made. This makes it difficult to see which allocations may be potentially harmful when the last students come to be scheduled at the end of the construction. By scheduling all students

in a timeslot simultaneously, rather than one student at a time, this may help us avoid having to make poor choices towards the end of the construction.

Each student node s will have arcs connecting it with each firm node f, where f belongs to a speciality student s has not yet covered. These arcs will therefore have

$$l = 0,\ u = 1,$$

since a student s will either be assigned to a particular firm f, or not. The cost on an arc (s_0,f_0) is given by

$$\sum_{f=1}^{v}\sum_{t=1}^{i-1} x_{s_0 ft}\left(10h_{ff_0} + 50c_{ff_0}\right),$$

where i is the timeslot being scheduled. Note that the weightings associated with the hospital and consultant clashes are the same as in the original GRASP construction. We cannot assign a student-clash cost with any of the arcs as these costs cannot be worked out in advance in this way; there is no disadvantage, therefore, in reducing the weightings from (10,50) to (1,5).

Arc2c

The arcs (f,q) simply connect each firm node f to its associated speciality node q. The bounds on the arcs are therefore given by

$$l = 0,\ u = C_f,$$

since the maximum number leaving the firm node must be given by that firm's capacity.

With the network flow model set up, the flow is optimised using the out-of-kilter algorithm (Dowsland 2005). The local search can then be used repeatedly to improve what should be a good starting solution to provide us with a set of potentially high-quality timetables.

Since only hospital and consultant information is incorporated into the construction using the network flow model, the descent part of the metaheuristic is responsible for reducing the number of student clashes. Given that these are of a much lower priority – one tenth

the weighting of the hospital clashes – it is hoped that by skewing the costs incurred towards this type of clash, the overall solution costs will be reduced.

6.5.1.2 Net-lookahead

The arcs for *Net-lookahead* are defined in the same way as for *Net-construct*, but with a difference in the way the *arc2b* costs are calculated. Rather than working out the cost solely on the clashes which would be incurred with firms in preceding timeslots, the *Net-lookahead* construction takes into account what costs are likely to be incurred in the following timeslots as well. For each assignment, some idea of future costs can be ascertained because the specialities which the student must study in the remaining timeslots will be known. By timeslot 5, it follows that the costs for *Net-lookahead* will be identical to those for *Net-construct*. To create a score for a firm f, the potential cost of making the assignment is calculated by considering all feasible future combinations of firms and the hospital and consultant clashes arising from each of these full schedules. We arrange these scores in order of increasing cost and take the mean of a percentage of the lowest cost combinations to give a value U. The score for an arc (s_0,f_0) is then the weighted sum of the *Net-construct* cost for the arc and U. Thus, the cost on arc (s_0,f_0) in timeslot i, $2 \le i \le 4$, is calculated as follows.

We cycle through all feasible firm combinations for the remaining timeslots

$$\left\{(f_{i+1},\ldots,f_5): a_{f_j q}a_{f_k q} = 0 \ \forall q, j \neq k, \ \sum_{j=1}^{5-i}\sum_{f=1}^{v}\sum_{t=1}^{i-1} x_{s_0 ft}a_{fq}a_{f_j q} = 0\right\}$$

and calculate the cost of the clashes with the current firm choice f_0 as well as previous firms assigned to student s_0 in timeslots 1 to $i-1$, using

$$\sum_{j=1}^{5-i}\left[10h_{f_0f_j}+50c_{f_0f_j}+\sum_{f=1}^{v}\sum_{t=1}^{i-1}x_{s_0ft}\left(10h_{ff_j}+50c_{ff_j}\right)\right]+R,$$

where $R=\begin{cases}\sum_{j=1}^{4-i}\sum_{k=j+1}^{5-i}h_{f_jf_k} & \text{if } i=2 \text{ or } 3\\ 0 & \text{otherwise}\end{cases}$.

The R term then counts the costs from clashes between these newly considered firms.

The mean of a percentage of the lowest cost combinations then gives the value U. The cost on an arc (s_0,f_0) is then calculated to be

$$1000\sum_{f=1}^{v}\sum_{t=1}^{i-1}x_{s_0ft}\left(10h_{ff_0}+50c_{ff_0}\right)+U.$$

The percentages of total combinations which will be considered at this stage are 25, 50 and 100 per cent. Note that selecting zero percent would be equivalent to using the *Net-construct* flow model. We would expect that taking the mean from 100 per cent of the possible combinations would give relatively poor results since the worst of these combinations would never be selected; the score does not then give as realistic an idea of the expected future costs as one produced using only the better combinations. The *Net-construct* algorithm can be viewed as the particular case of the *Net-lookahead* where zero per cent of the combinations are considered.

6.5.2 Further experimentation and results

The next set of experiment performed were ten runs using *Net-construct* and ten runs using *Net-lookahead* with each of these percentage combinations: 25 per cent, 50 percent and 100 per cent.

The results of these experiments were normalised against those obtained previously, excluding the inferior results obtained using *Feasibility* and *Combined*. The normalised best costs for each approach are shown in Tables 6.26 – 6.28.

Heuristic	Holis.	Last C.	Holis.	Last C.	Net-con.	Net-l. 25	Net-l. 50	Net-l. 100
Net. flow?	No	No	Yes	Yes	—	—	—	—
	8.93	11.58	5.23	6.64	8.31	-21.38	-5.74	-16.40
	9.86	13.54	6.07	7.63	17.92	-22.15	-9.23	-17.94
	12.24	12.87	3.62	1.45	12.27	-23.54	-8.95	-14.01
	5.09	9.70	8.46	8.79	14.93	-19.24	-6.78	-16.94
	9.42	15.66	9.26	6.70	8.82	-19.93	-13.69	-19.16
	8.01	8.77	9.64	10.78	10.04	-20.56	-10.74	-16.89
	15.11	7.42	7.32	11.35	8.42	-21.62	-9.60	-16.55
	7.63	0.59	13.35	11.35	3.69	-20.50	-7.74	-20.61
	11.85	12.92	2.19	7.44	8.31	-19.93	-7.97	-12.65
	21.44	13.42	9.64	5.03	9.04	-20.46	-11.42	-17.45
Mean	10.96	10.65	7.48	7.71	10.18	-20.93	-9.18	-16.86

Table 6.26. The normalised best results over ten runs and all datasets for *Holistic* and *LastChance* with and without network flow, *Net-construct* and *Net-lookahead* with 25%, 50% and 100% contributions.

Heuristic	Holis.	Last C.	Holis.	Last C.	Net-con.	Net-l. 25	Net-l. 50	Net-l. 100
Net. flow?	No	No	Yes	Yes	—	—	—	—
	9.01	5.77	10.50	5.68	12.13	-19.09	-18.86	-6.86
	9.12	7.34	7.21	6.42	14.05	-22.74	-17.57	-5.31
	14.01	8.57	8.13	4.38	11.84	-21.73	-17.52	-10.19
	12.55	9.74	9.21	5.14	11.10	-19.56	-15.28	-6.69
	8.39	7.47	10.60	4.24	12.27	-20.79	-13.62	-6.76
	12.31	5.71	8.36	8.00	14.28	-20.20	-19.10	-1.35
	11.29	7.53	10.43	7.78	11.92	-21.47	-17.36	-10.85
	14.31	6.99	8.03	5.78	8.23	-22.50	-16.13	-4.40
	11.85	6.71	8.47	6.77	9.31	-21.98	-16.84	-8.14
	8.30	5.14	12.57	0.84	9.65	-19.23	-17.68	-5.66
Mean	11.11	7.10	9.35	5.50	11.48	-20.93	-17.00	-6.62

Table 6.27. The normalised average results over ten runs and all datasets for *Holistic* and *LastChance* with and without network flow, *Net-construct* and *Net-lookahead* with 25%, 50% and 100% contributions.

Heuristic	Holis.	Last C.	Holis.	Last C.	Net-con.	Net-l. 25	Net-l. 50	Net-l. 100
Net. flow?	No	No	Yes	Yes	—	—	—	—
Mean	-14.88	-27.19	-11.34	-23.14	-54.75	480.36	577.26	486.60

Table 6.28. The normalised times for ten runs and all datasets for *Holistic* and *LastChance* with and without network flow, *Net-construct* and *Net-lookahead* with 25%, 50% and 100% contributions.

From Tables 6.26 and 6.27, it is clear that *Net-lookahead* in all cases is able to outperform the other approaches and although there is some variation between the successes of the other heuristics, this difference is relatively insignificant in comparison. In particular, *Net-lookahead* seems to perform better when a lower percentage of the remaining combinations are considered. This is likely to be because the scores arising from an approach including only the better combinations will be a more accurate

reflection of the costs which actually will be incurred in the later stages of construction; when 100 per cent of the possible combinations are considered, for example, this includes very poor cost options, which will certainly be discarded by the construction's greedy approach anyway. *Net-lookahead* using the best 25 per cent of the available combinations is therefore the most successful method so far. We denote this *Net-lookahead-25*.

However, Table 6.28 shows that this increase in solution quality is to the detriment of computational expense. In terms of actual solution time this adds up to an hour to the time required to perform one run. Since these timetables are to be produced annually, rather than weekly, as for the nurse scheduling problem, it is worth spending this extra time to create the best schedule possible although it is, of course, desirable that the schedules should be created as quickly as possible.

Although the *Net-lookahead-25* algorithm is slow, it has so far been the most successful at producing low cost solutions. However, if we look at the results produced in each of the ten runs, the solution costs are still very variable. Table 6.29 gives details of the best solution costs from each run for some of the datasets using *Net-lookahead-25*.

Dataset	10	11	12	16	17	18	22	23	24	41	48
Min.	211	278	428	330	384	504	530	568	769	1656	1441
Mean	233.0	304.6	483.4	344.5	406.1	549.7	543.8	595.4	801.2	1675.4	1459.7
Max.	245	331	522	361	417	581	571	632	829	1687	1470
Range	34	53	94	31	33	77	41	64	60	31	29

Table 6.29. The minimum, maximum, mean and range of the best costs from each run for a selection of datasets using *Net-lookahead-25*.

Table 6.29 indicates that although *Net-lookahead-25* is the best method, it is not robust. Although the optimal solutions are not known, *Net-lookahead-25* is certainly not achieving optimality for all ten runs, since the ten best costs show a great deal of variety and in many cases the difference between the best cost arising from one run and another is significant.

At this point it must be noted that a series of basic experiments have been carried out on the medical student scheduling problem using GRASP and it was at this stage of testing using the nurse scheduling problem, once a series of construction heuristics and neighbourhoods had been exploited, that an ACO approach was considered. The best performing approach, GRASP, was then further tested in order to produce a more aggressive algorithm for the problem. It is therefore prudent to discuss briefly why an ACO approach is not now being considered for the medical student scheduling problem.

As discussed earlier, one of the main reasons why medical student scheduling does not lend itself well to a feedback approach, such as ACO, is the inherent problem symmetry; the structure of the problem is such that there can be no benefit in rewarding a specific assignment or group of assignments since, at the start of the construction, all students are homogeneous, as are the 4 main timeslots and specialities. This means that there would be no way to judge whether an assignment in the current construction was equivalent to a previous assignment of the same description or not. Even using a method where the trails are used to group certain firms, rather than having a trail represent single assignments, the symmetry of the problem prevents information from one cycle reliably influencing the next, since there is symmetry between many firms as well. It should also be noted that the ACO approach was not highly successful even with the nurse scheduling problem, which indicates that, even without the inherent symmetry, the non-linearity of the objective function would hinder the success of an ACO approach. A further indication that pursuing an ACO approach would not be worthwhile is evident in the results so far in this chapter. The constructions thus far have proved to be of a relatively poor quality, relying heavily on the use of local search. If a constructive technique alone is not sufficient to produce at least reasonable solutions, then it is unlikely that an ACO approach, which relies heavily on the construction, would fare better. Note that even the most successful construction technique so far, *Net-lookahead-25*, relies on several hundred neighbourhood moves to improve every construction significantly. Therefore, in order to further investigate the medical student scheduling problem, the current GRASP approaches are enhanced.

In order to improve the GRASP solution approach, an element of memory is introduced from one cycle to the next. The experiments using GRASP with memory are detailed in the following section.

6.5.3 GRASP with memory

This section gives details of the memory procedure and the experiments performed and results obtained using this approach. We begin with an explanation of how the memory approach is incorporated into a GRASP run.

The first cycle is completed as normal. The memory approach then initialises each subsequent construction with the complete timetable of the previous cycle. One timeslot is selected at random and this timeslot is emptied of all current assignments and rebuilt using the heuristic. The local search phase then improves this modified solution as normal. Only timeslots 2-5 are considered for overhaul, since timeslot 1 is of a different nature. Note that this effectively turns the algorithm into a form of iterated local search (ILS), where a local optimum is perturbed and local search reapplied. As was mentioned in Chapter 2, an ILS approach was the most successful heuristic technique applied to the quadratic three-dimensional assignment problem (Q3AP) by Hahn et al. (2008) and since we have already established many similarities between these two problems, especially in their objective functions, there is reason to suppose that such an approach may also be effective here. This approach is applied to the *Holistic* and *LastChance* heuristics with and without the network flow model, and to *Net-construct*. Since *Net-lookahead* is identical to *Net-construct* when only one timetable remains to be scheduled, there seems little point in producing both sets of results.

The results from these experiments are shown in Table 6.30 and are normalised against *Net-lookahead-25*, since this was previously the most successful method.

	With Memory					
Heuristic	*Holis.*	*Last C.*	*Holis.*	*Last C.*	*Net-con.*	*Net-I 25*
Net. flow?	No	No	Yes	Yes	—	—
	10.56	7.99	6.60	11.51	-57.19	20.90
	11.95	9.78	10.99	11.56	-55.38	16.40
	6.18	12.94	9.79	7.18	-57.64	15.37
	8.49	9.62	6.80	10.39	-57.72	22.58
	6.54	9.30	8.71	12.96	-56.00	17.33
	5.92	12.47	10.87	8.36	-58.33	14.59
	8.98	12.94	9.46	11.57	-56.88	16.54
	6.98	11.22	8.63	9.96	-53.95	21.50
	8.42	11.29	5.86	10.04	-54.94	19.34
	2.46	8.60	9.34	14.35	-54.07	19.98
Mean	7.65	10.61	8.71	10.79	-56.21	18.45

Table 6.30. The normalised best results over ten runs and all datasets for *Holistic* and *LastChance* with and without network flow and *Net-construct* with memory and *Net-lookahead-25*.

From Table 6.30, it is clear that *Net-construct* with memory, now denoted *Net-construct-M*, is the most successful algorithm by far. However, it also interesting to note that even the original heuristic approaches are now superior to *Net-lookahead-25* when the memory is applied. Figure 6.31 shows some of the typical plots of the solution costs for each GRASP cycle using *Net-construct-M*, to demonstrate the success of the memory approach.

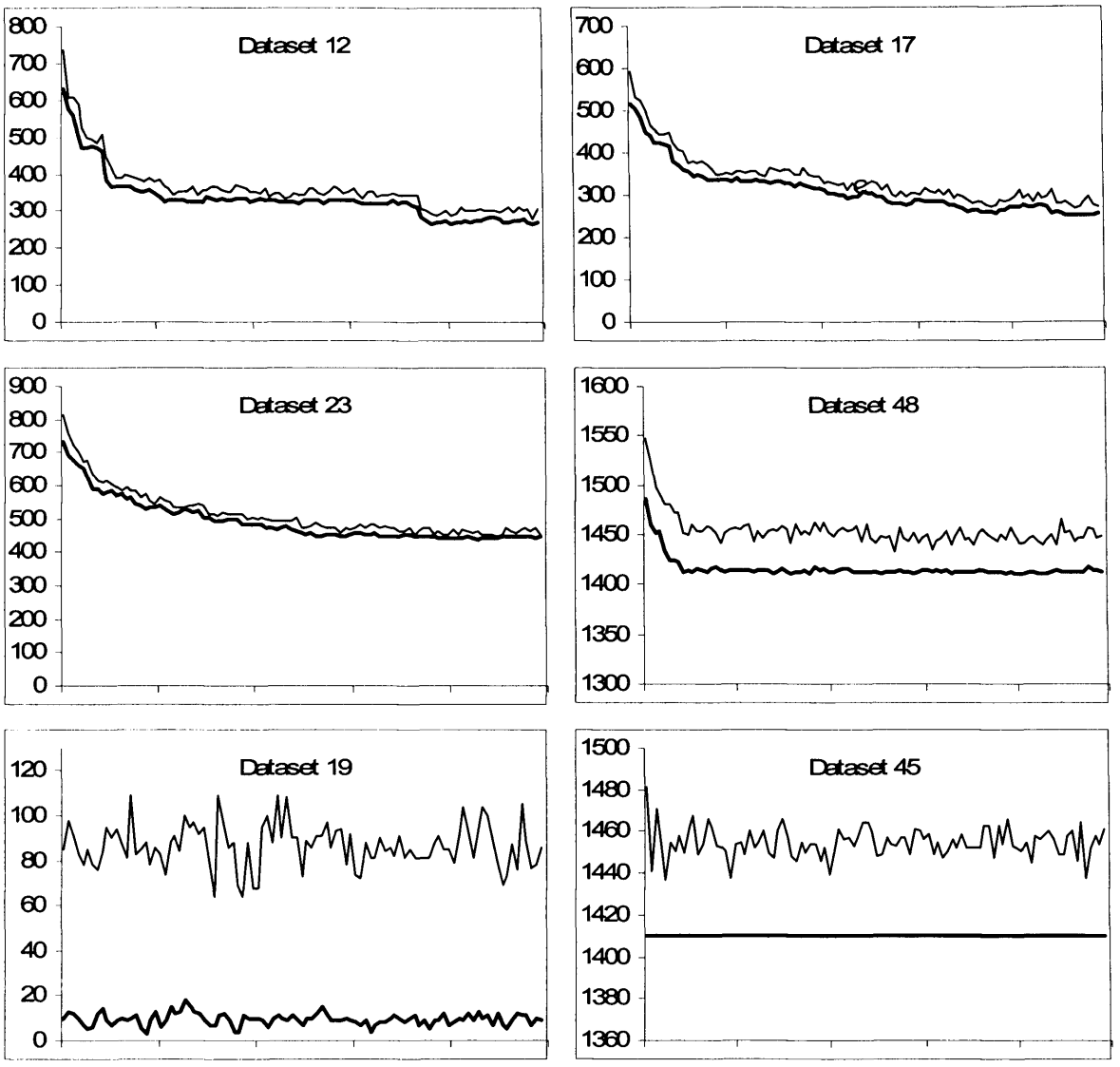


Figure 6.31. Some plots of the solution cost before and after local search over the 100 GRASP cycles.

As can be seen, for datasets 12, 17 and 23, for example, the use of memory is clearly improving the solution cost over time. By repairing a mostly completed solution, the local search is initiated with a relatively low-cost solution in a good area of the solution space and thus the solutions gradually improve over time. This improvement is more pronounced in the first few cycles, but improvement is generally seen even in the later stages of the run and is the result of a gradual improvement in hospital cost. The consultant constraints are usually satisfied in the first few cycles due to their higher weighting in the objective function and this produces the initial large decrease in cost. The hospital constraint violations are then slowly improved upon with the student costs as

a secondary consideration. Dataset 48 is an example of a situation in which both the consultant and hospital costs converge early on and the remaining fluctuations are solely with regard to the varying student costs. Datasets 19 and 45 are the final example plots, showing situations where the memory appears to have far less of an impact. Dataset 19 is similar to dataset 48. The consultant and hospital costs are easily minimised, but the algorithm struggles to reduce student-pair costs. In the case of dataset 48, however, the memory was of assistance in reducing the hospital and consultant costs initially, whereas for dataset 19, even this initial assistance is not required as both of these types of violation are reduced to zero easily. For datasets of this type, although the plots do not show any obvious signs that the memory is improving the schedules, the best solution costs found from each run are generally still better than those produced using *Net-lookahead-25*. Since it is only the student-pair costs which are difficult to minimise, by searching intensively in a good area of the solution space, the algorithm is more able to find high-quality solutions. The final example plot, from dataset 45, is where all three types of costs converge easily, and thus the same, potentially optimal, solution is obtained from each cycle.

The use of memory thus seems an important addition to the algorithm. Since the construction is doing much less work during each cycle, the solution times are also greatly improved. Table 6.32 shows the mean normalised solution times of the methods with memory and compares these with that of the original *Net-construct* approach, since this was previously the quickest algorithm.

	With Memory					
Heuristic	*Holis.*	*Last C.*	*Holis.*	*Last C.*	*Net-con.*	*Net-con.*
Net. flow?	No	No	Yes	Yes	—	—
Mean	-4.41	-4.36	-3.53	-0.12	-65.93	78.35

Table 6.32. Mean normalised times to complete one run for each of the approaches.

Net-construct-M is therefore both the most successful algorithm and the quickest so far. The time required to perform one run of 100 GRASP cycles varies depending on the dataset, but is less than a minute in all cases.

6.5.4 Sensitivity to weights

Although we have so far achieved the best results using a heuristic based on the network flow model, it is important to note that *Net-construct* does not include any information about the student pairings in the construction stage and its success may be partly reliant on the fact that the student costs have a much lower weighting than the other types of cost. In order to test this we re-ran new experiments with the respective weightings for the consultant, hospital and student-pair constraint violations set to 50:10:100, respectively, compared with the original weightings of 50:10:1. Ten runs were performed using these weightings for both *Net-construct-M* as well as the best of the original heuristics so far, *Holistic* with memory and no network flow, simply referred to as *Holistic-M* in the following.

Table 6.33 shows the normalised bests results from these experiments. These results are shown over all datasets as usual, but also split into datasets using the original and modified consultant clash matrices. Clearly, these results are not directly comparable with those obtained previously, but we will be able to discuss their relative success.

	All datasets		*Original matrix*		*Denser matrix*	
Heuristic	*Holis-M*	*Net-Con-M*	*Holis-M*	*Net-Con-M*	*Holis-M*	*Net-Con-M*
	19.54	-18.10	-0.82	5.00	20.36	-23.10
	16.48	-19.51	-3.42	3.72	19.90	-23.22
	14.36	-22.19	-7.00	-0.67	21.36	-21.52
	17.67	-11.40	-3.65	8.12	21.32	-19.52
	16.34	-19.53	-6.58	4.84	22.92	-24.37
	21.59	-15.12	-2.70	5.52	24.28	-20.64
	22.27	-22.95	-2.68	-1.56	24.95	-21.39
	18.87	-19.59	-2.35	2.68	21.22	-22.27
	20.30	-19.11	-2.11	4.54	22.41	-23.65
	17.61	-17.56	-3.29	2.39	20.91	-19.95
Mean	18.50	-18.50	-3.46	3.46	21.96	-21.96

Table 6.33. The normalised best results over ten runs with *Holistic-M* and *Net-construct-M* using the altered constraint weightings.

Although it is clear that *Net-construct-M* is still performing the best over all datasets, there is a clear divide in the successes of the two algorithms between datasets with and

without the denser consultant clash matrices. For datasets where it is relatively easy to minimise the consultant constraint violations, it is generally the student costs which are harder to minimise and make the difference between a good or a poor solution. In these situations the *Net-construct-M* algorithm is at a disadvantage, since these important student-pair costs are not considered during the construction stage at all, whereas *Holistic-M* is much more able to minimise these costs effectively.

However, once the consultant clash matrix is modified, it is the consultant costs which are more difficult to minimise. During construction, *Holistic-M* reduces all costs as far as possible, but gives preference to the higher-weighted student costs. This is to the detriment of the consultant and hospital costs. Once the student constraint violations have been minimised, no other improvements can be made which will negatively affect the student costs due to their high weighting. By minimising these 'secondary' costs before attempting to minimise the student costs, *Net-construct-M* is actually more successful.

The results of this section indicate that it is important to choose a construction approach which effectively tackles the most difficult set of constraints. For datasets where minimising the student-pair clashes was the most difficult task, the *Holistic-M* construction, which took these into account, was the more successful. For the datasets where the consultant clashes were more of a problem, however, the *Net-construct-M* approach, which focused more on these consultant costs, proved to be much more successful. It seems, then, that although there is benefit in balancing all constraints in the construction phase, the important factor in the success of a GRASP approach lies with ensuring that the construction is able to effectively reduce violations of the most difficult set of constraints. This was also seen in the GRASP approach to the nurse scheduling problem in Chapter 4, where the main difficulty was in finding feasible solutions; the best approaches were heavily biased towards feasibility in the construction phase and the incorporation of the knapsack was essential for this purpose.

We therefore maintain that *Net-construct-M* is the best solution method available thus far, although it is interesting to bear in mind that it does ignore a complete set of constraints

during the construction phase and so it may not be a prudent choice if the student constraints were to become more important.

Obviously, in a real world situation, it is unlikely that the student-pair constraints would be valued more highly that the others, but these experiments are included for other problems which may arise with a similar formulation, and for the sake of completeness.

6.5.5 Improvements to the memory approach

Since applying memory to the algorithms was so successful, we attempt to further improve solutions, by trying three further memory tactics. Previously, we initialised the construction with the schedule from the previous cycle and selected a random timeslot for modification.

We suggest the following improvements:

1. Initialise with the best schedule found so far
2. Select the timeslot with the highest contribution to the cost for modification
3. Both initialise with the best schedule and remove the highest-cost timeslot.

We applied these three approaches to *Net-construct* and denote them by the suffixes *-B*, *-W* and *-BW*, respectively. The best results obtained from each method are detailed in Table 6.34.

Heuristic	*Net-con-B*	*Net-con-W*	*Net-con-BW*	*Net-con-M*
	-9.44	0.77	10.46	-3.16
	-6.52	-1.96	15.15	-5.37
	-10.47	-6.34	5.23	-7.26
	-11.49	1.65	10.26	-9.34
	-10.18	2.42	6.86	-4.91
	-5.90	1.66	15.54	-7.90
	-15.16	3.61	11.61	-3.81
	-2.94	5.81	11.05	-0.82
	-7.15	4.87	16.75	-1.79
	-8.90	0.24	19.33	-2.46
Mean	-8.81	1.27	12.22	-4.68

Table 6.34. The normalised best results over ten runs and all datasets for *Net-construct* with the different types of memory enhancements.

It was expected that each enhancement would be likely to improve solution quality and that combining the two should give the most improvement, however this is clearly not the case. While modifying the best solution so far (*Net-construct-B*) does give some improvement, selecting the highest-cost timeslot for modification (*Net-construct-W*) actually decreases solution quality and using both enhancements (*Net-construct-BW*) gives the worst results of all the methods.

The reason for the poor performance of *Net-construct-W* can be explained by looking more closely at the results obtained. In most of the initial cycles, it is the last of the five timeslots which contributes the most to the solution cost and is therefore chosen for rebuilding. However, it is not just these initial cycles for which the fifth timeslot is selected; the fifth timeslot is selected over 90 per cent of the time and this results in the GRASP stagnating, since it is rebuilding from the same partial solution over and over again. When *Net-construct-BW* is used, this situation is only compounded; since the best solution often remains the best for several cycles and, since the worst timeslot is selected for removal, it is the same partial solution being rebuilt in every cycle. Thus the algorithm is not able to explore the search space fully and so stagnates easily.

However, despite the overall lack of success of these methods, there are datasets for which these generally poorer methods produce solution costs lower than the best found using *Net-construct-M* or *Net-construct-B*. For this reason, rather than abandoning these enhancements, two new approaches are suggested, which incorporate all three enhancements.

6.5.6 Further modifications to the memory approach

The first of these combined memory approaches, which will be denoted by *Net-construct-R* in the following, is a straightforward randomisation of all the memory approaches so far. At the beginning of each cycle, the heuristic will decide with equal probability which method will be applied. The choices are:

- Use the original memory approach (*-M*): remove a random timeslot from the solution of the previous cycle.
- Use the best solution approach (*-B*): remove a random timeslot from the best solution of the run.
- Use the worst timeslot approach (*-W*): remove the timeslot with the highest cost-contribution from the solution of the previous cycle.
- Use the combined approach (*-BW*): remove the timeslot with the highest cost-contribution from the best solution of the run.

The second approach, which will be denoted by *Net-construct-I*, also combines the different methods, but does so more intelligently. It begins by applying the '*-W*' approach, rebuilding the worst timeslot for each new cycle. Once the hospital and consultant costs have been unchanged and the same timeslot has been selected for five cycles, it is deemed that the search has stagnated and the heuristic changes to the '*-B*' approach, rebuilding a random timeslot from the best solution so far. Once the best cost solution so far has remained the same for five cycles, the straightforward memory approach '*-M*' applied initially is used until a new best cost solution is found. If the new solution has decreased hospital or consultant costs, the worst timeslot approach is used

again or if the hospital and consultant costs are unchanged the best solution approach is revisited. The process repeats until the stopping criteria, reaching 100 cycles, is met.

Table 6.35 shows the normalised best results from ten runs using each of these two combined memory approaches, *Net-construct-R* and *Net-construct-I*, against the best performing method so far, *Net-construct-B* and the original memory approach *Net-construct-M*.

Heuristic	*Net-con-R*	*Net-con-I*	*Net-con-B*	*Net-con-M*
	-5.80	-5.50	-3.38	-0.75
	-5.06	8.78	4.05	2.48
	0.48	1.59	-3.21	-4.80
	-11.22	0.87	-4.79	-5.78
	6.06	11.07	-2.03	-0.87
	0.09	7.55	8.37	0.61
	-4.89	-3.46	-10.44	4.67
	2.70	-7.40	12.05	12.39
	-1.32	0.33	-2.19	1.33
	-10.77	1.23	-0.87	7.82
Mean	-2.97	1.51	-0.24	1.71

Table 6.35. The normalised best results over ten runs and all datasets for the two memory approaches combining the different enhancements as well as the best of the previous approaches.

The results in Table 6.35 show that *Net-construct-R* appears to be the most successful method so far, although there is still a great deal of variation in cost between the ten runs and the difference between the different approaches is not very pronounced.

6.5.7 Further analysis and discussion

Until this point, we have looked at the results obtained giving equal importance to each of the datasets. However, this is not necessarily prudent as shall be explained in the following. For many of the datasets, the best solution found is repeated in nearly all runs using each method. In the rare circumstance that a run of higher cost is produced, the impact on the normalised costs is enormous, although this is probably unrepresentative in real-world terms. Obviously if all methods produce the 'optimal' result ten times, but one only produces it nine times, this method may not be as robust, but when the added cost is

negligible it seems impractical to base long-term decisions on these inflated statistics when decisions on relative success should more probably be based on solutions to more 'difficult' datasets.

Table 6.36 shows details of the actual lowest cost found for each dataset. The '*Net-con-R*' column gives the best result produced for each dataset using *Net-construct-R*, the most successful method so far. Where this is higher than the best known solution, the figure is shown in italics. The third column gives the number of times from ten these results were produced using *Net-construct-R* and the final columns give the number of times from ten that these best costs were within 9 and 49 of the overall best cost for the dataset. Investigating the solutions within 9 of the best known ensures that these are all allowing further violations of the least important constraint only, related to the student pairings. Similarly, solutions within 49 ensure no additional consultant costs. At this point it is interesting to note that dataset 1 describes the same problem as that solved by Fuller (1998), although Fuller assigns a weight of 5, rather than 50, to violations of the consultant constraints. However, the solution produced by Fuller's approach, as mentioned briefly in Chapter 2, has 1 consultant clash, 24 hospital clashes and 46 student pair clashes, giving a total cost of 291 using consultant weight 5 or 336 using the updated weighting scheme. It is therefore encouraging to note that the GRASP approach has reduced this total cost to just 30, comprising three hospital constraint violations.

Table 6.36 clearly shows that *Net-construct-R* is very capable of producing good solutions for all datasets created from the original second dataset (datasets 25-48) and for the three versions (original, tightened and tight) of the first original dataset, but struggles to find consistently good solutions for the remaining datasets. When looking at which datasets are solved within 9 of the best found so far a clear pattern emerges: For datasets with the original consultant clash matrix it is the student cost which is lowering the solution quality, but for the denser matrix the quality is more likely to be adversely affected by extra hospital and, in some cases, consultant constraint violations.

The nature of the consultant clash matrix has clearly had a major impact on the difficulty of the problem. As we mentioned in Section 6.3.4, the original matrices had the attribute that any two firms with a consultant in common would be in the same hospital. The result of this was that by avoiding consultant clashes, further hospital clashes were automatically avoided as well. By allowing consultants to head firms in different hospitals, however, the two constraints are no longer in concordance and the conflict between the two creates additional difficulty in allowing the solution to steadily converge to a low-cost solution.

We are therefore interested not only in the normalised best solution costs obtained, but also in the robustness of the algorithm and the likelihood that solutions found will be of an acceptable quality. For datasets which have proved particularly difficult, this is especially the case and so we provide more details in the following of the success of each algorithm with respect to each individual dataset. Given that there has been little difficulty in finding the best known solution for datasets 1-3 and 25-48, it is only datasets 4-24 which seem to be proving difficult to converge.

Dataset	Best found	*Net-con-R*	/10	in 9	in 49
1	30	30	10	10	10
2	80	80	10	10	10
3	240	240	10	10	10
4	168	*181*	1	0	10
5	209	*217*	1	1	8
6	293	293	1	6	10
7	0	0	3	10	10
8	1	*2*	1	10	10
9	2	2	1	10	10
10	96	*104*	1	1	8
11	125	*132*	1	1	8
12	146	*181*	1	0	1
13	0	0	8	10	10
14	20	20	3	10	10
15	180	180	9	10	10
16	189	*209*	1	0	7
17	248	*260*	1	0	9
18	292	*303*	1	0	8
19	1	1	1	10	10
20	1	1	1	10	10
21	3	3	1	10	10
22	281	*317*	2	0	2
23	341	*354*	1	0	3
24	411	*435*	1	0	2
25	900	900	10	10	10
26	850	850	10	10	10
27	610	610	10	10	10
28	900	900	10	10	10
29	850	850	10	10	10
30	610	610	10	10	10
31	780	780	10	10	10
32	730	730	10	10	10
33	490	490	10	10	10
34	780	780	10	10	10
35	730	730	10	10	10
36	490	490	9	10	10
37	1700	1700	10	10	10
38	1650	1650	10	10	10
39	1410	1410	10	10	10
40	1700	1700	10	10	10
41	1650	1650	10	10	10
42	1410	1410	10	10	10
43	1700	1700	10	10	10
44	1650	1650	10	10	10
45	1410	1410	10	10	10
46	1700	1700	10	10	10
47	1650	1650	10	10	10
48	1410	1410	10	10	10

Table 6.36. Details of the best known costs for each dataset, and the relative success of *Net-construct-R*.

The following figure, Figure 6.37, shows the breakdown by dataset for the ten runs produced using some of the most recent methods. The black bars indicate how many times the best known solution was found, the grey bars, solution costs within 9 of the best known solution, indicating violation of only the least significant, student pair, constraints and the white bars, solution costs within 49 of the best known solution, indicating that there have been no consultant cost violations.

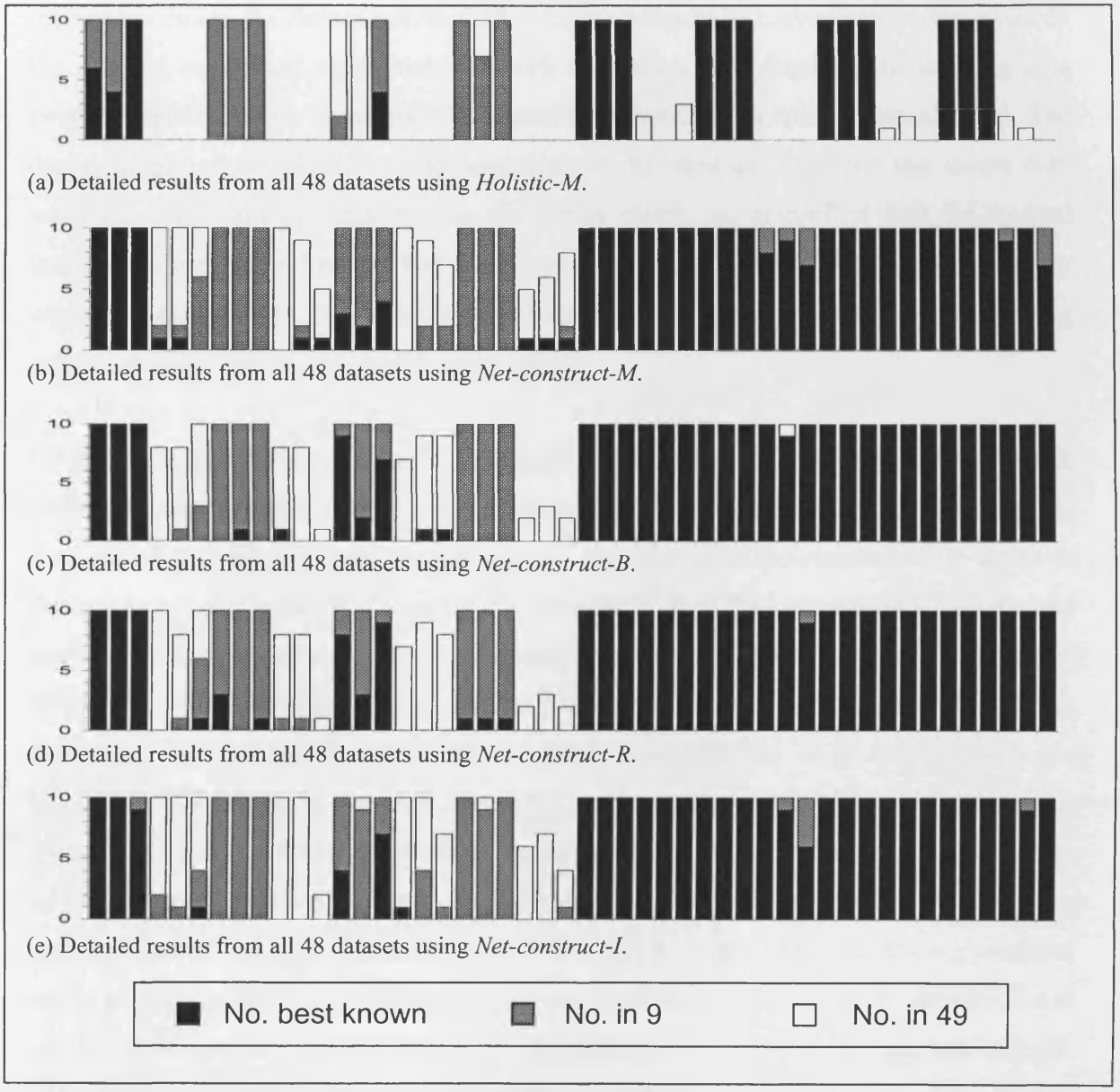


Figure 6.37. Solution cost breakdown by dataset for timetables produced using different heuristic approaches with memory.

It can be seen from Figure 6.37 that all of the approaches (b)-(e) using *Net-construct* with memory strategies produce relatively similar results. It is clear that although datasets 25-48 easily produce the best known solution in each run, the *Net-construct* method is required in order to produce these results and that without this assistance in the constructions, *Holistic-M* (Figure 6.37 (a)) is unable to produce results of a similar standard for datasets using the denser consultant clash matrix. One of the reasons for the noticeable difference in success between datasets with the original and denser clash matrices is due to the denser matrices allowing consultants to move between hospitals. In the original consultant clash matrices each consultant was restricted to working in a single hospital and no movement of consultants between hospitals was allowed. The denser matrix allows consultants to head firms in different hospitals and this means that, while the consultant constraints using the denser matrix are in conflict with the hospital constraints, the original matrix has the effect of reinforcing them. Thus while the denser consultant clash matrix provides the opportunity to solve a problem with three conflicting soft constraints, the original matrix only has two.

Of the *Net-construct* approaches it is difficult to determine which is the most successful. Although the statistical analysis over all datasets would suggest that *Net-construct-R* is the best on average, it must be pointed out that although *Net-construct-I* is worse at finding very good solutions, more of the solutions it does find are within 49 of the best known. If a solution of cost at least 50 greater than the best known solution is considered undesirable, then while *Net-construct-R* produces at least one such undesirable solution from ten runs for ten datasets, *Net-construct-I* does this for only five datasets and generally produces fewer 'undesirable' solutions for each of these. Thus, *Net-construct-I*, could be considered the most robust of the methods; the solutions found may not always be of the highest quality, but they are more likely to all be of a reasonable quality. Note that for most of the datasets we are unable to judge how good the best known solutions are in relation to the optimal. However, as can be seen in Table 6.36, for datasets 7 and 13, the best known solutions have zero cost and so we know that these are optimal. Clearly, for any dataset where the best known solution is very low in terms of actual cost, even if this is not the optimal value, it cannot be very far from optimal.

6.6 Conclusions

As stated in the introduction in Chapter 1, the focus of this thesis is concerned with balancing different objectives using greedy, randomised constructive techniques within a metaheuristic framework. The medical student scheduling problem presents an excellent opportunity to study such a problem. Not only are conflicts present between the hard and soft constraints, but also between the various soft constraints; in order to be able to minimise all costs a balance must be struck which gives an appropriate amount of consideration to each aspect and thus produce solutions of as high a quality as possible. By altering the consultant clash matrix to allow movement of consultants between hospitals we were able to solve datasets with both two and three sets of conflicting soft constraints. It was found that although a network flow approach could be incorporated to guarantee feasible solutions, the difficulty of finding feasible solutions to this problem was not severe enough to warrant the enforcement of such restrictions; not only were plenty of feasible solutions found without this aid, but the quality of the solutions were impeded by its use. Thus a straightforward approach fared better, although the overall quality of the solutions obtained using any of these constructive heuristics was generally quite poor.

Although the network flow model was not successful when used as an aid for feasibility, when a modified model was used to find a good balance for the hospital and consultant constraints, it performed very well. Initial experiments comparing different heuristics showed that, although the *Holistic* and *LastChance* heuristics outperformed *Feasibility* and *Combined*, all approaches performed relatively poorly, compared with the results obtained from the network flow construction approaches. Furthermore, although the idea of using look-ahead was not very successful for individual assignments, the *LastChance* and *Holistic* heuristics performing similarly, it proved successful when looking ahead at the potential consultant and hospital clashes in the remaining timeslots, as shown by the success of the *Net-lookahead* construction. However, one of the drawbacks of using this type of method is that it relies upon the fact that the student pair constraints, which cannot be modelled into the network, are the least important. We showed, by assigning the

student costs more weight, that there may be cases where the best solution is not found using a network flow construction approach if the importance of these constraints were altered. However, even with a high weight on the student pair costs, the difficulty of balancing just the hospital and consultant clashes meant that the results obtained using *Net-lookahead* were reasonable. Thus we have provided a good construction which builds a solution based on only the feasibility and the hospital and consultant constraints while the local search is entirely responsible for optimising the student costs. For the medical student scheduling problem, then, we have had the opportunity to study the conflict between the different soft constraints and have discovered that the least significant of these, namely the student pair constraints, may be omitted entirely from the construction without any loss of final solution quality. However, the experiments carried out with increased weight on the student costs showed that it is not necessarily the highest-weighted constraints which should be given priority in the construction, but the ones for which minimising the number of violations proves the most difficult. For the medical student scheduling problem, however, the particular difficulty of each set of constraints varied between datasets and so the final choice of approach was the one which provided the best solutions on average.

The final improvement to the algorithm came from the incorporation of an element of memory. It was shown previously that the symmetry of the medical student scheduling problem, as well as the poor quality of the constructed solutions, meant that using a feedback approach such as ACO would be unlikely to be successful. However, by rebuilding a single timeslot from a previous solution, and ensuring that this did not result in early stagnation of the search, solutions were found of a much higher quality than those previously. As was discussed in the previous section, the optimal cost of each dataset is not known, but since the best known solution found is not produced in every run for all datasets, we know that the algorithm is not consistently producing optimal solutions. We do know that the algorithm is more likely to find better solutions more consistently for datasets using the original consultant clash matrix, and therefore investigating the balance between two, rather than three, conflicting soft constraints. However, given that this problem has not previously been studied in such depth, we have

gained a lot of information about the balance between the different costs and the methods which do and do not assist in the search for high-quality solutions.

In terms of the aims of this thesis, we have been successful in showing that the role of the construction is important, with a great deal of variation in solution quality between approaches using different constructive heuristics, and with the *Net-lookahead* providing far superior results when using a straightforward approach with no memory. Furthermore, we have shown the benefit obtained by tackling the most difficult constraints within the construction phase. We have also deduced that, although the construction is important it needs to be used in combination with a suitable local search element in order for the approach to be successful. This was shown most clearly in the difference in solution quality between approaches with and without the swap neighbourhood, and it was shown that the constructions relied heavily on local search in order to improve solution quality. Furthermore, we showed that although it is possible to balance the different constraints within the construction, using an exact method to minimise the costs relating to the simpler, but more important, constraints in each timeslot and allowing the local search to minimise the costs relating to the more complex, but less important, student constraints is a more effective method. Finally, exploiting the underlying feasibility structure of the problem to hybridise the GRASP with an exact method capable of ensuring satisfaction of the hard constraints was not found to beneficial in this case; the problem of finding feasible solutions for medical student scheduling was not found to be sufficiently difficult for such an approach to be worthwhile.

This problem has been shown sufficiently interesting to warrant further research and, given that it is not possible to solve exactly it will be interesting to see if further experimentation will produce results better than the best solutions obtained here. Clearly, we have provided an explanation of why an ACO algorithm would not be suitable for application to this problem, but it may be interesting to experiment with other metaheuristics such as Tabu Search, which has been applied so effectively in the past to nurse scheduling and would also be applicable here. In may also be interesting to experiment with a metaheuristic such as simulated annealing; by slowly reducing the cost

of uphill moves allowed during the search, we would gradually be pruning the search of accepting the different types of constraint violation. Since this problem offers different costs for each of the conflicting constraints the search would slowly allow no further violations of the consultant, hospital and finally student constraints, in that order as the temperature is reduced and it would be interesting to see the effect this has on solution quality. Given that the memory approach of rebuilding a timeslot was so successful, it would also be interesting to apply an iterated local search approach, which also relies on repeatedly perturbing and improving solutions.

A further point of interest would be to consider instances where there is not an exact match between the number of students and the spaces available in timeslot 1. Thus far, there has been no slack in timeslot 1 and so the schedules created in this first timeslot have created a basis on which the rest of the solution is built. It would be interesting to see how this added choice can be managed if spare places were allowed in timeslot 1. There would then be reason to widen the local search to include moves affecting timeslot 1.

In conclusion, we have provided an algorithm to a new problem, and built a method based on hybridising a network flow construction with local search and memory in order to solve it. Although the optimal costs are not known, we know we have found optimal solutions for at least two datasets and it is likely that several others, especially those based on the larger dataset, are also optimal. However, those datasets for which the best known solution is not found on every occasion provide an interesting framework for further research and the prospect of creating an even more robust algorithm, even if the best known solutions cannot be improved upon.

Chapter 7

Conclusions and suggestions for further research

This thesis has focused on the concept of balancing conflicting constraints in large, NP-hard problems within a constructive metaheuristic framework. The two problems which have been the subject of this investigation were both shown to exhibit the necessary conflicts to make them suitable for such a study and it has been shown for both problems that finding robust methods, able to balance these constraints by using a constructive approach, is non-trivial. In the case of the nurse scheduling problem, the conflict was between feasibility and optimality, and finding feasible solutions alone has been shown in the literature to be particularly difficult. The medical student scheduling problem presented a different set of conflicts and finding feasible solutions was, on its own, a relatively easy task. However, the complex objective function incorporated quadratic as well as higher order terms from up to three conflicting soft constraints and so balancing all these factors in order to produce solutions which were both feasible and optimal with regards to the objectives was much more difficult.

The constructive metaheuristic approaches used were GRASP, which was applied to both problems, and ACO, which was only applied to nurse scheduling. While GRASP

was easy to apply in both cases, there were important factors which suggested that an ACO approach to medical student scheduling would be unsuccessful. Firstly, as was discussed in Chapter 2, the medical student scheduling problem is highly symmetrical which means that applying feedback through the use of a trail is likely to be ineffective, since specific assignments are not important to solution quality, only the relationship between assignments. Furthermore, since the quality of solutions obtained using an ACO approach for the nurse scheduling problem could not rival those of the GRASP, and the quality of the GRASP constructions for medical student scheduling were relatively poor, this only reinforced the conclusion that applying ACO to medical student scheduling would not be worthwhile. Before discussing the work in this thesis in relation to the aims stated in the introduction, a statement of the key results and conclusions is provided.

Main conclusions:

- For both GRASP and ACO, selection of an appropriate construction heuristic is important.
- For both problems, the best construction exploited problem structure although, for the medical student scheduling problem, this was not in the way anticipated.
- The balance of the constraints in the construction is important, and there is a suggestion that 'difficult' constraints should be given priority.

Finer points worth noting:

- Key features of the problem considered should be identified and exploited where possible.
- Large construction steps, where several assignments are made simultaneously, can be beneficial, especially if some form of exact method can be used to optimise some set, or sets, of constraints. This is a similar idea to using large neighbourhoods in the local search phase where several assignments are replaced in one step.
- Local search is important, even for generation-based methods with feedback.
- Suitable trail definitions are difficult to define for these types of problem and so further complicate the design of an effective algorithm.

- Utilising information from previous cycles is important, especially where the local search does a lot of work. By utilising such information, the construction is generally able to provide a better starting point for the local search in the next cycles. This was especially evident for the memory approach for medical student scheduling.
- Looking ahead has benefits over a purely myopic approach.
- Diversity in the constructed solutions is important, but not at the expense of solution quality. The nurse scheduling problem approach was improved with the use of knapsack-based diversification and the medical student scheduling problem approach was hindered by memory approaches which did not allow a variety of constructions. However, both approaches were more successful when a shorter RCL was applied in the construction.

Note that all the above conclusions are based on the results obtained for just the two problems investigated in this thesis. Therefore further investigation is required in order to be able to draw conclusions in the general case.

Now that the key results have been stated, a discussion is provided which examines how these findings fit in with the original aims of the thesis.

As stated in the introduction, this thesis had three main aims and a fourth, secondary aim:

5. To investigate the role of the construction within a metaheuristic approach.
6. To investigate how different constraints may be balanced within a construction.
7. To investigate how exploiting problem structure with regards to feasibility may improve constraint balance within the construction.
8. To produce a robust method for each problem, capable of producing high-quality solutions for all problem instances.

The work in this thesis has dealt with the fulfilment of these aims and, in many cases, there has been overlap between them. The following sections provide a discussion of how each aim has been fulfilled and the subsequent conclusions which may be drawn.

7.1 Investigating the role of the construction within a metaheuristic approach.

As was mentioned in the introduction, there are two types of metaheuristic approaches: those which are initialised with a random solution and those for which the role of the construction is important. This thesis has focused on the second type and, in particular, on ascertaining how important the role of the construction is on final solution quality and whether an aggressive construction would be able to compensate for the use of only minimal local search.

It has been stated repeatedly in the literature that no constructive technique can guarantee locally optimal solutions and therefore an element of local search is almost always beneficial and for the problems presented in this thesis this was certainly the case. However, although the construction may not be sufficient to produce high quality solutions on its own, this thesis has certainly shown that the construction plays an important part in solution quality. All approaches developed showed significant variation between the results obtained using different constructions. With the same amount of local search, but employing different constructive approaches, the solution quality was shown to be very variable and this indicates that the role of the construction is to create a starting point for the local search which is in a good area of the solution space, one which is in the basin of attraction of a high quality local optimum, given the neighbourhoods employed.

However, the GRASP approach in Chapter 4 showed that, even though there was a significant difference between the quality of the solutions obtained using different construction heuristics, the local search was necessary to provide a good number of feasible solutions. For the GRASP approach applied to medical student scheduling in Chapter 6, the local search part of the algorithm was clearly important to solution quality and it was shown, for example, that incorporating the swap neighbourhood made a significant improvement to the results obtained. The ACO approach in Chapter 5 was initially applied with no local search and results showed a definite improvement in solution quality when even a relatively small amount of local search was incorporated. Thus it can be deduced that although the role of the construction is important, it is, indeed, beneficial to have some element of local search.

It has been shown that for the nurse scheduling approaches, a construction is able to create a reasonable balance between feasibility and optimality in the construction and, although some local search is necessary in order to find very good solutions, the constructions are able to steer the search towards areas of the solution space which are promising both in terms of feasibility and nurse preferences. However, although the heuristics employed for the medical student scheduling problem were able to incorporate costs relating to all constraints, it was found that a construction focusing on just some of the constraints was the most successful. In particular, it was important that the construction tackled the most difficult set of constraints, since minimising violations of these in the construction provided a starting point for the local search from which high-quality solutions were more likely to be found. For the nurse scheduling problem, the feasibility constraints were known to be the most difficult, while for the medical student scheduling problem, the most difficult set of constraints to meet was more data-specific. The consultant and hospital clashes were generally more difficult to minimise where the consultant clash matrix had been altered, whereas the student costs provided a greater challenge for most other datasets.

Consider the use of the knapsack model, in conjunction with the construction of a nursing schedule, where feasibility is the most difficult problem. This construction makes sure that the local search is much more likely to be able to find a feasible solution. Applying local search to a construction without this additional help, even one employing the cover heuristic, which is biased towards feasibility, the chance of obtaining a feasible solution on a given dataset is still fairly low. Although Dowsland (1998) showed that feasibility could be obtained with the use of complex, problem-specific neighbourhoods, this thesis has shown that feasibility can be obtained in most cases by using a more aggressive construction in combination with a much simpler local search element.

Overall, this thesis has approached this question of the role of the construction from many different angles and it is possible to conclude that while a good construction is able to balance many different elements and provide solutions which are of a reasonable quality, the idea of a construction works best when combined with a compatible local search. Effectively, the construction builds a good starting solution

for the local search and a trade off exists whereby, the better the starting solution, the simpler the local search required to find high-quality solutions.

7.2 Investigating how different constraints may be balanced within a construction

The role of the construction within a wider solution framework has been discussed, and the best constructions will provide high-quality solutions where little local search is required in order to optimise. And, where a problem has conflicting constraints, it is natural to suppose that the construction should aim to reduce violations of all of these, rather than reducing one set at the expense of another. For the nurse scheduling problem, however, it has been shown that this is not necessarily the case, as will be discussed later in this section. One of the aims of this thesis, as stated in the introduction, was to investigate ways of establishing balance within a constructive approach, such that the final solution obtained is of a reasonable quality with respect to all constraints.

For both problems a number of experiments were carried out initially into ways of balancing the different constraints within the construction. The initial investigation took the form of trying a number of different heuristics, with various weights tested for the costs associated with feasibility and optimality.

The different heuristics each approached the task of balancing constraints in a slightly different way. The first, denoted *Cover* for nurse scheduling and *Feasibility* for medical school scheduling, was concerned only with satisfaction of the feasibility constraints, while *Combined* introduced a weighting system which gave significance to both the hard and soft constraints. Both of these heuristics created a schedule in a set order, taking the nurses, or students, in turn. The *Holistic* and *Last chance* approaches were based on a similar approach of balancing the two sets of constraints, but introduced more flexibility by not using a fixed nurse, or student, ordering and the *Last chance* heuristic incorporated an element of look-ahead, by favouring assignments which, if delayed, could lead to higher costs. The exploitation of the feasibility structure of each problem by use of the knapsack and network flow models provided a further way for balance to be investigated by ensuring that some aspects of

the feasibility constraints were satisfied at each stage of the construction and the success of these exact methods will be discussed further in the next section. It was found that, as expected, heuristics which utilised a score function based both on the feasibility and optimality costs were more likely to produce higher-quality solutions. Although the *Cover* and *Feasibility* heuristics were better at producing feasible solutions, this was offset by the high cost of the solutions they produced and, since a reasonable number of feasible solutions were found by the other heuristics, they showed no overall advantage.

A further way in which balance was investigated was in the way the two parts of the score function were combined, using *Combined*, *Holistic* and *Last chance* for the nurse scheduling problem. Two approaches were tested: an additive and a multiplicative method. The main difference between them is that the multiplicative approach will not allow any assignments to be considered which do not contribute to the feasibility if there are assignments available which will do so. However, the results showed that there was generally not a big difference between the quality of the solutions attained using each of these types of combination, compared with the difference in quality from using a different heuristic. For this reason, the experiments carried out for medical student scheduling did not test both of these methods of combination and instead used just the multiplicative approach, which had been shown to be the more robust of the two.

The last piece of the initial investigation into achieving balance within the construction involved testing the actual weights applied to each aspect of the score function. Clearly, this is one of the most important parts of the investigation, since finding a suitable weighting is almost equivalent to finding a suitable balance. For each problem a range of weights were tested. The weight settings which produced the best results depended on the method of combination, additive or multiplicative, whether or not the feasibility was enhanced using an exact method and, for nurse scheduling, on the metaheuristic used. The method of combination influenced the choice of weights due to the fact that the multiplicative heuristics used the weights as exponents, rather than multipliers and so the same weights applied to each aspect of the problem had a greater impact in this case. When the construction was hybridised with an exact method, this too influenced the weights required to find good solutions.

In most cases where hybridisation was employed, the best results occurred when the weight relating to the soft constraint part of the score was increased relative to the feasibility weight, since finding feasible solutions was no longer such a difficult problem. In terms of the metaheuristic used for nurse scheduling, it was found that the ACO approach, which did not have the benefit of local search, struggled to produce feasible solutions and so the best weight settings were found to be those which favoured feasibility, since these were the most likely to produce feasible solutions.

The main experiments, considering possible ways of balancing the different types of constraints, have been detailed and, as mentioned, the experiments involving the hybridisation of the heuristic methods with the exact techniques to improve the feasibility will be discussed in the next section. However, for the medical student scheduling problem, the exact network flow model was used to create a new type of construction, and this is relevant to the current examination of the ways in which a construction may be used to balance different aspects. Chapter 6 saw the introduction of two network flow construction techniques, *Net-construct* and *Net-lookahead.* In all heuristic approaches for the medical student scheduling problem, weights were used to balance the conflicting aspects of feasibility and optimality, where the soft constraints were combined to provide a single weighted score. *Net-construct* and *Net-lookahead* used a different approach, however; for each timeslot a network flow approach was used to model the feasibility constraints and the hospital and consultant costs only. Although the student costs were necessarily excluded from the model, these construction techniques were able to find the best possible allocation of students to firms for the given timeslot, according to all other constraints. The *Net-lookahead* construction, in particular, was able to incorporate potential future costs and so provide a solution suitable not only for the current timeslot, but which would be likely to reduce the overall cost of the final solution. These methods were shown to be more successful, generally, than the heuristic approaches and the high-quality solution produced in terms of the more important costs was able to compensate for the fact that the student pair costs were not considered in the construction. Thus it has been shown that it can be beneficial to use the construction to minimise just some constraints allow the local search to minimise violations relating to other constraints.

This thesis has considered many ways of balancing different and conflicting constraints using a constructive approach. Although it has been possible to find, for each problem and each metaheuristic approach, constructive techniques which are able to provide the necessary balance, it has also been shown that there is no one method which is guaranteed to provide a good balance in all situations. The weight settings required, for example, will vary depending on how easy each aspect of the problem is and experiments using the nurse scheduling problem showed that a higher weight for feasibility was required when using an ACO approach, with which feasible solutions were less likely to be produced. The approaches developed for the medical student scheduling problem also showed that, while a straightforward heuristic could easily be applied, much better solutions were found when a novel, problem-specific method was adopted. In summary, it has been shown that it is possible to balance constraints within a constructive approach, but that, in order to do so successfully, careful consideration of the design of the construction and the choice of parameters is necessary.

7.3 Investigating how exploiting problem structure with regards to feasibility may improve constraint balance within the construction

The last of the three main aims of the thesis was to investigate the benefits of hybridising the construction with an exact method to ensure that the feasibility constraints, or a relaxed version of the constraints in the case of nurse scheduling, are satisfied at the end of the construction.

The impact of utilising such a procedure is very different for each of the two problems in this thesis. The main difference between the advantages of the knapsack and network flow models was made clear in Chapter 2; while hybridising the medical student scheduling constructions with a network flow model guarantees feasible schedules, the knapsack model only guides the construction towards creating balanced solutions which can more easily be made feasible by the application of local search.

For the nurse scheduling problem, finding feasible solutions has been shown to be particularly difficult and there is no way to guarantee the construction of feasible solutions. However, as has been repeatedly shown throughout this thesis, one of the

main obstacles to finding feasible solutions is the occurrence of unbalanced solutions, where simple neighbourhood moves, such as those used in this thesis, would not be sufficient to correct the imbalance of nurses working days and nights. The use of the knapsack model guaranteed that all solutions would be balanced at the end of the construction, significantly improving the likelihood of feasible solutions being found by the local search. Even using an ACO approach, where local search was not initially incorporated, the number of feasible solutions found was greatly increased using the knapsack model. It can be deduced that the knapsack model is an essential part of any aggressive construction method for the nurse scheduling problem. Furthermore, for the GRASP approach, it was shown that the use of the knapsack allowed the weight associated with the feasibility part of the heuristics' score functions to be reduced, relative to the weight associated with the preference costs. By guaranteeing balanced solutions, the likelihood that the local search would be able to produce feasible schedules was increased, and so the balance in the construction was able to shift more in favour of satisfying the soft constraints.

For the medical student problem, however, finding feasible solutions is not such a difficult problem. Not only can the network flow model be used to guarantee feasibility, but it was shown in Chapter 2 that finding a feasible solution is relatively trivial if a cyclic approach is taken. The problem of finding a feasible solution is therefore much simpler for this problem than for that of nurse scheduling. It is therefore not surprising that, for this problem, the hybridisation of the construction with the network flow algorithm did not significantly influence solution quality. If anything, the constructions were shown to perform slightly better without the use of the network flow and this could be due to the lack of restrictions imposed on the construction; by allowing the heuristic to choose all solution elements, lower-cost solutions may be found and, even if the constructed solution is not feasible, the local search may be able to correct this with minimal impact on the solution cost. Thus it has been shown that, although the addition of the knapsack model was necessary for the creation of high-quality schedules, the difficulty of satisfying the hard constraints of the medical student scheduling problem is not sufficient for the use of the network flow model to be beneficial. However, although exploiting an exact method to produce feasible schedules was not found to be advantageous, exploiting the problem

structure further, to incorporate the hospital and consultant clashes in the model, was found to be a very successful approach.

The success of hybridising a constructive approach with an exact method to exploit the feasibility structure of a given problem has been shown to depend on the difficulty of the original problem. Where nurse scheduling, for which feasibility is a difficult problem, benefits greatly from such an approach, this is not the case for the medical student scheduling problem, for which feasibility is less of an issue. In other cases, whether or not such an approach could, or should, be employed, will depend very much on the problem under investigation and the particular difficulty of solving the hard constraints. However, while the initial aim was to investigate how exploiting problem structure may benefit the feasibility, a broader range of ideas have been implemented, with the inclusion of the network flow constructions, and it is clear that a more general approach to the exploitation of problem structure can be highly beneficial.

7.4 Producing a robust method for each problem, capable of producing high-quality solutions for all problem instances.

Although the main aims of this thesis were related to investigating the successful characteristics of a constructive metaheuristic approach able to balance conflicting constraints, a secondary aim was to find good solution approaches for each problem.

For the nurse scheduling problem, the aim was to produce a method capable of rivalling the best in the literature so far, a tabu search method, by exploiting a construction-based approach. Both ACO and GRASP were applied to the problem and, although both provided reasonable results, finding feasible solutions for all datasets, the GRASP performed significantly better in terms of producing optimal, and near-optimal, solutions. For this reason, further experimentation was carried out using a GRASP approach. The local search was enhanced by allowing a percentage of plateau moves to be accepted and feedback was introduced in the form of the preference cost threshold and knapsack-based diversification. The final method produced excellent results for all 52 datasets and the aim of producing a method to rival that of the tabu search was therefore achieved. Furthermore, it was shown that,

although the additional enhancements allowed all construction heuristics to produce high-quality solutions, the best results were still obtained by utilising the most successful heuristic. And, since utilising the local search with all the enhancements and feedback using a randomly constructed solution gave relatively poor results, it can be determined that the final method produced is one which does rely on the construction to provide a good starting solution. For this problem the aim has therefore been met successfully.

For the medical student scheduling problem it was more difficult to determine the success of a method, since optimal solutions are not known and there are no results in the literature with which to compare the quality of the solutions produced for each dataset. A solution cost for just one dataset appears in the literature and it was shown that the cost obtained in this thesis was a great improvement. For many of the datasets, the same best cost was produced for every run and it is likely that this is the optimal value; even if this is not the case, it shows that the method produced is fairly robust for these problem instances. However, there were several datasets for which the final algorithms struggled to reproduce the best-known cost, including the two datasets for which zero-cost, and therefore optimal, solutions had been found. It can therefore be concluded that although this thesis has produced a method capable of solving the medical student scheduling problem, the quality of the solutions obtained is not known with any certainty and only by employing different techniques for comparative purposes can the success of the GRASP approach be appropriately measured.

7.5 Suggestions for further research

The work presented in this thesis gives rise to many possible directions for future research, both in terms of applying new and enhanced techniques to the problems studied in this thesis and also extending the techniques used here to apply to other problems.

For the nurse scheduling problem, robust metaheuristic methods are available, although neither the construction-based GRASP method presented in this thesis, nor the improvement-based tabu search method, has been able to produce optimal

solutions for all datasets in every run. A common GRASP enhancement is that of path-relinking, as discussed in Chapter 3, and it would be interesting to hybridise this technique with the GRASP developed for nurse scheduling to see whether the quality of the results produced for the more difficult datasets could be improved further, and especially to see whether the computational time required to produce high-quality results could be reduced. To further test the methods developed it would be interesting to extend the problem definition to include a range of more challenging datasets. These could be provided by considering larger problems, where several wards are combined or the planning period is extended, or by including additional constraints, such as those in Burke et al. (2003a), where there are restrictions on which staff may work together. Note that, for the nurse scheduling problem presented here, adding these types of nurse-pairing constraints would result in the necessary inclusion of constraints of the type introduced for dealing with student pairings for the nurse scheduling problem as these could not be included into the shift pattern scores for each nurse. The scores for each nurse's shift pattern would then be altered appropriately depending on the number of shifts which coincide with those assigned to the other nurse. It would also be sensible to think about assigning these types of pairs either consecutively or simultaneously and assigning nurses whose presence will affect several other staff members early on in the process.

There are many cases in the literature where GRASP has been enhanced and hybridising the GRASP with other metaheuristic approaches may also provide an interesting study. For example, GRASP has often been hybridised with other metaheuristics, such as simulated annealing or tabu search and these techniques also may be able to improve the solution method. In particular, the tabu search algorithm developed by Dowsland (1998) has been shown to provide high-quality solutions for the nurse scheduling problem and hybridising the GRASP constructions developed here with the tabu search algorithm, by utilising it in the improvement phase may provide a more efficient and robust method overall.

This thesis has also provided a basic AS approach for solving the nurse scheduling problem and, although the algorithm struggled to compete with the GRASP approach, it would be interesting to give more attention to the range of ACO heuristics available and see whether one of the more advanced ACO approach, such as ACS or MMAS,

would be better placed to provide high-quality solutions. Most ACO algorithms these days do incorporate an element of local search. Further research on this topic could take the form of finding the best possible ACO approach for this problem; with a more advanced ACO algorithm incorporating a more intensive local search it may be possible to produce an ACO approach able to produce optimal solutions for all datasets, and this would certainly be worth further study.

This thesis was also concerned with the relatively new medical school scheduling problem, which has been solved primarily using a GRASP approach. Since this work presents the first known application of a metaheuristic approach to solve this problem, it would be beneficial to apply a wider range of techniques to the datasets introduced here in order to make a comparative study possible. Since the optimal solutions are not known for the datasets provided for this problem, it would be advantageous for the problem to be studied in detail using different approaches in order that an understanding of the difficulty of the problem can be gained and, further, that a comparison of the resulting costs for each dataset can be made. Only by studying this problem further can the success of any solution approach be gauged with any confidence.

An obvious next step in studying this problem would be to apply another metaheuristic approach. It has already been discussed why an ACO approach or any feedback-type techniques may not work well, but an interesting direction would be to apply a tabu search algorithm to this problem. For the nurse scheduling problem, the GRASP and tabu search approaches were the most successful and, in keeping with the comparisons between the two problems drawn throughout this thesis, the application of a tabu search approach would be the logical, and most interesting, next step. The medical student scheduling problem is one which is likely to lend itself well to an improvement-based metaheuristic and so a tabu search approach could be easily implemented. The choice of neighbourhoods and whether or not the inclusion of chain-type neighbourhoods is necessary would be topics covered by such a study.

Aside from the implementation of a tabu search approach, there are several other metaheuristic approaches, in particular improvement-based metaheuristics, which could easily be applied to medical student scheduling. It was described how the

memory approach applied in Chapter 6 is similar to that of iterated local search (ILS) and, since this was such a successful technique, it would be interesting to apply other ILS approaches, perhaps using different types of 'perturbing' moves. Simulated annealing is another example of an improvement-based metaheuristic which could easily be applied to this problem.

Other interesting aspects include further study of the importance of the assignments in the first timeslot. Thus far, the firms in the initial timeslot have been filled in a straightforward manner and these assignments have not been subject to change even using local search. It may be that allowing moves in this timeslot enables a more efficient search of the solution space, since making one change in this timeslot could be equivalent to a succession of several moves in other timeslots.

It would also be interesting to see whether further research would enable some of the symmetry of this problem to be removed, thus potentially allowing feedback-type approaches to be applied. Other suggestions for further study include ways to focus specifically on minimising the student-pair constraint violations, since these have not been studied in as much depth as the other constraints. This could include trying to find a way to incorporate the student costs into a network flow type approach and finding neighbourhoods which deal exclusively with trying to minimise the student costs. Certainly, further work could be done identifying key difficulties with the medical student scheduling problem and, although optimal costs are not known, it may be possible to obtain bounds for each dataset. It is clear that the medical student scheduling problem provides an interesting topic for further study for which a great deal of further research is possible.

Finally, this thesis has introduced interesting techniques for solving problems with conflicting constraints and it would be worthwhile applying these types of approaches to other, similar, problems. One such example would be the operating theatre scheduling problem, which requires that a set of limited hospital resources be assigned to jobs for the given planning period, in order to maximise the use of hospital resources such that tasks are performed according to their needs and priorities.

Many of the enhancements developed for nurse scheduling were problem-specific, however, the GRASP enhancements introduced for the medical student scheduling problem, such as using partial restarts and considering large portions of the construction at once rather than considering all assignments individually, are ideas which could potentially be applied to other problems. It would therefore be interesting to find similar problems in the literature and determine how the techniques applied in this thesis may be adapted to solve them successfully.

Bibliography

Abdennadher, S., & Schlenker, H. (1999). Nurse Scheduling Using Constraint Logic Programming. In *Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, IAAI-99*. Orlando, Florida.

Abramson, D. (1991). Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms. *Management Science, 37*(1), 98.

Ahmadi, S., & Osman, I. H. (2005). Greedy random adaptive memory programming search for the capacitated clustering problem. *European Journal of Operational Research, 162*(1), 30-44.

Ahuja, H., & Sheppard, R. (1975). Computerised nurse scheduling. *Industrial Engineering, 7*, 24-29.

Aickelin, U. (1999). *Genetic Algorithms for Multiple-Choice Problems.* University of Wales Swansea, UK.

Aickelin, U., Burke, E., & Li, J. (2007a). An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society, 58*, 1574–1585, *doi: 10.1057/palgrave.jors.2602308.*

Aickelin, U., Burke, E., & Li, J. (2007b). Solving Personnel Scheduling through Improved Squeaky Wheel Optimisation. *(under review) IEEE Transactions on Evolutionary Computation.*

Aickelin, U., & Dowsland, K. A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *J. Sched., 3*(3), 139-153.

Aickelin, U., & Dowsland, K. A. (2004). An indirect Genetic Algorithm for a nurse-scheduling problem. *Computers & Operations Research, 31*(5), 761-778.

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

Aickelin, U., & White, P. (2004). Building Better Nurse Scheduling Algorithms. *Annals of Operations Research, 128*(1 - 4), 159-177.

Aiex, R., Resende, M. G. C., Pardalos, P. M., & Toraldo, G. (2005). GRASP with Path Relinking for Three-Index Assignment. *INFORMS Journal on Computing, 17*(2), 224-247.

Aiex, R. M., Binato, S., & Resende, M. G. C. (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing, 29*(4), 393-430.

Andronescu, M., & Rastegari, B. (2003). *Motif-GRASP and Motif-ILS: Two New Stochastic Local Search Algorithms for Motif Finding*: Computer Science Department, University of British Colombia, Vancouver, Canada.

Areibi, S., & Vannelli, A. (2000). *Efficient hybrid search techniques for circuit partitioning.* Paper presented at the IEEE 4th World Multiconference on Circuits, Systems, Communications and Computers.

Argüello, M. F., Feo, T. A., & Goldschmidt, O. (1996). Randomized methods for the number partitioning problem. *Computers & Operations Research, 23*(2), 103-111.

Atkinson, J. (1998). A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *Journal of the Operational Research Society, 49*(7), 700-708.

Bard, J., & Purnomo, H. W. (2005a). Short-term nurse-scheduling in response to daily fluctuations in supply and demand. *Health Care Management Science, 8*, 315-324.

Bard, J. F., & Purnomo, H. W. (2007). Cyclic preference scheduling of nurses using a Lagrangian-based heuristic. *Journal of Scheduling, 10*(1), 5-23.

Bard, J. F., & Purnomo, H. W. (2005b). Preference scheduling for nurses using column generation. *European Journal of Operational Research, 164*, 510-534.

Beddoe, G. R., & Petrovic, S. (2003). A novel approach to finding feasible solutions to personnel rostering problems, *Proceedings of the 14th Annual Conference of the Production an Operation Management Society.* Savannah, Georgia, United States.

Beddoe, G. R., & Petrovic, S. (2005). Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *European Journal of Operational Research, 175(2)*, 649-671.

Bellanti, F., Carello, G., Della Croce, F., & Tadei, R. (2004). A greedy-based neighbourhood search approach to a nurse rostering problem. *European Journal of Operational Research, 153*(1), 28-40.

Beltrán, J. D., Calderón, J. E., Cabrera, R. J., Moreno-Pérez, J. A., & Moreno-Vega, J. M. (2004). *GRASP/VNS hybrid for the Strip Packing Problem.* Paper presented at the Hybrid Metaheuristics, First International Workshop, HM 2004, Valencia, Spain.

Berrada, I., Ferland, J. A., & Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences, 30*(3), 183-193.

Binato, S., Hery, W. J., Loewenstern, D. M., & Resende, M. G. C. (2002). A GRASP for job shop scheduling. *Essays and surveys in metaheuristics (Angra dos Reis, 1999), 15*, 59-79.

Blum, C., & Roli, R. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268-308.

Bresina, J. L. (1996). *Heuristic-biased stochastic sampling.* Paper presented at the Proceedings of the AAAI-96.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999a). An improved Ant System algorithm for the Vehicle Routing Problem. *Annals of Operations Research, 89*(0), 319-328.

Bullnheimer, B., Hartl, R., & Strauss, C. (1999b). A New Rank Based Version of the Ant System - A Computational Study. *Central European Journal of Operations Research, 7*(1), 25-38.

Burke, E., Cowling, P., de Causmaeker, P., & Vanden Berghe, G. (2001). A memetic approach to the nurse rostering problem. *Applied Intelligence, 15*, 199-214.

Burke, E. K., De Causmaecker, P., Petrovic, S., & Vanden Berghe, G. (2002). A multi criteria meta-heuristic approach to nurse rostering. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)* (Vol. 2, pp. 1197-1202): IEEE.

Burke, E. K., De Causmaecker, P., Petrovic, S., & Vanden Berghe, G. (2003a). Variable Neighbourhood Search for Nurse Rostering Problems. In M. G. C. Resende & J. P. de Sousa (Eds.), *METAHEURISTICS: Computer Decision-Making* (pp. 153-172): Kluwer.

Burke, E. K., De Causmaecker, P., & Vanden Berghe, G. (1999). A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. In B. McKay (Ed.), *Simulated Evolution and Learning* (Vol. 1585, pp. 187-194): Springer.

Burke, E. K., De Causmaecker, P., & Vanden Berghe, G. (2004b). Novel metaheuristic Approaches to Nurse Rostering Problems in Belgian Hospitals. In J. Leung (Ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis* (pp. 1-18 Ch.44): CRC Press.

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

Burke, E. K., Kendall, G., & Soubeiga, E. (2003b). A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics, 9*(6), 451-470.

Carreto, C., & Baker, B. (2002). A GRASP Interactive Approach for the Vehicle Routing Problem with Backhauls. In C. Ribeiro & P. Hansen (Eds.), *Essays and Surveys in Metaheuristics* (pp. 185-200): Kluwer Academic Publishers.

Casey, S., & Thompson, J. (2003). GRASPing the examination scheduling problem. *In·E. Burke and P. De Causmaecker (Eds.), PATAT 2002, Vol. 2740 of Lecture Notes in Computer Science*, 232-244.

Charles, F., & Fred, G. (1999). Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory. *INFORMS Journal on Computing, 11*(2), 198.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems – a bibliographic survey. *European Journal of Operational Research, 151*(3), 447-460.

Chen, H., & Cheng, A. M. K. (2005). Applying Ant Colony Optimization to the partitioned scheduling problem for heterogeneous multiprocessors. *SIGBED Review, 2*(2), 11-14.

Cheng, B. M. W., Lee, J. H. M., & Wu, J. C. K. (1997). A nurse rostering system using constraint programming and redundant modelling. In *Transactions in Information Technology in Biomedicine* (Vol. 1, pp. 44-54): IEEE.

Corberan, A., Marti, R., & Sanchis, J. M. (2002). A GRASP heuristic for the mixed Chinese postman problem. *European Journal of Operational Research, 142*(1), 70-80.

Cordón, O., Herrera, F., & Stützle, T. (2002). A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. *Mathware and Soft Computing 9.*

Costa, D., & Hertz, A. (1997). Ants can colour graphs. *Journal of the Operational Research Society, 48*, 295-305.

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

de Aragão Trindade, V., & Ochi, L. S. (2005). *Hybrid Adaptive Memory Programming Using GRASP and Path Relinking for the Scheduling Workover Rigs for Onshore Oil Production* Paper presented at the Fifth International Conference on Hybrid Intelligent Systems (HIS'05), Rio de Janeiro, Brazil.

De Causmaecker, P., & Vanden Berghe, G. (2003). Relaxation of Coverage Constraints in Hospital Personnel Rostering. *Lecture Notes in Computer Science, 2740*, 129-147.

Delorme, X., Gandibleux, X., & Rodriguez, J. (2004). GRASP for set packing problems. *European Journal of Operational Research, 153*(3), 564-580.

Dias, T. M., Ferber, D. F., de Souza, C. C., & Moura, A. V. (2003). Constructing nurse schedules at large hospitals. *International Transactions in Operational Research, 10*(3), 245-265.

Dorigo, M. (1992). *Optimisation, Learning and Natural Algorithms (in Italian).* Politecnico di Milano.

Dorigo, M., Di Caro, G., & Gambardella, L. M. (1999). Ant Algorithms for Discrete Optimization. *Artificial Life, 5*, 137-172.

Dorigo, M., & Gambardella, L. M. (1996). Ant colonies for the traveling salesman problem. TR/IRIDIA/1996-3 Université Libre de Bruxelles.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on, 1*(1), 53-66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Ant System: An Autocatalytic Optimizing Process*: Technical report No. 91-016 Politechnico di Milano.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The Ant System: Optimization by a colony of cooperating agents *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, *26*(1), 1-13.

Dorigo, M., & Stützle, T. (2003). The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics*: Kluwer Academic Publishers.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*: MIT Press.

Dowsland, K. A. (1998). Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, *106*(2-3), 393-407.

Dowsland, K. A. (2005). Classical Techniques. In E. Burke & G. Kendall (Eds.), *Search Methodologies - Introductory tutorials in optimization and decision support techniques* (pp. 19-68): Springer.

Dowsland, K. A., & Thompson, J. M. (2000). Solving a nurse-scheduling problem with knapsacks, networks and tabu search. *Journal of the Operational Research Society*, *51*, 825-833

Dowsland, K. A., & Thompson, J. M. (2005). Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, *56*(4), 426-438.

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

Feo, T. A., Bard, J. F., & Holland, S. D. (1996a). A GRASP for scheduling printed wiring board assembly. *IIE Transactions*, *28*(2), 155-165.

Feo, T. A., & González-Velarde, J. L. (1995). The Intermodal Trailer Assignment Problem. *Transportation Science*, *29*(4), 330-341.

Feo, T. A., & Resende, M. G. C. (1989). A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, *8*, 67-71.

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*, 109-133.

Feo, T. A., Resende, M. G. C., & Smith, S. H. (1994). A greedy randomised adaptive search procedure for maximum independent set. *Operations Research*, *42*, 860-878

Feo, T. A., Sarathy, K., & McGahan, J. (1996b). A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, *23*(9), 881-895.

Feo, T. A., Venkatraman, K., & Bard, J. (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research, 18*(8), 635-643.

Ferland, J. A., Berrada, I., Nabli, I., Ahiod, B., Michelon, P., Gascon, V., & Gagné, É. (2001). Generalized Assignment Type Goal Programming Problem: Application to Nurse Scheduling. *Journal of Heuristics, 7*, 391-413.

Ferretti, I., Zanoni, S., & Zavanella, L. (2006). Production-inventory scheduling using Ant System metaheuristic. *International Journal of Production Economics, 104*, 317-326.

Festa, P., & Resende, M. G. C. (2004). *An annotated bibliography of GRASP*: Technical Report TD-5WYSEW AT&T Labs Research.

Fleurent, C., & Glover, F. (1999). Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory. *INFORMS Journal on Computing, 11*(2), 198-204.

Forsyth, P., & Wren, A. (1997). *An ant system for bus driver scheduling*: Research Report 97.25 University of Leeds School of Computer Studies.

Frieze, A. M. (1983). Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research, 13*(2), 161-164.

Fügenschuh, A., & Höfler, B. (2006). Parametrized GRASP Heuristics for Three-Index Assignment. *Lecture Notes in Computer Science, 3906*, 61-72.

Fuller, E. J. (1998). *Tackling scheduling problems using integer programming.* University of Wales, Swansea.

Gagné, C., Gravel, M., & Price, W. L. (2006). Solving real car sequencing problems with ant colony optimisation. *European Journal of Operational Research, 174*, 1427-1448.

Gambardella, L., & Dorigo, M. (1995). *Ant-Q: A Reinforcement Learning approach to the traveling salesman problem.* Paper presented at the Twelfth International Conference on Machine Learning.

Gambardella, L., & Dorigo, M. (1996). *Solving Symmetric and Asymmetric TSPs by Ant Colonies.* Paper presented at the IEEE Conference on Evolutionary Computation (ICEC'96), Nagoya, Japan.

Gambardella, L. M., Taillard, E. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society, 50*(2), 167-176.

García, J. M., Smith, K., Lozano, S., & Guerrero, F. (2001). A Comparison of GRASP and an exact method for solving a production and delivery scheduling problem. *Computers & Industrial Engineering, 48(4)*, 733-742.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*: Freeman.

Gendreau, M., & Potvin, J.-Y. (2005). Tabu Search. In E. Burke & G. Kendall (Eds.), *Search Methodologies* (pp. 165-186): Springer.

Gilbert, K. C., & Hofstra, R. B. (1987). An Algorithm for a Class of Three-Dimensional Assignment Problems Arising in Scheduling Applications. IIE Transactions, 19(1), 29-33.

Gilbert, K. C., & Hofstra, R. B. (1988). Multidimensional assignment problems. *Decision Sciences, 19*(2), 306-321.

Glover, F., & Laguna, M. (1995). Tabu Search. In C. R. Reeves (Ed.), *Modern heuristic techniques for combinatorial problems* (pp. 70-141): McGraw-Hill.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research, 13*, 533-549.

Gomes, A., & Oliveira, J. (2001). A GRASP approach to the nesting problem, *Proceedings of MIC'2001* (pp. 47-52).

Gravel, M., Price, W. L., & Gagné, C. (2002). Scheduling continuous casting of aluminium using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Research, 143*, 218-229.

Gupta, S. R., & Smith, J. S. (2006). Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research, 175*(2), 722-739.

Gutjahr, W. J., & Rauner, M. S. (2007). An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. *Computers & Operations Research, 34*(3), 642-666.

Hahn, P. M., Kim, B. J., Stützle, T., Kanthak, S., Hightower, W. L., Samra, H., Ding, Z., & Guignard, M. (2008). The quadratic three-dimensional assignment problem: Exact and approximate solution methods. *European Journal of Operational Research, 184*, 416-428.

Harris, R. A. (1997). Private communication.

Hart, J. P., & Shogan, A. W. (1987). Semi-greedy heuristics: An Empirical Study. *Operations Research Letters, 6*(3), 107-114.

Hofstra, R. B. (1983). *Multidimensional assignment and scheduling with application to a tourism scheduling problem.* The University of Tennessee.

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

Isken, M. W. (2004). An Implicit Tour Scheduling Model with Applications in Healthcare. *Annals of Operations Research, 128*(1 - 4), 91-109.

Isken, M. W., & Hancock, W. M. (1991). A heuristic approach to nurse-scheduling in hospital units with non-stationary, urgent demand, and a fixed staff size. *Journal of the Society for Health Systems*, *2*(2), 24-41.

Jan, A., Yamamoto, M., & Ohuchi, A. (2000). Evolutionary algorithms for nurse scheduling problem, *Proceedings of the 2000 Congress on Evolutionary Computation CEC00* (pp. 196-203): IEEE Press.

Jaumard, B., Semet, F., & Vovor, T. (1998). A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, *107*(1), 1-18.

Kim, K. H., & Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of Operational Research*, *156*(3), 752-768.

Klincewicz, J. G. (1992). Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, *40*, 283-302.

Kontoravdis, G., & Bard, J. (1995). A GRASP for the Vehicle Routing Problem with Time Windows. *ORSA Journal on Computing*, *7*(1), 10-23.

Laguna, M., & González-Velarde, J. L. (1991). A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, *2*(4), 253-260.

Laguna, M., & Martí, R. (2001). A GRASP for Coloring Sparse Graphs. *Computational Optimization and Applications*, *19*(2), 165-178.

Laguna, M., & Martí, R. (1999). GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, *11*(1), 44-52.

Levine, J., & Ducatelle, F. (2004). Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, *55*(7), 705-716.

Li, J., & Aickelin, U. (2003). A Bayesian Optimization Algorithm for the Nurse Scheduling Problem. In *Proceedings of 2003 Congress on Evolutionary Computation (CEC2003)* (pp. 2149-2156). Canberra, Australia: IEEE Press.

Li, J., & Aickelin, U. (2004). The application of Bayesian Optimization and Classifier Systems in Nurse Scheduling. In *Proceedings 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII), LNCS 3242* (pp. 581-590). Birmingham, UK.

Li, Y., Pardalos, P. M., & Resende, M. G. C. (1994). A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *16*, 237-261.

Liu, X., Pardalos, P. M., Rajasekaran, S., & Resende, M. G. C. (2000). A GRASP for frequency assignment in mobile radio networks. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *52*, 195-201.

Lourenço, H., Paixão, J., & Portugal, R. (2001). Multiobjective Metaheuristics for the Bus-Driver Scheduling Problem. *Transportation Science, 35*(3), 331-343.

Lourenço, H., & Serra, D. (1998). *Adaptive Approach Heuristics for the Generalized Assignment Problem*: Technical Report 288, Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain.

Maenhout, B., & Vanhoucke, M. (2006). New Computational Results for the Nurse Scheduling Problem: A Scatter Search Algorithm. *Lecture Notes in Computer Science, 3906*, 159-170.

Magos, D. (1996). Tabu Search for the Planar Three-Index Assignment Problem. *Journal of Global optimization, 8*, 35-48.

Mäkinen, J., Miettinen, K., & Mäkelä, M. M. (1999). Some Penalty Methods with Genetic Algorithms. In K. Miettinen, M. M. Mäkelä & J. Toivanen (Eds.), *Proc. EUROGEN'99, Short Course on Evolutionary Algorithms in Engineering and Computer Science* (Vol. A2, pp. 105-112).

Maniezzo, V. (1998). *Exact and Approximate Nondeterministic Tree-search Procedures for the Quadratic Assignment Problem*: Computer Science, Report CSR 98-1 University of Bologna.

Maniezzo, V., & Colorni, A. (1999). The Ant System Applied to the Quadratic Assignment Problem. *Knowledge and Data Engineering, 11*(5), 769-778.

Martello, S., & Toth, P. (1990). *Knapsack Problems*: Wiley, Chichester.

Mavridou, T., Pardalos, P. M., Pitsoulis, L. S., & Resende, M. G. C. (1998). A GRASP for the biquadratic assignment problem. *European Journal of Operational Research, 105*(3), 613-621.

Merkle, D., & Middendorf, M. (2001). A New Approach to Solve Permutation Scheduling Problems with Ant Colony Optimization. *Lecture Notes in Computer Science, 2037*, 484-493.

Meyer auf'm Hofe, H. (2001). Solving Rostering Tasks as Constraint Optimization. In *Lecture Notes in Computer Science* (Vol. 2079, pp. 191-212): Springer.

Millar, H. H., & Kiragu, M. (1998). Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European Journal of Operational Research, 104*(3), 582-592.

Montgomery, J., Randall, M., & Hendtlass, T. (2004). Search bias in constructive meta-heuristics and implications for ant colony optimisation. In *Lecture Notes in Computer Science: ANTS 2004.* (Vol. 3172, pp. 390-397): Springer.

Montgomery, J., Randall, M., & Hendtlass, T. (2005). Structural Advantages for Ant Colony Optimisation Inherent in Permutation Scheduling Problems. *Lecture Notes in Computer Science, 3533*, 218-228.

Moura, A., & Oliveira, J. F. (2005). A GRASP Approach to the Container-Loading Problem. *IEEE Intelligent Systems, 20*(4), 50-57.

Moz, M., & Vaz Pato, M. (2007). A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research, 34*, 667-691.

Naidu, K. D., Sullivan, K. M., Wang, P. P., & Yang, Y. (2000). Managing Personnel through Staff Scheduling Algorithms. In *Proceedings of the Fifth Joint Conference on Information Sciences (JCIS'00)* (Vol. 2, pp. 829-835).

Oliveira, C., Pardalos, P. M., & Resende, M. G. C. (2004). GRASP with path-relinking for the quadratic assignment problem. *Lecture Notes in Computer Science, 3059*, 356-368.

Osogami, T., & Imai, H. (2000). Classification of Various Neighbourhood Operations for the Nurse Scheduling Problem. *Lecture Notes in Computer Science, 1969*, 72-83.

Özcan, E. (2005). *Memetic Algorithms for Nurse Rostering.* Paper presented at the 20th International Symposium on Computer and Information Sciences.

Pardalos, P. M., Qian, T., & Resende, M. G. C. (1999). A Greedy Randomized Adaptive Search Procedure for the Feedback Vertex Set Problem. *Journal of Combinatorial Optimization, 2*, 399-412.

Parr, D., & Thompson, J. M. (2007). Solving the multi objective nurse scheduling problem with a weighted cost function. *Annals of Operations Research 155*(1), 279-288.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research, 176*, 774-793.

Piñana, E., Plana, I., Campos, V., & Marti, R. (2004). GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research, 153*(1), 200-210.

Pitsoulis, L., & Resende, M. G. C. (2001). Greedy randomized adaptive search procedures. In P. M. Pardalos & M. G. C. Resende (Eds.), *Handbook of Applied Optimization* (pp. 168-181).

Randall, M. (2004). Heuristics for Ant Colony Optimisation and the Generalised Assignment Problem, *Proceedings of the Congress of Evolutionary Computing* (pp. 1916-1923): IEEE Press.

Randhawa, S. U., & Sitompul, D. (1993). A heuristic-based computerized nurse scheduling system. *Computers & Operations Research, 20*(8), 837-844.

Resende, M. G. C. (2001). *Greedy Randomized Adaptive Search Procedures (GRASP)* (Vol. 2): Kluwer Academic Press.

Resende, M. G. C., & Feo, T. A. (1996). A GRASP for satisfiability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 26*, 499-520.

Resende, M. G. C., & Festa, P. (2003). *An updated bibliography of GRASP*: Technical Report TD-5SB7BK AT&T Labs Research.

Resende, M. G. C., & Ribeiro, C. C. (2002). Greedy randomized adaptive search procedures. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 219-249): Kluwer Academic Publishers.

Ribeiro, C., Uchoa, E., & Werneck, R. (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing;*, *14*(3), 228-246.

Ritchie, G. (2003). *Static Multi-processor Scheduling with Ant Colony Optimisation & Local Search.* University of Edinburgh, Edinburgh.

Robertson, A. J. (2001). A Set of Greedy Randomized Adaptive Local Search Procedure (GRASP) Implementations for the Multidimensional Assignment Problem *Computational Optimization and Applications*, *19*(2), 145-164.

Rojanasoonthon, S., & Bard, J. (2005). A GRASP for Parallel Machine Scheduling with Time Windows. *INFORMS Journal on Computing*, *17*(1), 32-51.

Rosenbloom, E. S., & Goertzen, N. F. (1987). Cyclic nurse scheduling. *European Journal of Operational Research*, *31*(1), 19-23.

Sastry, K., Goldberg, D., & Kendall, G. (2005). Genetic Algorithms. In E. Burke & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 97-125): Springer.

Scott, S., & Simpson, R. (1998). Case-bases incorporating scheduling constraint dimensions - experiences in nurse rostering. *Lecture Notes in Computer Science*, *1488*, 392-401.

Skorin-Kapov, N., & Kos, M. (2006). A GRASP Heuristic for the Delay-Constrained Multicast Routing Problem. *Telecommunication Systems*, *32*(1), 55-69.

Socha, K., Knowles, J., & Sampels, M. (2002). A MAX-MIN Ant System for the University Timetabling Problem. In M. Dorigo, G. Di Caro & M. Sampels (Eds.), *Proceedings of ANTS 2002 -Third International Workshop on Ant Algorithms* (Vol. 2463, pp. 1-13). Brussels, Belgium: Springer-Verlag.

Socha, K., Sampels, M., & Manfrin, M. (2003). Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In *Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization* (Vol. 2611): Springer-Verlag.

Stützle, T., & Dorigo, M. (1999a). ACO Algorithms for the Quadratic Assignment Problem. In D. Corne, M. Dorigo & F. Glover (Eds.), *New Ideas in Optimization* (pp. 33-50). London: McGraw-Hill.

Stützle, T., & Dorigo, M. (1999b). ACO algorithms for the traveling salesman problem. In K. Miettinen, M. M. Mäkelä, P. Neittaanmäki & J. Periaux (Eds.), *Evolutionary Algorithms in Engineering and Computer Science* (pp. 163-183). Chichester, UK: John Wiley & Sons.

Stützle, T., & Hoos, H. H. (1997). MAX-MIN Ant System and Local Search for the Traveling Salesman Problem In *Proceedings of the Fourth International Conference on Evolutionary Computation (ICEC'97)* (pp. 308-313): IEEE Press.

Stützle, T., & Hoos, H. H. (2000). Max-Min Ant System. *Future Generation Computer Systems*, *16*(8), 889-914.

Taillard, E. D. (1999). *Ant Systems*: IDSIA-05-99 Istituto Dalle Molle di Studi sull'Intelligenza Artificiale.

Tanomaru, J. (1995). Staff scheduling by a Genetic Algorithm with Heuristic Operators, *Proceedings of the IEEE Conference on Evolutionary Computation* (pp. 456-461).

Thompson, J. M., & Dowsland, K. A. (1996). Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research*, *63*, 105-128

Tsai, J.-F., Lin, M.-H., & Hu, Y.-C. (2008). Finding multiple solutions to general integer linear programs. *European Journal of Operational Research*, *184*, 802-809.

Valouxis, C., & Housos, E. (2000). Hybrid optimization techniques for the workshift and rest assignment of nursing personnel. *Artificial Intelligence in Medicine*, *20*(2), 155-175.

Voß, S. (2000). Heuristics for nonlinear assignment problems. In P. M. Pardalos & L. Pitsoulis (Eds.), *Nonlinear Assignment Problems* (pp. 175-215): Kluwer Academic Publishers.

Warner, D. M. (1976). Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach. *Operations Research*, *24*, 842-856.

Wright, M. (1996). School timetabling using heuristic search. *Journal of the Operational Research Society*, *47*, 347-357

Wright, P. D., Bretthauer, K. M., & Côté, M. J. (2006). Reexamining the nurse scheduling problem: staffing ratios and shortages. *Decision Sciences*, *37*(1), 5-38.

Xu, J., & Chiu, S. Y. (2001). Effective Heuristic Procedures for a Field Technician Scheduling Problem. *Journal of Heuristics*, *7*(5), 495-509.

Appendix A

Nurse scheduling datasets

Chapters 4 and 5 perform experiments using 52 known datasets. The details of these datasets are given in this appendix. For each dataset, the number of nurses, along with the numbers of each type of nurse, is given for each of the three grades. For each type of nurse, a number of these may be forced to work either days or nights, depending on their preference costs. Since any pattern with a preference cost of 100 is deemed unacceptable and is disallowed, a nurse for whom all day patterns are voided in this way must work nights and vice versa.

The data in Table A.1 is displayed in the following manner:

		(5,4)			(4,3)		
19	23	4	1	0	3	1	0
		6	0	1	0	0	0
		3	0	1	1	0	0
20	23	3	0	1	3	2	0
		4	0	1	1	0	0
		3	0	1	1	0	0

The top row gives the nurse type and it must be noted that the 11 nurse types which arise do not all occur in every dataset; most datasets include only about half this number. The first two columns give the number of the dataset and the total number of nurses available in the dataset, respectively. Note that (0,0) nurses do not work any shifts, but are still included in this total number of nurses. For each dataset, we have a 3×3 grid giving information for each type of nurse. This information is organised as follows. For each dataset, the three rows represent the three grades, with the top row representing grade 1. The three columns for each type indicate: the number of nurses of that type and grade, and the number of these forced to work days and nights respectively.

Thus dataset 19 has 23 nurses in total and four of type (4,3). Of these, three are grade 1 and one is grade 3. Of the three grade 1 nurses, one must work days. Clearly, for types (1&3), (1&2) and (0,0) the last two columns are not necessary since these nurses cannot be forced to work just days or nights.

Table A.2 gives details of the cover requirements of each dataset.

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
1	26	6	0	0	3	0	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		7	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	27	6	0	0	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	2
		3	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	27	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
		4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	26	8	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
		3	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
5	26	8	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	24	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	3
		3	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
7	23	7	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0
		3	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
8	24	8	3	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	0
		3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1

Table A.1. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
9	24	7	3	1	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	1
		3	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
10	24	4	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		5	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
11	25	5	2	2	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		6	0	0	2	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12	25	5	2	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	0	3	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
13	25	4	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0
		4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	2
14	22	4	2	1	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
		3	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		3	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1
15	21	4	1	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
		2	0	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		3	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	22	4	2	1	3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
		2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		3	0	1	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Table A.1 contd. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
17	23	3	'1	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2
		2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	1	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
18	22	4	2	2	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
		4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		2	0	1	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
19	23	4	1	0	3	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		6	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		3	0	1	1	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
20	23	3	0	1	3	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3	0	1	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
21	23	2	1	0	5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
		5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
22	26	5	2	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		6	0	0	3	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
23	26	3	1	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		6	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
24	23	3	2	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		3	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.1 contd. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
25	23	4	2	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	2	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	26	3	0	0	2	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		7	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3
27	26	3	1	0	4	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
		3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		6	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3
28	27	5	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
		3	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	4
29	24	4	1	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
		3	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		5	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2
30	27	6	2	1	2	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	2
		3	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		7	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
31	28	8	2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
		4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		9	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
32	27	8	3	1	2	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
		3	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1

Table A.1 contd. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
33	27	8	3	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
		4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		7	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
34	23	7	3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	1	0
		3	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
35	24	8	3	1	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	0
		3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1
36	24	7	3	1	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	1
		3	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
37	25	5	2	2	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		6	0	0	2	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
38	25	5	2	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	0	3	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
39	25	4	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0
		4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	2
40	21	4	1	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
		2	0	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		3	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.1 contd. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
41	22	4	2	1	3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
		2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		3	0	1	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
42	23	3	1	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2
		2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	1	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
43	22	4	2	2	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
		4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		2	0	1	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
44	23	3	0	1	3	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3	0	1	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
45	23	2	1	0	5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
		5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
46	26	3	1	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		6	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
47	23	3	2	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
		3	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	23	4	2	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		4	0	0	2	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.1 contd. The number of nurses of each type and grade available for each dataset. (Continued on next page).

		(5,4)			(4,3)			(3,2)			(2,1)			(4,4)			(3,3)			(2,2)			(1,1)			(1&3)	(1&2)	(0,0)
49	26	3	1	0	4	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
		3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		4	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	3
50	27	5	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
		3	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	4
51	24	4	1	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
		3	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
		5	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2
52	23	5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.1 contd. The number of nurses of each type and grade available for each dataset.

The cover requirements for each day and grade are given for each dataset in Table A.2. Note that for all datasets, the requirement for the night shifts is the same and the day requirement at grades 1 and 2 is also unaltered. However, the overall requirement for each day shift varies between datasets. For completeness, the details of all requirements at all grades are given below.

	Days								Nights							
	S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
1	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	13	13	13	13	13	13	13	91	3	3	3	3	3	3	3	21
2	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	11	11	11	11	11	11	11	77	3	3	3	3	3	3	3	21
3	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	12	12	12	12	12	12	12	84	3	3	3	3	3	3	3	21
4	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	11	11	11	11	11	11	11	77	3	3	3	3	3	3	3	21
5	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	12	12	12	12	12	12	12	84	3	3	3	3	3	3	3	21
6	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
7	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	9	9	9	9	9	9	63	3	3	3	3	3	3	3	21
8	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21
9	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21
10	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
11	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21

Table A.2. Details of the cumulative cover requirements for each day and grade for each dataset.
(Continued on next page).

	Days								Nights							
	S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
12	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
13	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
14	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
15	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
16	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
17	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
18	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
19	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
20	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	8	8	8	8	8	8	56	3	3	3	3	3	3	3	21
21	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
22	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	11	11	11	11	11	9	73	3	3	3	3	3	3	3	21
23	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	10	10	10	10	10	8	66	3	3	3	3	3	3	3	21
24	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	9	9	9	9	9	7	59	3	3	3	3	3	3	3	21

Table A.2 contd. Details of the cumulative cover requirements for each day and grade for each dataset.
(Continued on next page).

	Days								Nights							
	S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
25	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
26	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	10	10	10	10	10	7	64	3	3	3	3	3	3	3	21
27	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	10	10	10	10	10	7	64	3	3	3	3	3	3	3	21
28	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
29	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
30	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	11	12	12	12	12	12	11	82	3	3	3	3	3	3	3	21
31	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	12	13	13	13	13	13	12	89	3	3	3	3	3	3	3	21
32	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	12	12	12	12	12	12	12	84	3	3	3	3	3	3	3	21
33	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	12	12	12	12	12	12	12	84	3	3	3	3	3	3	3	21
34	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	9	9	9	9	9	9	63	3	3	3	3	3	3	3	21
35	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21
36	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21
37	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	11	11	11	11	11	10	75	3	3	3	3	3	3	3	21

Table A.2 contd. Details of the cumulative cover requirements for each day and grade for each dataset.
(Continued on next page).

	Days								Nights							
	S	M	T	W	T	F	S	Tot.	S	M	T	W	T	F	S	Tot.
38	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
39	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
40	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
41	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
42	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
43	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	9	10	10	10	10	10	9	68	3	3	3	3	3	3	3	21
44	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	8	8	8	8	8	8	56	3	3	3	3	3	3	3	21
45	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
46	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	10	10	10	10	10	8	66	3	3	3	3	3	3	3	21
47	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	9	9	9	9	9	7	59	3	3	3	3	3	3	3	21
48	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	8	9	9	9	9	9	8	61	3	3	3	3	3	3	3	21
49	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
50	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21

Table A.2 contd. Details of the cumulative cover requirements for each day and grade for each dataset.
(Continued on next page).

51	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	7	8	8	8	8	8	7	54	3	3	3	3	3	3	3	21
52	2	2	2	2	2	2	2	14	1	1	1	1	1	1	1	7
	4	4	4	4	4	4	4	28	2	2	2	2	2	2	2	14
	10	10	10	10	10	10	10	70	3	3	3	3	3	3	3	21

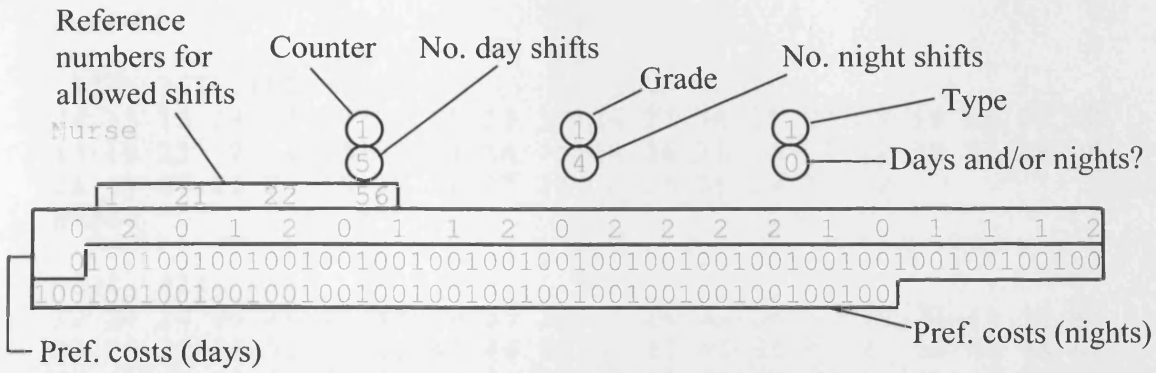
Table A.2 contd. Details of the cumulative cover requirements for each day and grade for each dataset.

A.1 Sample dataset

The following is an example of the raw data supplied for one of the 52 datasets.

The first 3×14 set of numbers indicate the cover requirements. Each row represents the cumulative cover required at each grade with the first row representing grade 1. The 14 columns each represent a single shift, with columns 1 – 7 representing Sunday – Saturday days and columns 8 – 14 representing Sunday – Saturday nights.

The details are then provided for each nurse in turn. The first row provides three numbers. The first of these indicates the nurse counter, the second number provides the nurse's grade and the third provides a number indicating the nurse's type; all nurses in the dataset with this value will work the same number of day and night shifts, although the labelling system may change between datasets.. The second row provides the number of day and night shifts the nurse is contracted for, respectively, along with a third value indicating whether the nurse will work both days and nights in the same week. As mentioned earlier, all 411 possible shift patterns may be enumerated and the third row refers to the ordered list of shift patterns giving the list numbers of the first and last of these patterns this nurse may work if on days and the first and last numbers they may work if on nights, in this order. Nurses working both days and nights in the same week will have just two numbers in this row, indicating the first and last patterns in the list they may work. The final list of numbers gives the preference cost values for this nurse for each of these potential shift patterns, in the order given in the third row. Here, a value of 100 signifies an infeasible pattern which the nurse will not be allowed to work. Thus, certain nurses are forced to work either days or nights according to these values.



The following constitutes the full version of the raw data supplied for dataset 41.

```
  2  2  2  2  2  2  2  1  1  1  1  1  1  1
  4  4  4  4  4  4  4  2  2  2  2  2  2  2
  7  8  8  8  8  8  7  3  3  3  3  3  3  3
 Nurse             1           1           1
                   5           4           0
    1   21   22   56
  0  2  0  1  2  0  1  1  2  0  2  2  2  2  1  0  1  1  1  2
  0100100100100100100100100100100100100100100100100100100100
10010010010010010010010010010010010010010010010010010010010
 Nurse             2           1           1
                   5           4           0
    1   21   22   56
  0  2  0  1  2  0  1  1  2  0  2  2  2  2  1  0  1  1  1  2
  0100100100100100100100100100100100100100100100100100100100
10010010010010010010010010010010010010010010010010010010010
 Nurse             3           1           2
                   4           3           0
   57   91   92  126
 13 15 15 15 14 15 16 14 16 15  3  3  3  3  5  4  2  3  4  3
 13 15 14 14 15 14 15 15 15 15  0  2  2  2  0100100100100100
10010010010010010010010010010010010010010010010010010010010
10010010010010010010010010010010010
 Nurse             4           1           1
                   5           4           0
    1   21   22   56
 25 15 14 14 16  2 26 27 16 14 26 27 15 16 26 28 29 17 17 30
 28 25 27 15 28 38 27 40 26 40 27 39 27 40 27 41 28 38 52 40
 39 28 18 30 17 30 17 30 42 30 29 28 42 29 29 40
 Nurse             5           1           1
                   5           4           0
    1   21   22   56
10010010010010010010010010010010010010010010010010010010010
100  0 14 14 15 13 14 15 25 27 26 14 14 15 26 28 27 25 27 27
 38 12 14 14 25 26 25 26 26 26 37 24 26 25 37 36
 Nurse             6           1           2
                   4           3           0
   57   91   92  126
  0  2  2 13  1  2 14  1 14 13  2  2 13  2 15 14  2 13 14 13
  0  2 13  1 14 13  2 14 14 14  0 13 13 13 12 12 25 25 12 14
 14  1 26 13 12 14 14  2 26 14 13 14  2  2 12 27 27 14 39 27
 26 27 15 15 26 27 15 14 25 12
```

```
Nurse              7            1            4
                   3            2            0
 127  161  162  182
24 13 13 24 14 14 25  2 13 12 26 27 38 15 27 25 14 26 25 12
13 14 25  2 14 14  2 14 14  1 14 26 26 13 20 12 13 24 14 26
24 14 26 25 24 15 27 26 25 24  3 15 15 14 13 12
Nurse              8            1            3
                   1            3            1
 332  411
12 14 14 20 26 27 33 26 33 32 24 26 32 26 33 32 36 44 44 42
24 26 32 26 33 32 36 44 44 42 41 41 47 41 47 47 36 44 44 42
41 41 47 53 59 59 53 59 59 54 12 41 47 41 47 59 41 47 59 59
12 26 47 26 47 59 24 47 59 59 18 32 32 32 33 44 30 32 44 42

Nurse              9            1            2
                   4            3            0
  57   91   92  126
 0  3  3  3  2  3  4  2  4  3  3  3  3  3  5  4  2  3  4  3
 0  3  2  2  3  2  3  3  3  3  0  2  2  2  0 37 26 26 25 27
27 26 15 14 13 39 39 39 27 27 26 27 27 27 13 29 29 27 17 16
15 17 16 16  3 29 28 27 14 13
Nurse             10            2            1
                   5            4            0
   1   21   22   56
100100100100100100100100100100100100100100 14 15 27 27 28
26 24 26 38 38 25 38 38 37 38 49 26 38 38 38 39 50 37 38 50
49 12 26 26 25 26 37 26 26 38 37 24 26 37 37 36
Nurse             11            2            7
                   0            0            0
 218  218  219  219
   0    0
Nurse             12            2            7
                   0            0            0
 218  218  219  219
   0    0
Nurse             13            2            7
                   0            0            0
 218  218  219  219
   0    0
Nurse             14            2            1
                   5            4            0
   1   21   22   56
100100100100100100100100100100100100100100100100100100
100 12 14 14 27 13 14 27 13 27 26 14 14 27 14 28 27 13 27 27
 26  0  2 14  1 14 13  2 14 14 13  0 14 13 13 12
Nurse             15            2            2
                   4            3            0
  57   91   92  126
 0 14  2  1 13  2  2 13 14  1 14  2  2 14 15  2 13 13  2 13
12  2  1 13 14  1 14 14  2 14 12 13  1 13 12 22 11 23 22 12
24 23 12 11 22 12 24 24 12 12 23 12 12 24 10 13 25 24 13 13
24 13 13 25 12 13 13 24 11 10
```

```
 Nurse            16           3           1
                   5           4           0
    1   21   22   56
 13 15 14 14 16 14 14 15 16 26 14 15 15 28 26  0  2  2 14 15
 13 25 15 15 16 26 27 28 14 16 15 27 27 28 15 17 16 26 28 28
 27 13 15 15  2  3  2 15 15 15 14 13 15 14 14 25
 Nurse            17           3           1
                   5           4           0
    1   21   22   56
100100100100100100100100100100100100100100100100100100100100
100  0 14 14 14 13 14 14 25 26 25 14 14 14 26 27 26 25 26 26
 37 12 14 14 25 26 25 26 26 26 37 24 26 25 37 36
 Nurse            18           3           1
                   5           4           0
    1   21   22   56
  0  3  2  2  4  2  2  3  4  2  2  3  3  4  2  0  2  2  2  3
  0  0  3  3  4  2  3  4  2  4  3  3  3  4  3  5  4  2  4  4
  3  0  3  3  2  3  2  3  3  3  2  0  3  2  2  0
 Nurse            19           3           4
                   3           2           0
  127  161  162  182
 12  1 13 12  2 14 13  2  1 12  2 15 14  3  3 13  2  2 13  0
  2 14 13  2  2 14  2  2 14  2  2  2 14  1  0  0 13 12 14 14
 24 14 14 25 24  3  3 14 13 12  3  3 15 14 13  0
 Nurse            20           3           2
                   4           3           0
   57   91   92  126
  8  2  2  1  1  2  2  1  2  1  2  2  1  2  3  2  1  1  2  1
  0  2  1  1  2  1  2  2  2  2  0  1  1  1  0  0  1  1  0  2
  2  1  2  1  0  2  2  2  2  2  1  2  2  2  0  3  3  2  3  3
  2  3  3  3  2  3  3  2  1  0
 Nurse            21           3           2
                   4           3           0
   57   91   92  126
  0  2  2  1  1  2  2  1  2  1  2  2  2  2  3  2  2  2  2  2
  0  2  1  1  2  1  2  2  2  2  0  1  1  1  0 10 11 11 10 12
 12 11 12 11 10 12 12 12 12 12 11 12 12 12 10 13 13 12 13 13
 12 13 13 13 12 13 13 12 11 10
 Nurse            22           3           5
                   1           1           0
  211  217  204  210
100  0  0  0  0  0100100100100100100100100
```

Appendix B

Calculation of nurses' preference costs

Chapters 4 and 5 detail experiments performed with the 52 datasets listed in Appendix A. For each nurse, all shift patterns which are feasible with regard to the number of day and night shifts they are contracted to work are assigned a single cost in the range [0,100] which incorporates the costs relating to constraints P1 – P7. The exact manner in which these costs are derived is explained here. The idea is that anything infeasible adds 90 to the cost, anything feasible but highly undesirable adds 18 and any violation of little importance adds 3. Other costs range between 4 and 12. These costs were derived as the result of several consultations at the hospital by Dowsland and Thompson and trials over several weeks/wards. The following costs were created.

Basic shift pattern cost

Each of the 411 shift patterns was given a cost, in consultation with the hospital, according to the mix of consecutive working days or off duties. These were allocated subjectively and are mostly in the range 0 to 3, but some undesirable patterns for (1&2) and (1&3) were given a cost of 18. All costs can be customised for individual nurses but this is not the case for any of the 52 test data sets.

Preferences and requests

Night/Day contracts/preferences

Some nurses are contracted to work only nights or only days. The 'wrong' types of shift pattern were therefore given an additional penalty of 90. Other nurses are contracted to work only nights or only days 'if possible'. This condition can be violated in order to have sufficient cover. All shifts of the 'wrong' type were given an additional penalty cost of 18.

Requests for current week

Requests for off-duties are classified by the person in charge of rostering, using 5 different categories of increasing importance, with category 5 being essential, 4 very important etc. These were given costs of 3, 8, 12, 18 and 90 for categories 1 – 5 respectively. This cost was added for every shift included in the request and covered by the shift pattern. For example, a category 3 request for Wednesday and Thursday off would add a penalty cost of 24 for all patterns covering both Wednesday and Thursday and a cost of 12 for patterns covering just Wednesday or Thursday. Note that this option was preferred by the hospital to that of penalising each unmet request just once regardless of the number of shifts involved.

Requests for the following week

When scheduling week i the requests for week i+1 are usually known. If the number_of_requests + number_of_shifts worked = 7 then for the requests to be met next week's shift pattern will be fixed. Any pattern that would cause more than 7 consecutive days work when combined with the fixed pattern for week i+1 is penalised with an extra cost equal to the penalty associated with the first day off requested in week i+1.

Penalties based on historical information

Infeasible shifts

When scheduling week i, if the Saturday night was worked in week i–1 then the cost of all patterns involving Sunday day have an additional penalty of 90.

Isolated shifts and isolated weekends

When scheduling week i, if the Saturday of week i–1 was not worked then all patterns which involve working Sunday but not Monday have an increased cost of 1. Similarly for patterns which do not include a Sunday shift when week i–1 included a Saturday shift, but not a Friday.

Rotate nights

If the nurse is not contracted to work nights, or is not on a preferred nights contract, then if nights were worked in week i–1 a penalty of 5 is added to all night shifts. A

further penalty of 5 is added if nights also worked in week i–2 before. Similarly, if the nurse is not contracted to work days, or is not on a preferred day contract then if days have been worked for the last 3 weeks, a penalty of 4 is added to all day shifts.

Spread weekend working

If the nurse worked both Saturday and Sunday for the last k weeks, penalise any shifts including both Saturday and Sunday with an additional penalty cost of k.

More than seven consecutive shifts

Let the sum of the consecutive shifts at end of last week plus the consecutive shifts at start of the current pattern be represented by k. If $k \geq 8$ add k to the penalty cost. Note that since no more than 5 shifts can be worked in a given week, no more than 10 consecutive days are possible.

Share out 'bad' shifts

If the nurse did not work a zero-cost shift pattern last week add the cost of last week's pattern to any pattern with a non-zero cost.

After the above calculations, all penalty costs greater than or equal to 90 are then set equal to 100 and thus are considered infeasible.

Appendix C

Nurse scheduling pre-processing phase

The number of bank nurses and dummy shifts required were determined by solving a knapsack problem to determine the maximum number of days D_{max} that can be covered given that the night requirement must be satisfied. If D_{max} fell short of the required number of day shifts, by say k shifts, then k bank nurses, each working 1 shift, were added. If D_{max} exceeded the required number by k, however, then there was a requirement the surplus be allocated to days, spread evenly, and that weekdays should be over-covered before weekends. In order to ensure this, the requirement on weekdays was increased by an appropriate amount and a dummy nurse was added working sufficient shifts to cover the extra requirement that could not be met as follows.

First if $k \geq 7$ then we can cover every day by an extra $\left\lfloor \frac{k}{7} \right\rfloor$ shifts. Thus we increase the requirement on every day by this amount.

For all values of k if $k\text{mod}7 \neq 0$ we will have $k\text{mod}7$ surplus shifts to allocate and they must allocated to $k\text{mod}7$ different days. These must be weekdays if $k\text{mod}7 \leq 5$, and include Saturday or Sunday if $k\text{mod}7 = 6$. In the former case we increase the day requirement by an additional shift for weekdays and add a dummy nurse who is able to work $5 - k\text{mod}7$ days, excluding weekends. In the latter we increase all day requirements by 1 and add a dummy nurse who is able to work either Saturday or Sunday.

Appendix D

Glossary of notation

Throughout the thesis various types of notation have been introduced. In some cases, notation has been reused, referring to different things in different parts of the thesis as this was the best available notation and choosing different symbolism would have made the topic less clear. We present here a glossary of the notation used within the thesis, which may be used both for clarification and reference purposes.

D.1 Notation for Chapter 4 – Nurse scheduling with GRASP

a_{jk}	A variable whose value is 1 if shift pattern j covers shift k, and 0 otherwise
Best(c)	The preference cost of the best solution found in cycles 1 to c
c	An arbitrary GRASP cycle
C_{gk}	The number of nurses of grade g or higher required on day k
Cons(P)	The set of constraints related to finding a solution to problem instance P which is compatible at all grades
d_t	The number of day shifts worked by a nurse of type t
d_{gk}	The current amount by which shift k at grade g is understaffed, known as the undercover of shift k at grade g
D_g	The total day shift requirement at grade g
e_t	The number of night shifts worked by a nurse of type t
E_g	The total night shift requirement at grade g
Freq(c)	The frequency with which a day/night partition Y appears in cycles 1 to c
g	An arbitrary grade
g_i	The grade of nurse i
h_{ig}	A variable whose value is 1 if nurse i is of grade g or higher, and 0 otherwise
i	An arbitrary nurse
j	An arbitrary shift pattern

k	An arbitrary shift
$KS_g(P)$	The problem of finding a solution to the knapsack equations (4.14_g) – (4.16_g) of problem instance P at grade g
n	The number of highest-scoring candidates from which each constructive selection is made
n_{tg}	The number of nurse of type t of grade g or higher
$N_C(s)$	The set of solutions reachable from solution s by making one move in the *Change* neighbourhood
$N_E(s)$	The set of solutions reachable from solution s by making one move in the *Extended* neighbourhood
$N_S(s)$	The set of solutions reachable from solution s by making one move in the *Swap* neighbourhood
p_{ij}	The preference cost associated with nurse i working shift pattern j
P	An arbitrary problem instance
$P(R')$	A partial solution to problem instance P in which the number of day and night shifts worked by nurses in the set R' is known, giving a problem of reduced size
r	The total number of nurses
R	The set of nurses
R^+	The set of nurses who have already been assigned a shift pattern
R^{both}	The set of nurses who work both days and nights
R^{fixed}	The set of nurses who must work either days or nights due to feasibility restrictions
S_i	The set of shift patterns which are feasible for nurse i
t	An arbitrary nurse type
T	The total number of nurse types
ThreshK	The dynamic threshold for the number of times a particular day/night partition may be utilised
ThreshP	The dynamic preference cost threshold
$V(c)$	The number of different day/night partitions Y which have been employed in cycles 1 to c
w_c	The weight associated with the cover cost part of the heuristics' construction scores

w_g	The weight associated with grade g
w_p	The weight associated with the preference cost part of the heuristics' construction scores
x_{ij}	A variable whose value is 1 if nurse i has been assigned to shift pattern j, and 0 otherwise
y_{tg}	The number of type t nurses of grade g or higher who have been assigned to nights
Y	A solution consisting of a compatible set of values Y_g at all grades g
Y_g	A solution to a problem $KS_g(P(R'))$

D.2 Notation for Chapter 5 – Nurse scheduling with ACO

a	An arbitrary ant
c^a	The cover cost of the solution built by ant a
C^a	The cost relating to the solution built by ant a, referring to the general case
$\Delta\tau^a_{tt'}$	The amount to be added to trail matrix element $\tau(t,t')$ relating to the success of the schedule created by ant a
gen	The total number of cycles
gen^+	The number of completed cycles
i	An arbitrary nurse
j	An arbitrary shift pattern
$\|j\|$	The number of shifts worked in shift pattern j
k	An arbitrary shift
m	The total number of nurses to be scheduled
$nants$	The total number of ants in the population
$nants^+$	The number of ants in the population which have completed a solution
p^a	The preference cost of the solution built by ant a
Q	A parameter signifying the importance of the elements chosen for inclusion in the initial cycles
ρ	The evaporation rate
R	The set of nurses
R^+	The set of nurses who have already been assigned a shift pattern

τ	The trail matrix
τ_0	The initial value assigned to each element of the trail matrix, τ
$\tau(t,t')$	The value of the trail matrix relating to the arbitrary items t and t'
$t(i)$	The trail score related to assigning nurse i next, for use with the nurse-nurse trail
$t(i,j)$	The trail score related to making assignment (i,j), for use with the nurse-pattern and nurse-shift trails
T^a	The schedule created by ant a
$v(i,j)$	The visibility score related to making assignment (i,j)
w_c	The weight associated with the cover part of the visibility score
w_p	The weight associated with the preference cost part of the visibility score
w_t	The weight associated with the trail part of the construction score
w_v	The weight associated with the visibility part of the construction score
W	The weight indicating the importance of feasibility in a completed solution when feeding back to the trail matrix

D.3 Notation for Chapter 6 – Medical student scheduling with GRASP

a_{fq}	A variable whose value is 1 if firm f covers speciality q, and 0 otherwise
A_{qi}	The total number of students who have been assigned to study speciality q in timeslot i
B_{pq}	The number of students of type p who may study speciality q in a given timeslot, in order to ensure a feasible completion of the solution is possible
c_{fg}	A variable whose value is 1 if firms f and g have a consultant in common, and 0 otherwise
C_f	The capacity of firm f
C_{ft}^+	The number of students assigned to firm f in timeslot t
δ	An arbitrary dataset
d_{qt}	The number of places remaining in speciality q in timeslot t
D_q	The set of students who have already been assigned to a firm covering speciality q

f	An arbitrary firm
G_{st}	The set of firms covering a speciality s has not previously studied in timeslots 1 to $t-1$
h_{fg}	A variable whose value is 1 if firms f and g are in the same hospital, and 0 otherwise
λ	The number of modifications made to a dataset with regards to the amount of slack in the problem
L	The restricted candidate list
μ	A solution approach
$m_\lambda(\delta)$	A set of eight datasets based on original dataset δ and with λ modifications
$M_C(\sigma)$	The set of solutions reachable from solution σ by making one move in the *Change* neighbourhood
$M_P(\sigma)$	The set of solutions reachable from solution σ by making one move in the *Permutation* neighbourhood
$M_S(\sigma)$	The set of solutions reachable from solution σ by making one move in the *Swap* neighbourhood
n	The number of items on the restricted candidate list, L
N	The number of timeslots (excluding timeslot 1) in the general problem case
N_{szt}	The number of times students s and z have been placed together in timeslots 1 to $t-1$
p	An arbitrary student type, indicating the particular specialities remaining to be covered
ρ	The density of a given clash matrix, that is, the probability that a randomly selected pair of firms share a consultant
r	The total number of students
σ	An arbitrary solution
σ_δ	The standard deviation of all solution costs obtained for dataset δ
s	A student belonging to set S
S	The set of students
S^+	The set of students who have been assigned a placement in the relevant timeslot

t	An arbitrary timeslot
q	An arbitrary speciality
q'	The speciality with the most remaining space in a given timeslot
v	The total number of firms
w_f	The weight associated with the feasibility part of the heuristics' construction scores
w_o	The weight associated with the optimality part of the heuristics' construction scores
$\overline{\chi}_\delta$	The mean of all solution costs obtained for dataset δ
$\chi_{\mu\delta}$	A final cost arising from applying method μ to dataset δ
$\hat{\chi}_{\mu\delta}$	A final, normalised, cost arising from applying method μ to dataset δ
x_{sft}	A variable whose value is 1 if student s is assigned to firm f in timeslot t, and 0 otherwise
$y(s,f,t)$	The score associated with assigning the triplet (s,f,t).

Appendix E

Tabu search

As tabu search (TS) is a metaheuristic which has often been applied to the nurse scheduling problem and, in particular, has been very successfully applied to the variant of the nurse scheduling problem presented in this thesis, we give a description of the method here.

In order to find good solutions to a problem, local search is often implemented. Starting from a fully constructed solution, the neighbouring solutions, those which may be obtained from the current solution by making a small change, are investigated; if any of these neighbours are of lower cost, the current solution may be replaced by the improved solution and the process repeated until a local optimum has been found. It is important that the search be able to escape from such minima in order to fully search the solution space. If the landscape of solutions is hilly, it is impossible to fully explore the landscape without making some worsening moves and a straightforward 'descent' approach will not be as successful.

Tabu search, proposed by Glover (1986), is one of the methods used to overcome this problem. During the local search, the solution will gradually change as each improvement is accepted. The tabu search method makes use of a short-term memory, selectively recording the previous versions of the solution in a 'tabu' or taboo list H. The search then takes the following form: All neighbouring solutions are evaluated and the best of these, which will not result in the solution reverting to a state on the list H, is accepted as the new solution. H may then be updated so that the most recent solution is added and the oldest solution currently on the list is removed and may now be revisited. Generally, the tabu list does not hold a list of complete solutions, but a list of solution attributes, so that a particular assignment may be disallowed, for example. However, an 'aspiration' move is generally allowed. This is one which accepts a solution of a higher quality than the best found so far, regardless of the tabu list, ensuring that good solutions are not overlooked.

Obviously, once a local optimum is reached, the next solution accepted by the local search must then be a worsening move and, depending on the landscape and size of H, it is likely that the next few moves will also be worsening in terms of cost. The theory is that, if H is large enough, by the time the search next starts accepting improving moves, the solution will have moved far enough through the landscape to descend to a different local optimum. By utilising this form of short-term memory, the search is able to diversify and by allowing a more rigorous exploration of the solution space it is now much more likely that good solutions will be found; with tabu search, high-quality solutions may still be reached even if the initial solution is in a problematic area of the solution space, whereas with a simple descent algorithm this would be impossible.

Appendix F

Genetic algorithms

Genetic Algorithms (GAs) are a well-known and widely-used class of metaheuristics. Due to their prevalence in the literature with regards to nurse scheduling applications, and particularly the nurse scheduling problem presented in this thesis, we give a description of this metaheuristic here.

In the same way as Ant Colony Optimisation attempts to find solutions by emulating ant behaviour, GAs also draw on concepts found in nature, imitating the genetic evolution of a population. All GAs must be initialised with a population of solutions, each encoded as a string of finite length. Two parent members of the population 'breed' by using information from both parents to form a new, child, solution; the particular method, or crossover operators, by which the new solution is formed from the two parents is variable and will usually significantly impact the quality of the results.

Each child may or may not be included in the next generation of the population depending on its fitness, this, in turn, dependant on some appropriate objective function. The idea is that by selecting fitter parents for breeding and only accepting children of a certain calibre into the next generation, the population will gradually evolve into one containing, on average, higher-quality solutions.

Another important part of GAs is the idea of mutation. A small, but random change is made to a solution in the population. This is important for injecting diversity into the population and preventing stagnation.

GAs have been widely used to solve a number of problems, but are often difficult to implement for highly-constrained problems; when a crossover operator acts on two parents representing feasible solutions to a highly-constrained problem, it is often difficult to ensure that the child solution will also be feasible. This leads to an

additional 'repair' operation being required in many situations. The possible options available are as follows:

- Include the constraints into the encoding of the solution.
- Penalise infeasible solutions.
- Repair infeasible solutions so that they become feasible.
- Use an indirect approach: the encoding on which the GA acts is a set of instructions for how to assemble the complete solution.

For highly constrained optimisation problems, it is unlikely that all constraints may be contained within the solutions' encoding and so one or more of the other methods must be employed in order to find good solutions using a GA approach. Using a penalty function to guide the selection process and thus penalise infeasible solutions is an easy method to implement and is often utilised in GA approaches. However, it does not guarantee feasibility and so repair functions, requiring much more problem-specific information, but more often able to guarantee feasibility are also a widely-used GA feature. Using both penalty and repair functions in combination with each other is also common (Aickelin 1999).

Appendix G

Paper accepted for publication

The following paper, produced from the research in this thesis, has been accepted for publication and is to appear in the Journal of Heuristics.

A GRASP-KNAPSACK HYBRID FOR A NURSE-SCHEDULING PROBLEM

ABSTRACT

This paper is concerned with the application of a GRASP approach to a nurse-scheduling problem in which the objective is to optimise a set of preferences subject to a set of binding constraints. The balance between feasibility and optimality is a key issue. This is addressed by using a knapsack model to ensure that the solutions produced by the construction heuristic are easy to repair. Several construction heuristics and neighbourhoods are compared empirically. The best combination is further enhanced by a diversification strategy and a dynamic evaluation criterion. Tests show that it outperforms previously published approaches and finds optimal solutions quickly and consistently.

KEY WORDS: Nurse-scheduling, rostering, GRASP, hybrid, knapsack.

1. INTRODUCTION

Construction heuristics, neighbourhood search techniques and evolutionary algorithms have all been used successfully in the solution of practical scheduling problems. However, there is often a conflict between feasibility and solution quality, and the difficulty in maintaining a suitable balance between these two objectives is well documented. For example Michalewicz and Fogel (2004) devote a whole chapter to the handling of constraints in evolutionary algorithms. Early examples from the fields of timetabling and scheduling include Wright (1991) and Abramson (1991) while observations made by Aickelin and Dowsland (2004), Zhu and Lim (2006) and Lim, Rodrigues and Zhang (2006) show that these issues are still relevant. They arise for a number of reasons. In some cases it may not be practical to limit the solution space to the set of feasible solutions, as simply finding a feasible solution may itself be a non-trivial problem. Even if this is not so, it may still be difficult to define operators, such as crossover in a genetic algorithm or neighbourhood moves in local search, that preserve feasibility. In other cases, restricting the search to feasible solutions may result in a solution space that is very sparsely connected and this may hamper the ability of the search to seek out high quality solutions. (See for example Thompson and Dowsland (1996) and Dowsland (1998)). There are a number of ways of overcoming this, for example the use of penalty functions, repair functions, wider neighbourhoods etc. However, none of these is able to provide a universal answer and when they do work their success is often the result of exploiting information about the problem structure, for example in setting appropriate penalty weights, or allowing the efficient exploration of large neighbourhoods.

One approach that has received considerable interest in the solution of scheduling problems is Greedy Randomised Adaptive Search Procedure (GRASP). A construction phase is followed by an improvement phase and these are repeated for several cycles. Examples include Drexl and Salewski (1997) who use a two phase greedy randomised algorithm to solve the school timetabling problem, Binato et al. (2001) who incorporate intensification strategies into their GRASP for Job Shop Scheduling, and Gupta and Smith (2006) who schedule a single machine to minimise tardiness using a variation of Variable Neighbourhood Search in the improvement phase of their GRASP.

This paper is concerned with a GRASP approach to a variant of the nurse-scheduling problem in which the objective is to find a schedule that optimises a set of preferences while meeting a series of binding constraints. The constraints are often tight i.e. the nurses available are just sufficient to meet the requirements without any slack. Thus simple heuristic rules often fail to produce feasible solutions and existing approaches capable of producing high quality solutions across a range of problem instances all use problem specific information to balance the focus between optimality and feasibility (Dowsland (1998), Dowsland and Thompson (2000), Aickelin and Dowsland (2000, 2004)). The objectives of our investigation are two-fold: firstly to produce a robust solution approach to the problem, which is typical of that arising in many UK hospitals; and secondly to examine the impact of the construction heuristic used within the GRASP algorithm on solution quality. In particular, we investigate the use of problem specific knowledge, in the form of a look-ahead procedure based on a knapsack model, to strengthen the construction heuristic. This allows us to use a

straightforward descent algorithm based on simple neighbourhood moves during the improvement phase.

In the next section we describe the problem and outline other solution approaches reported in the literature, highlighting the fact that the most successful of these have all included special procedures to deal with the quality/feasibility issue. This is followed by an introduction to GRASP in Section 3. Section 4 deals with the family of GRASP heuristics we have developed for the problem, while Section 5 describes an enhancement achieved by hybridising the construction phase with a knapsack algorithm. Section 6 is concerned with computational experiments. These are based on 52 data sets from a large UK hospital and are designed to compare the different variants with each other, and to compare the most successful with other approaches from the literature. Finally, conclusions and further work are discussed in Section 7.

2. THE NURSE-SCHEDULING PROBLEM

The problem of allocating nurses to duty rosters occurs in hospital wards all over the world. The essence of the problem remains the same in each case: every nurse in the ward must be assigned a suitable set of shifts for the given planning period. However, the precise details of the problem differ from hospital to hospital. Objectives range from minimising costs to maximising the satisfaction of the personnel involved (e.g. (Jaumard et al. (1998), Berrada et al. (1996)), while a wide range of working practices leads to an equally wide range of problem constraints. Some hospitals employ a cyclic schedule where for each planning period the same schedule is repeated, and each nurse cycles around a subset of the selected shift patterns, (e.g. Rosenbloom and Goertzen (1987)). Others use non-cyclic schedules in order to provide more flexibility in dealing with absence, fluctuations in patient numbers and staff requests (e.g. Cheang et al. (2003)). Other variations include the length of the planning period, the numbers and lengths of shifts on each day, whether or not different skill levels need to be taken into account, and constraints on working practices such as the number of consecutive work days, minimum rest period between shifts, rotation of night/weekend working etc. The papers by Dowsland and Thompson (2000), Isken (2004), Bellanti et al. (2004) and Burke et al. (2003a) illustrate many of these differences. A range of solution approaches have been used to tackle the different variants of the problem, including exact approaches, often based on mathematical programming (e.g. Isken (2004)), artificial intelligence (e.g. Meyer auf'm Hofe (2000), Petrovic et al. (2003)) and a variety of heuristics including genetic algorithms (e.g. Burke et al. (2001), Moz and Pato (2007)), ant colony optimisation (Gutjahr and Rauner (2007)), simulated annealing (e.g. Brusco and Jacobs (1995)), tabu search (e.g. Dowsland (1998), Bellanti et al. (2004)) and other forms of local search e.g. SAWing (Parr and Thompson (2007)). Of these, tabu search seems to be the most popular. Most of the approaches reported in the literature deal with a fixed problem definition, but others such as the algorithm PLANE described in Burke et al. (1999) allow many of these details to be user defined. Recent surveys can be found in Burke et al. (2004) and Ernst et al. (2004).

The variant of the problem considered here is that introduced in Dowsland (1998) and also tackled in Dowsland and Thompson (2000), Aickelin and Dowsland (2000, 2004), Aickelin and Li (2007) and Burke et al. (2003b). The objective is to produce weekly work rosters for individual hospital wards. These must satisfy individual work

contracts and ensure there are sufficient nurses with the desired skill mix on duty at all times, whilst also meeting the nurses' preferences and requests as far as possible. The day is divided into three shifts; an 'early' day shift, a 'late' day shift and a night shift. The majority of nurses work either days or nights in a given week, although a few may be contracted to work a combination of the two. Night shifts are longer than day shifts. Due to the different shift lengths, a nurse working day shifts will typically, but not always, work more shifts than a nurse working nights. For example, a full time nurse works either 5 day shifts or 4 night shifts. Part time nurses work other combinations e.g. 4 days or 3 nights and 3 days or 2 nights, while some part-timers are contracted to work equal numbers of days or nights. In total there are ten possible day/night combinations. At the start of the planning period the combination pertaining to each nurse is known. We will refer to this combination as the *type* of the nurse.

Each nurse is graded according to their level of expertise and experience, resulting in 3 grade bands, with band 1 consisting of senior nurses who are capable of being in charge of the ward, and band 3 those nurses who are trained only to the minimum level. For each shift there is a minimum requirement of nurses of each grade, and as nurses of higher grades can cover all the duties of those at lower grades these are expressed cumulatively. For example, requirements of 1, 2 and 5 nurses at grades 1, 2 and 3 respectively means that at least one grade 1 nurse, two grade 1 or 2 nurses, and five nurses in total are required. Where there are more nurses than required the additional cover should be spread uniformly over the planning period and if there are insufficient nurses additional 'bank' nurses must be used. The requirements that each nurse works the correct number of shifts, that the minimal covering requirements are met, and that shifts at the start of the week are compatible with those worked at the end of the previous week are binding, and must be met if the solution is to be feasible. In addition, there is a requirement to produce rosters that are user friendly. This involves attempting to meet requests not to work certain shifts/days, rotation of nights and weekend working, and avoiding sequences of shifts that are unpopular e.g. working alternate days.

Thompson and Dowsland (2000) show that the problem can be pre-processed so that any bank nurses required can be added beforehand, and 'dummy' nurses can be added to deal with the problem of spreading any excess cover. This results in a tight problem in which there is no slack in the overall cover requirements. They also show that the allocation of day nurses to 'earlies' and 'lates' can be dealt with in a post-processing phase. Thus the 'early' and 'late' requirements can be amalgamated into a single 'day' requirement. The remaining problem can be modelled as an integer program as follows.

For nurses working d day shifts or e night shifts, the set of feasible shift patterns for that type is defined by a set of binary vectors of length 14, with the first 7 values representing the day shifts and the remaining 7 values the night shifts. A '1' implies that the nurse is working and a '0' implies a day or night off. The number of possible shift patterns for each nurse is $\binom{7}{d}$ day patterns and $\binom{7}{e}$ night patterns. Patterns for those nurse-types working a combination of days and nights can be defined similarly. The feasible patterns for individual nurses are then formed by removing any patterns that would violate the constraints relating to working practices, annual leave etc. The quality of each pattern is reflected by the allocation of a penalty cost, referred to in the

following as the *preference cost*, in the range [0, 100] with '0' implying the shift pattern is ideal and '100' signifying it is infeasible. These values were obtained as the result of an iterative consultation process with a major UK hospital. The IP representation is then:

$$\text{Minimise} \sum_{i=1}^{r} \sum_{j \in S_i} p_{ij} x_{ij} , \tag{1}$$

$$st \sum_{j \in S_i} x_{ij} = 1 \quad \forall i , \tag{2}$$

$$\sum_{i \in R_g} \sum_{j \in S_i} a_{jk} x_{ij} \geq C_{gk} , \quad \forall g, k \tag{3}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if nurse } i \text{ works shift pattern } j \\ 0 & \text{otherwise} \end{cases}$$

$$a_{jk} = \begin{cases} 1 & \text{if shift pattern } j \text{ covers shift } k \\ 0 & \text{otherwise} \end{cases}$$

p_{ij} is the preference cost of nurse i working shift pattern j, S_i is the set of shift patterns that nurse i can work, C_{gk} is the number of nurses of grade g or better required on shift k, R_g is the set of nurses at grade g or better and r is the total number of nurses. The objective function (1) is the total preference cost, constraints (2) ensure that each nurse is assigned to a valid shift pattern and constraints (3) ensure that the cover requirements are satisfied. Typical problem dimensions are between 21 and 28 nurses and a total of 411 shift patterns. For some instances solving this IP to optimality can take several hours using the professional version of the Xpress-MP integer programming software (Fuller 1998).

Several heuristic methods have already been applied to this problem. The tabu search approach of Dowsland (1998) uses a series of complex neighbourhoods, evaluation functions and candidate lists to increase the scope of the search as it moves through the space of feasible solutions. Nevertheless a strategic oscillation strategy in which the search is periodically allowed to visit solutions outside of the feasible region is necessary in order to guarantee quality solutions. Aickelin and Dowsland (2000, 2004) develop two genetic algorithm (GA) approaches. In the first of these, the individuals of the population represent complete solutions to the problem. Fitness is defined as a weighted sum of the preference cost and the total number of undercovered shifts at each grade. In order to deal with constraints (3) (the covering constraints), three problem-specific modifications to the classical GA implementation are required. These are: a complex co-evolutionary strategy using sub-populations that allow constraints at different grades to be relaxed and then recombined intelligently; a hill-climbing repair operator; and an additional penalty/reward scheme that rewards offspring that are likely to be repairable and penalises those that are not. The second, more successful, approach uses a GA to order the nurses and then applies a heuristic decoder to produce a solution from this ordering. Several different heuristics using a different balance between feasibility and quality were tested and it was shown that a bias towards seeking feasibility while including some measure of

quality produced the best solutions. The final implementation also used a modified crossover operator and information from a lower bound to improve the decoder.

The success of the above approaches was achieved at the expense of a series of modifications and enhancements based on aspects of the problem structure. Two more generic approaches are those of Burke et al. (2003b) and Aickelin and Li (2007). The former applies a tabu search hyperheuristic that makes use of several of the simpler neighbourhoods used in Dowsland (1998). For each move the choice as to which neighbourhood to use is made via a scoring system based on the success of that type of move to date. However, their objective was to produce a fast, generic approach to the problem requiring little problem specific information or fine-tuning. While they achieved this objective the results are not comparable with those of Dowsland in terms of solution quality. Aickelin and Li schedule the nurses sequentially using a Bayesian network to determine an appropriate scheduling rule for each nurse. The results show that this approach consistently produces feasible solutions but once again solution quality does not match that of Dowsland's tabu search implementation.

The above suggests that balancing feasibility and optimality is indeed a difficult task for both population based and sequential search approaches to the problem. In the two more successful approaches i.e. Dowsland's Tabu Search method (1998) and Aickelin and Dowsland's indirect GA (2004), this was achieved by the use of problem specific information to drive a number of intelligent add-ons. In the following sections we investigate the possibility of using this type of information in the construction phase of a GRASP approach to the problem.

3. GRASP

GRASP was first introduced by Feo et al. (1994). It can be viewed as a multi-start local search approach in which the starting solutions are generated by a probabilistic greedy construction approach. In its original form, GRASP consists of *nrep* independent repetitions of two phases: *Construct* and *Improve*. *Construct* is the greedy construction phase that builds a solution one element at a time. At each stage in the construction each available element, i, is given a score $f(i)$ based on a local measure of the benefit of adding that element next. The element to be added is then chosen probabilistically according to the scores. There are obviously many strategies available for achieving this but the most common approach is to determine a candidate list of the best n elements for a given n and then to select randomly from this list. This is usually achieved using roulette wheel selection where each element is selected with a probability proportional to its relative score. *Improve* is a local search based improvement heuristic, in which the incumbent solution is repeatedly replaced by a neighbour of higher quality until no such neighbour exists i.e. a local optimum has been reached.

As with any metaheuristic search technique, in order to implement a GRASP approach to a given problem, a number of problem specific decisions must be made. These include the usual local search decisions i.e. the definition of the solution space, neighbourhood structure and evaluation function, together with the details of the construction heuristic including the definition of the score function $f(i)$.

Since the first GRASP implementations were published, researchers have suggested a number of ways of improving performance. Some of these focus on improving the solutions obtained from the construction phase. For example Laguna and Martí (2001) describe a GRASP implementation for the graph colouring problem. The construction phase builds up the colour classes one at a time, completing one colour class before moving on to the next. Rather than just allowing a single attempt at each colour class several attempts are made and the best, according to a suitable criterion, is selected. Others, for example Binato et al. (2001), Fleurent and Glover (1999), try to improve the partial solutions by periodically applying local search during the construction phase.

An alternative is to focus on improving the local search phase. Typical examples involve replacing the simple descent algorithm with a more powerful local search technique such as simulated annealing or tabu search (e.g. Laguna and González-Velarde (1991)). However, Laguna and Martí (2001) found that these additions did not improve the solutions significantly when given a limited amount of time for the improvement phase. Others have attempted to exploit information from earlier cycles to seek out better solutions. A popular approach is to use some sort of feedback mechanism to influence the greedy selection process in the construction phase. This is achieved by increasing the probability of selecting elements that correspond to features of good solutions and/or decreasing those that correspond to bad solutions, in much the same way as the trail mechanism influences the construction phase of an ant algorithm (Fleurent and Glover (1999)). Others use previously gleaned information in other ways. For example, Aiex et al. (2003) use the best existing solutions as the basis for a path relinking phase which is added after the local search phase in each iteration. For a summary of these, and other developments in GRASP and its applications, see Resende (2001).

Our main focus is on improving the construction phase, and instead of selecting the next element purely on a myopic basis we employ a look-ahead strategy based on a knapsack model to produce solutions that are more likely to be repaired by the local search phase. The variables in the knapsack model also form the basis of a feedback mechanism aimed at diversification. We also make minor modifications to the improvement phase, by relaxing the requirement that only improving moves are accepted, and using feedback information to place a threshold on the preference cost.

4. GRASP ALGORITHMS FOR THE NURSE-SCHEDULING PROBLEM

Due to the complexity of the problem and the tightness of the constraints, generating feasible schedules is itself a demanding task and, as such, our solution space must include both feasible and infeasible schedules. Thus our construction phase needs to produce a complete, but not necessarily feasible, schedule that will be used as the starting point for the local search improvement phase.

4.1. Construction

At each stage of the construction phase a nurse and an appropriate shift pattern for that nurse must be selected. Both selections could be made according to the score function or the nurse could be selected *a priori* according to a predefined ordering.

(Note that as each shift pattern may be used once, more than once, or not at all, pre-ordering the shift patterns and then selecting a nurse is not a sensible option.) Given a partially constructed schedule, the allocation of unscheduled nurse i of grade g_i to shift pattern j will add p_{ij} to the total preference cost and will improve the cover for those shifts included in pattern j that do not currently meet the staffing requirements for grades g_i and below. We therefore need a 2-part score function to capture both these aspects. Experiments with a number of different score definitions were carried out and these will be described in detail after the following outline of the construction process.

For any given score function, $f(i,j)$, the construction phase proceeds as follows:

Let R be the set of nurses and R^+ be the set of nurses already allocated.

Step 1: Set $R^+ = \varnothing$.
Step 2: Calculate the score $f(i,j)$ associated with allocating nurse i to shift pattern j, for all feasible pairs (i,j). (Note that the set of feasible allocations (i,j) may include all nurses not already allocated, or may relate to a single nurse i given by a predefined ordering.)
Step 3: Let L be the candidate list of the n highest scoring options.
Step 4: Select $(i,j) \in L$ using roulette wheel selection, i.e. each (i,j) is selected with a relative probability proportional to $f(i,j)$.
Step 5: Update the schedule with this allocation and set $R^+ = R^+ \cup \{i\}$
Step 6: If $R^+ \neq R$ go to Step 2.

Aickelin and Dowsland (2004) suggest three different score functions to guide the greedy construction heuristics used as decoders for their indirect GA implementation. The part of the score relating to preference cost is given by:

$$prefscore_{ij} = 100 - p_{ij}, \text{ where } p_{ij} \text{ is as defined in Section 2.}$$

Two different functions are used for the part of the score relating to cover. Our construction methods are based on these scores and we therefore present them in detail.

Both cover related scores are functions of the current total undercover of shift k at grade g, defined as:

$$d_{gk}(R^+) = \max\left\{0, C_{gk} - \sum_{q \in R^+ \cap R_g} \sum_{j \in S_q} a_{jk} x_{qj}\right\}$$

The first cover score was designed for use without a preference cost element in a situation where the nurse i to be allocated next is decided *a priori*. It is defined as follows.

Let

$$g' = \begin{cases} \min\left\{ g \geq g_i : \sum_{k=1}^{14} d_{gk}(R^+) > 0 \right\} & \text{if such a } g \text{ exists} \\ 3 & \text{otherwise} \end{cases}$$

and let $k' = \text{argmax}\{d_{g'k}\}$, then

$$covscore1_{ij} = \sum_{k \in K} a_{jk} d_{g'k}(R^+),$$

where $K = \begin{cases} \{1,...,7\} & \text{if } k' \leq 7 \text{ or the nurse must be on days} \\ \{8,...,14\} & \text{if } k' \geq 8 \text{ or the nurse must be on nights.} \end{cases}$

For a given nurse i, this score is a measure of the improvement in cover at grade g_i (or at the first uncovered grade below g_i if all cover at g_i is already satisfied), giving a greater score to those shifts with the most undercover. If the greatest shortfall in cover on a single shift is on nights then the score is based on night shifts only and vice versa. As the scores for different nurses are measured over different grades this method of scoring is not suited to situations in which the nurse has not already been determined. The priority given to nights or days also make it unsuited to combination with the preference score as it is common for there to be a large difference between the preference costs for days and nights for some nurses, and such differences will not be considered.

The second cover score is intended to overcome these problems. It is more flexible in that it includes a term for each grade where cover would be improved. These terms are weighted so that cover at higher grades, for which there are fewer nurses available, are given priority. This score is simply the weighted sum of the undercover and is given by:

$$covscore2_{ij} = \sum_{g=g_i}^{3} w_g \left(\sum_{k=1}^{14} a_{jk} d_{gk}(R^+) \right),$$

where w_g is the weight associated with grade g for $g = 1,...,3$.

We use these individual scores to form the basis of a family of construction heuristics by changing the way in which the scores are combined, using both dynamic and static orderings of the nurses, and by incorporating a simple look-ahead rule. Each of these will provide a different balance between the opposing goals of optimising preference costs and meeting the covering constraints, and can be expected to provide solutions with different attributes as starting points for the local search phase.

Our simplest construction heuristics define the score function *f(i,j)* to be a weighted sum of the preference score and a cover score and combine this with a fixed ordering of the nurses. We implemented two variants using this strategy, denoted *cover* and *combined* and given by:

cover: $f_1(i,j) = covscore1_{ij}$

combined: $f_2(i,j) = w_p.prefscore_{ij} + w_c.covscore2_{ij}$,

where w_p and w_c are two weight parameters.

In *cover* all the emphasis is on attempting to satisfy the cover constraints, while *combined* is a more balanced strategy. However, given that the ordering of the nurses is fixed we would not expect either to be as successful in producing high quality solutions for the start of the improvement phase as a more flexible approach in which both the nurse and the shift pattern are selected dynamically by the score functions. Thus for these options, if the full GRASP implementation is to be successful, the local search phase will be required to play a significant role. In particular, for *cover,* all the optimisation of the preference cost will be left to the local search phase.

We also consider two construction heuristics based on a dynamic ordering. The first uses the *combined* score function to select both the nurse and shift pattern. This option is referred to as *holistic* as it encompasses the whole space of options at each stage. However the number of available options with this approach will be large, especially in the early stages of the construction. This may result in longer solution times and a lack of focus during this phase. We therefore introduce our fourth variant, *last chance,* which strengthens the construction by incorporating an element of look-ahead, while reducing the number of shift patterns considered for each nurse to one. This is achieved by comparing the highest scoring option for each nurse with the second highest and defining a single score function for each nurse based on the difference. The selected nurse is then allocated to the shift pattern with the highest score. The score function for this option is calculated as follows:

$$\text{Let } j^*(i) = \arg\max_{j \in S_i} \{f_2(i,j)\} \text{ . Then}$$

$$\textit{last chance: } f_3(i) = \max_{j \in S_i} \{f_2(i,j)\} - \max_{\substack{j \in S_j \\ j \neq j^*(i)}} \{f_2(i,j)\}.$$

This variant has fewer options at each stage as only the nurse has to be chosen. However it does not suffer from the limitations of the fixed ordering in the *cover* and *combined* variants in that the best shift pattern associated with each nurse also changes adaptively, depending on the allocations made in previous stages. In addition, the inclusion of some degree of look-ahead into the score should help to overcome the problem inherent in all greedy approaches, that of potentially being left with a set of expensive options in the last few stages.

4.2 The improvement phase.

None of the above construction heuristics is guaranteed to produce a feasible solution. Therefore, the solution space for the improvement phase is the set of solutions in which each nurse is allocated to a feasible shift pattern, regardless of whether or not the covering constraints are satisfied. Our basic search strategy is that of random descent i.e. at each iteration we sample the space of neighbours of the current solution without replacement and accept an improving solution as soon as it is encountered. Our evaluation function reflects the importance of meeting the covering constraints over that of optimising the preference cost, and we define a solution as being better than its neighbour if the cover is improved or if the cover is unchanged and the penalty is improved. More precisely our evaluation function is defined as:

$$\sum_{i \in R} \sum_{j \in S_i} p_{ij} x_{ij} + W \cdot \sum_{k=1}^{14} \sum_{g=1}^{3} d_{gk}(R),$$

where $W >> p_{ij}$, $\forall i, j$.

We also consider two minor modifications to the straightforward descent strategy. Firstly, a typical data set may have many x_{ij} variables with identical p_{ij} values. This suggests that there may be large plateau-like areas in the solution landscape and for this reason we also experiment with accepting equal cost solutions. Secondly, during the search for a feasible solution, the above objective function may allow the preference cost to increase significantly from that produced at the end of the construction phase. We therefore introduce a threshold on the preference cost using feedback information from previous cycles.

As our objective is to exploit the construction part of the GRASP algorithm, rather than to develop a complex local search algorithm in which the construction phase plays only a minor role, our neighbourhood structure needs to be simple, but flexible enough to allow moves in different areas of the search space. For this reason we consider combinations of three natural neighbourhoods: a 1-opt move neighbourhood, a 2-opt move neighbourhood, and a swap neighbourhood as defined below. We then combine the neighbourhoods by selecting each with a given probability that may change according to whether or not the current solution is feasible.

1. Change neighbourhood
$N_C(s)$ is the set of solutions obtained from any solution, s, by changing the shift pattern of a single nurse. This neighbourhood can change both the cover and the preference cost and in theory any solution is reachable from any other by a series of moves using this neighbourhood. However, it is not sufficiently flexible in the context of our descent strategies, as if the cover constraints are tight, any solution, s, satisfying the covering constraints will not have any improving moves within $N_C(s)$.

2. Swap neighbourhood
$N_S(s)$ is the set of solutions obtained from s by swapping the shift patterns of two nurses where their patterns are compatible i.e. where both patterns are feasible for both nurses involved in the swap. As swapping the patterns of two nurses on days (nights) will not affect the cumulative cover at grade 3, this neighbourhood is likely to allow the search to progress further once a feasible solution is found even on tight problems. Note that even for tight problems there is likely to be slack at grades 1 and 2 but moves are only accepted if the shortfall in cover does not increase at any grade. However, as swaps are limited to nurses of a single type some areas of the solution space are likely to be unreachable.

3. Extended neighbourhood
$N_E(s)$ is the set of solutions obtained from s by changing the shift patterns of two nurses. This gives more flexibility than the above neighbourhoods when s satisfies the covering constraints. However it is a larger neighbourhood and it may be computationally expensive to find improving moves close to local optima when they

may be sparsely distributed. We overcome this by restricting its use to situations where s is already feasible and sampling from a candidate list of neighbours that is a superset of all those that result in an improvement. This is achieved as follows. Given s, an element of $N_E(s)$ is defined by the quadruple (i_1,j_1,i_2,j_2) where i_1 and i_2 are the 2 nurses involved and j_1 and j_2 are the new patterns for these nurses. The four parameters are sampled in the above order with all the options for the remaining parameters being exhausted before parameters higher in the order are changed. i_1 is sampled uniformly from the full set of nurses, but the candidate sets for the remaining parameters can be considerably reduced using the following logic. The brackets in the following refer to the case where we accept moves of equal cost as well as improving moves.

Given that s satisfies the cover constraints an improving move must result in a reduction in the preference cost. For any two nurses involved in such a move at least one must be moved to a pattern with lower (or equal) preference cost. As we are sampling from the set of nurses uniformly we can assume without loss of generality that this is true for i_1. Thus, given i_1 we sample j_1 from the set of patterns with penalty cost less than (or equal to) the cost of i_1's current allocation. Further efficiency gains are possible if the patterns for each nurse are pre-sorted into p_{ij} order. If there is little or no slack in the covering constraints then moving nurse i_1 to pattern j_1 may result in some shifts being under-covered. The second half of the move must repair this. Thus nurse i_2 must be of a sufficiently high grade to cover the shortfall and must currently be off duty on all the under-covered shifts. Our candidate list for i_2 is therefore restricted to nurses who satisfy these conditions. Finally pattern j_2 must be chosen from those patterns for which the increase in preference cost for i_2 is less than (or equal to) the reduction obtained from moving nurse i_1 to pattern j_1.

5. HYBRIDISING THE CONSTRUCTION HEURISTIC WITH A KNAPSACK ROUTINE

The construction heuristics and local search strategies described above can be combined in different ways to form a family of traditional GRASP algorithms based on a relatively straightforward greedy construction phase and a descent based improvement phase. As outlined in Section 3, many of the more successful GRASP implementations include some additional ingredients aimed at improving either the construction or improvement phases. In this section we describe a hybridisation of the construction phase with a knapsack model. This allows us to incorporate a powerful look-ahead mechanism that will ensure that solutions produced from the construction are promising from the point of view of producing feasible solutions during the improvement phase. Our rationale for this is based on observations in Aickelin and Dowsland (2004) and Dowsland (1998). They suggest that the solution landscapes produced by our neighbourhood structures and evaluation function are likely to consist of subspaces separated by 'ridges' so that the improvement phase will be restricted to a single subspace, even if we were to allow some small uphill steps. As stated in Section 2, the nurses are partitioned into types according to the number of day and night shifts for which they are available and the ridges are caused by the changes in cover that result in moving nurses of different types from days to nights and vice versa. The subspaces correspond to the number of nurses of each type

allocated to days and nights, and for tight problems many of these subspaces will not contain any feasible solutions. This problem can be overcome by a construction heuristic that only produces solutions that lie in those subspaces that also contain some feasible solutions.

Dowsland and Thompson (2000) observe that the fact that nurses work a different number of night shifts and day shifts means that it is not possible to decide if a problem is feasible simply by counting the number of shifts available. They go on to show that this problem can be overcome using a knapsack model. We will use this model to ensure that the solutions produced by the construction phase lie in suitable subspaces. As we will show, this is equivalent to ensuring that the day/night allocations have sufficient numbers of nurses to meet the total covering requirements on both days and nights for all grades, g.

Essentially Dowsland and Thompson show that there are sufficient nurses to meet the cumulative cover constraints for grade g if the solution to the knapsack problem:

$$\text{Maximise } Z = \sum_{t=1}^{T} e_t y_{tg} \qquad (4_g)$$

$$st \quad \sum_{t=1}^{T} d_t y_{tg} \le \sum_{t=1}^{T} d_t n_{tg} - D_g \qquad (5_g)$$

$$y_{tg} \le n_{tg} \qquad \forall t \qquad (6_g)$$

is at least E_g, where

$t = 1,...,T$ is the nurse type index
d_t = number of day shifts worked by a nurse of type t
e_t = number of night shifts worked by a nurse of type t
n_{tg} = number of nurses of type t of grade g or better
$D_g = \sum_{k=1}^{7} C_{kg}$ (i.e. the total day shift requirement at grade g and better)

$E_g = \sum_{k=8}^{14} C_{kg}$ (i.e. the total night shift requirement at grade g and better)

and y_{tg} is the number of nurses of type t and grade g or better assigned to nights.

The expression in the objective function, (4_g) is the total number of night shifts covered, constraint (5_g) ensures that there are sufficient day shifts remaining and constraints (6_g) ensure that the number of nurses allocated does not exceed those available. For a given problem instance P (i.e. for given T, d_t, e_t, n_{tg}, D_g and E_g) we will denote the problem of finding a solution to (4_g), (5_g), (6_g) of value at least E_g by $KS_g(P)$. Clearly, if P is feasible there must be solutions satisfying $KS_g(P)$ for $g = 1$, 2 and 3. However, this is not sufficient, as we need to ensure that the three solutions are compatible with one another. In order to ensure a compatible set of solutions we need to add the constraint:

$$0 \le y_{tg+1} - y_{tg} \le n_{tg+1} - n_{tg} \quad \forall t, g = 1,2 \qquad (7)$$

The left hand inequality ensures that the number of night allocations at grade g+1 and better is at least as many as that at grade g and better, and the right hand inequality ensures that there are sufficient grade g+1 nurses to make up the difference. We will refer to the set of constraints given by $KS_1(P)$, $KS_2(P)$, $KS_3(P)$ and (7) by Cons(P). For a given problem instance, P, we ensure that our constructions always satisfy Cons(P) as follows.

Let $P(R')$ denote a partial solution to instance P in which the number of day and night shifts worked by those nurses in the set R' is known. At the start R' consists of the set of nurses who work both days and nights, and the set of nurses who must be allocated to days together with those who must be allocated to work nights. We define this set to be R^{fixed}. At any stage in the construction let R^+ be the set of nurses already allocated to their shift patterns. Then the remaining nurses can be allocated to days and nights in such a way as to satisfy Cons(P) if there is a feasible solution to Cons($P(R')$) where $R' = R^{\text{fixed}} \cup R^+$. Assume the allocation selected is nurse i to shift pattern j. If $i \in R^{\text{fixed}}$ R' will not change when i is added to R^+. Thus Cons($P(R' \cup \{i\})$) must still be feasible and the allocation can be made i.e. the construction moves onto step 5. If $i \notin R^{\text{fixed}}$, we need to check for feasibility by attempting to find a feasible solution to Cons($P(R^{\text{fixed}} \cup R^+ \cup \{i\})$). If there is a solution then the allocation is made. If not, then if j is a night shift, nurse i must be restricted to days and vice versa. We therefore restrict the set of feasible shift patterns for nurse i, add i to R^{fixed} and return to Step 2 without making the allocation. The restriction is enforced for the remainder of the construction phase only. It remains to be shown that we can determine whether or not there is a feasible solution to Cons($P(R')$) for any P and R' at minimal computational expense.

Our approach is based on the following observations:

Observation 1.
For any R' we can obtain Cons($P(R')$) from Cons(P) by reducing n_{tg} by one for each nurse of type t and grade g or better in R' and reducing D_g and E_g by the number of day and night shifts of grade g or better covered by the assignment of nurses in R'.

Observation 2.
Given a solution Y_g^* to $KS_g(P(R'))$ any solution to the knapsack problem:

$$\max Z = \sum_{t=1}^{T} e_t y_{tg-1} \qquad (4_{g-1})$$

$$st \quad \sum_{t=1}^{T} d_t y_{tg-1} \le \sum_{t=1}^{T} d_t n_{tg-1} - D_{g-1} \qquad (5_{g-1})$$

$$y_{tg-1} \le \min(n_{tg-1}, y^*_{tg}) \qquad (8_{g-1})$$

$$y_{tg-1} \ge \max(0, y^*_{tg} - (n_{tg} - n_{tg-1})) \qquad (9_{g-1})$$

satisfying $Z \ge E_{g-1}$ is a compatible solution to $KS_{g-1}(P(R'))$.

Note that the upper bounds given by (8_{g-1}) are obtained from (6_{g-1}) and the left-hand part of (7) and the lower bounds given by (9_{g-1}) are simply the right-hand part of (7). We denote this problem $KS_{g-1}(P(R')|Y_g^*)$

Observation 3.
The set of all feasible solutions to a bounded knapsack problem with an objective function value at least Z_{target} can be enumerated using a branch and bound tree search in which the branches at level i represent the set of feasible values for the ith variable. The problems represented by the nodes in the tree are smaller knapsack problems in which some of the variables have been fixed. Therefore local lower bounds can be obtained from the linear programming relaxations of these problems. For knapsack problems these linear programmes can be solved trivially without the use of an LP-solver (see Martello and Toth (1990) for details). The tree is pruned by backtracking whenever the bound is less than Z_{target}.

We can use the above observations to solve the appropriate Cons($P(R')$) at each stage of the construction using a hierarchical tree search. The tree at the top of the hierarchy represents a branch and bound search for all solutions of $KS_3(P(R'))$. Each terminal node representing such a solution, Y_3^*, spawns a new tree representing a branch and bound search for solutions to problem $KS_2(P(R')|Y_3^*)$. Similarly, the nodes representing these solutions spawn new trees representing a branch and bound search for solutions to $KS_1(P(R')|Y_2^*)$. The whole structure is searched using a depth first strategy. Once a feasible solution with respect to grade g is found the search proceeds to the next tree in the hierarchy i.e. grade g-1. If the search is not successful then control is returned to the tree relating to grade g until the next feasible solution is found. When a solution is reached at the third hierarchy i.e. grade 1, then a feasible solution to Cons($P(R')$) has been obtained and the allocation of nurse i to shift j can be made. If the search terminates without finding a solution for grade 1 then Cons($P(R')$) is infeasible and the current allocation cannot be accepted. Appendix One illustrates this procedure plus that in the following paragraph.

The above procedure is able to find feasible solutions or confirm that none exists in very short computational time for typical data sets. However, we can make the whole process more efficient if we note that we do not necessarily need to solve a new knapsack problem every time we consider a new allocation. At any point in the construction we have the y_{tg} values for a feasible solution to Cons($P(R')$). If we then consider allocating nurse i of type t and grade g to nights, then the existing solution will still be feasible as long as $y_{tg} \geq 1$ and (if $g \neq 1$) $y_{tg} > y_{tg-1}$. Similarly a day allocation will be feasible if $n_{tg} - y_{tg} \geq 1$ and $n_{tg} - y_{tg} > n_{tg-1} - y_{tg-1}$ If these conditions hold, we can allocate nurse i without solving a new knapsack problem. Having made the allocation we update the variables and constants to reflect the new solution.

The process is summarised in Figure 1.

(Insert Figure 1 around here)

We incorporate this look-ahead rule based on the knapsack calculations described in this section into each of our four construction heuristics to form four further variants. The best of these is then subject to further modifications. These are described in the following section.

6. FURTHER ENHANCEMENTS

In this section we consider three enhancements to the above implementations. The first is a simple extension of the local search phase, namely the acceptance of non-worsening moves within the neighbourhood N_E. The remaining two utilise the cyclic structure of the GRASP algorithm and use information from previous cycles. This use of feedback takes two forms. The first involves the use of a threshold on the preference cost during the search for a feasible solution, while the second is a diversification mechanism based on the values of the knapsack variables.

1. Plateau moves.
The introduction of the swap neighbourhood N_E increases the probability of generating moves that do not change the preference cost. It is possible that a sequence of such moves may lead to a point where further improvements can be made. On the other hand allowing all such moves may mean that there is not sufficient pressure for improvement. We therefore experimented with accepting such moves in N_E with a given probability.

2. Preference cost threshold.
According to the evaluation function given in Section 4, if the cover is improved, a move is accepted regardless of any increase in the preference cost. Thus the influence of the preference cost in the score functions in the construction phase may be destroyed. However, it is not sensible to reject all moves that increase the preference cost as these are often necessary in reaching feasibility. Instead we place a threshold on the increase. Such a threshold will be data dependent. For example, it must be greater than the preference cost of the best feasible solution. We use feedback from previous cycles to impose such a threshold in two ways. Firstly, we do not impose a threshold until we reach a cycle where a feasible solution has been obtained. For subsequent cycles we set a threshold of best_feasible_solution + Q, where Q is a dynamic threshold, also determined by the results of previous cycles. This means that we can iterate towards a value of Q that is large enough to allow the improvement phase to find feasible solutions, but is small enough to preserve the quality of the initial solution as far as possible. Q is initialised with a value q_0, and increased according to the function $Q = Q + \alpha Q$ at the end of each cycle, where α is a function of the cover_cost of the solution produced in that cycle. Thus Q increases whenever a cycle fails to find a feasible solution. A move is then accepted during the improvement phase if cover_cost improves and preference cost is less than best_feasible solution + Q.

3. Knapsack based diversification.
The final enhancement addresses the issue of diversification. It is designed to ensure that different cycles visit different areas of the solution landscape based on the numbers of nurses of each type and each grade allocated to days/nights. These values are precisely the sets Y_g of y variables returned by the knapsack calculations. To ensure diversity we need to ensure that no sets Y_g occur too frequently. This is achieved by generating initial solutions as before, and rejecting those relating to sets Y_g whose frequency is above a given threshold. This threshold is given by:

$Freq_{min} + V_Y/3$ where $Freq_{min}$ is the minimum frequency of all the Y_g vectors accepted so far and V_Y is the number of different vectors generated so far. As the construction process is very fast, this is feasible as long as the number of rejected solutions does not get excessive, as is the case if there are some vectors with a very small probability of being generated. This situation is addressed by modifying the above process so that if no pattern satisfying the acceptance conditions is sampled after 20 trials, then one of the solutions relating to the vector with the minimum frequency of these 20 is accepted.

7. EXPERIMENTS AND RESULTS

7.1 Initial results.

Our experiments are based on fifty-two datasets from a large UK hospital. These datasets have also been considered by Dowsland (1998), Dowsland and Thompson (2000), Aickelin and Dowsland (2000, 2004) and Burke et al. (2003b). In all cases, the optimal solutions are known.

Our initial experiments were designed to evaluate the effect of the knapsack over the full range of data sets and construction heuristics, using a range of values for the parameters n (size of the candidate list), w_c and w_p (the weights on the cover cost and the preference cost in the objective function), and to identify the most promising for further investigation. The role of n is to balance aggression and diversity within the construction with smaller values of n leading to more emphasis on the better scoring options and larger values of n leading to more diversity. The value of n will obviously be partially dependent on the number of options available at each stage of the construction process. This corresponds to around 60 shift patterns for the *cover* and *combined* heuristics, around 1,500 nurse-pattern pairs for the *holistic* heuristic and around 25 nurses for the *last chance* heuristic. Thus the range of n will vary according to the heuristic. For the *combined*, *holistic* and *last chance* heuristics we also need to define w_c and w_p. Table 1 shows the parameters used.

(Insert Table 1 around here)

Each construction method was run with and without the knapsack routine, giving 8 different GRASP implementations. For each one, every combination of parameters was run once on each of the 52 data sets, using $nrep = 100$. The local search phases used neighbourhoods N_C and N_S, selected with equal probability for each move until feasibility was achieved and using neighbourhood N_S only after that point. (N_E was not included at this stage as some of the variants frequently failed to find feasible solutions.) Since sampling is without replacement, the search was allowed to continue until the local optimum had been reached.

In order to be able to combine the results from different data sets, the results have been standardised by subtracting the optimal cost. Figure 2 shows plots of the percentage of feasible solutions against the average standardised preference cost for

these feasible solutions, over all 100 cycles, for each of the eight heuristics. Each point represents a single combination of n, w_c and w_p.

(Insert Figure 2 around here)

It is clear from Figure 2 that the percentage of feasible solutions increases significantly with the use of the knapsack routine. A closer examination of the data also revealed that of the 228 parameter combinations without the knapsack only 5 generate at least one feasible solution for all 52 datasets, and these have much poorer solutions in terms of the preference cost than those generated by heuristics utilising the knapsack algorithm. Thus there is clear benefit in incorporating the knapsack into the construction phase.

For all but the *last chance* constructions, there is evidence that smaller values of n outperform larger ones suggesting that the added diversity of larger n is less beneficial than a greedier approach. Of the knapsack heuristics, Figure 2 suggests that the *last chance* variant is the most promising. In order to confirm this, each variant was run 10 times for 100 cycles on each of the 52 data sets using a single set of parameter values. Unlike the above experiment where results were averaged over all cycles we now consider each block of 100 cycles as a complete GRASP run and record only the best solution over these cycles. As all four heuristics produce a high proportion of feasible solutions the parameter choices were based on the average preference score over the 52 data sets and are shown in Table 2, along with the mean standardised preference costs.

(Insert Table 2 around here)

The results confirm the superiority of the *last chance* heuristic. It has an average preference cost lower than all other heuristics and about 95% of the cycles create feasible schedules. These results are promising but only provide an overall view averaged over all datasets. In the next subsection we evaluate the performance in more detail on each dataset and compare the results with those obtained when the neighbourhood N_E and the enhancements suggested in Section 6 are included.

7.2. More detailed results.

Although the amalgamated results in Figure 2 suggest that our GRASP based on the *last chance* construction together with the knapsack performs reasonably well, the variability of the results is apparent when we consider each data set separately. Figure 3(a) shows the number of runs returning an optimal solution and the number returning a result within 3 of the optimum. It is clear that there are many data sets for which the optimum is not found in any of the 10 runs, and several where there are no solutions even close to the optimum.

(Insert Figure 3 around here)

Figure 3(b) shows the results of expanding the local search to include the extended neighbourhood N_E. This was implemented by replacing N_S with N_E once the first feasible solution in each cycle has been found. The results have clearly improved but still fail to match those of Dowsland and Thompson (2000).

Further improvements were obtained by the introduction of plateau moves. The probability of accepting moves that do not change the preference cost was varied from 0% to 100% in steps of 25%, with 75% giving the best results. These results are given in Figure 3(c) and improve the results to the point where all instances are solved to optimality at least once.

Finally Figure 3(d) shows the results of adding the two feedback related enhancements. For the preference cost threshold, parameter choices $q_0 = 10$ for the initial threshold value and $\alpha = 4*\text{cover\_cost}$ for the rate of increase were found to work well. The knapsack based diversification was added as stated in Section 6.

Comparing these results, with those obtained previously, we can see that the algorithm now finds the optimal solution in every case for all but two datasets and nine times out of 10 for the remaining two. This is a better average performance than that reported in Thompson and Dowsland and has been obtained without recourse to complex neighbourhoods, which may be data specific.

7.3 The role of the construction heuristic.

The results of Section 7.1 suggested that even after the inclusion of the knapsack routine there was a clear difference between the different construction heuristics in terms of solution quality. Given the dramatic improvements obtained by the use of the extended neighbourhood, plateau moves and feedback mechanisms it is no longer clear that this is still the case. All four heuristics were therefore rerun 10 times on each data set with all the enhancements included. Feasible solutions were produced in all runs, with optimal solutions being produced for all 10 runs using all four heuristics for all but 8 of the 52 data sets. The results for these 8 are shown in Table 3. Once again results have been standardised by subtracting the optimal cost.

(Insert Table 3 around here)

The performance of all four algorithms is much improved but the *last chance* heuristic remains the best and most consistent performer. This, combined with the vast improvements offered by the knapsack routine, show that it is worthwhile investing some thought, effort and experimentation into the choice of construction heuristic when developing a GRASP implementation. The results from this best algorithm are compared with those quoted by others in Table 4. This shows the best results obtained from GRASP, Tabu Search (Dowsland and Thompson (2000)), Genetic Algorithm (Aickelin and Dowsland (2004)) and Estimated Distribution Algorithm (Aickelin and Li (2007)). The rows marked IP are the optimal solutions obtained from the integer programming formulation (Fuller 1998), which required several hours of computational time for some datasets. GRASP outperforms the EDA in terms of preference cost and the GA in terms of both preference cost and feasibility. As stated above when we compare average performance, GRASP also outperforms the Tabu Search approach. The run-times required by GRASP are also impressive. All experiments were performed on a 1 MHz Pentium III PC with 256 MB of RAM, and required between 1 and 4 seconds of CPU time per cycle. Although this is slightly slower than the Tabu Search approach, it is significantly faster than an exact IP approach. The time required to check for knapsack feasibility is almost negligible,

while using plateau moves has a significant impact on the time taken by the improvement phase. When averaged over all 10 runs for all 52 datasets, using 75% plateau moves, construction takes approximately 2.5% of the solution time and checking the knapsack takes less than 0.005% of the construction time. Without plateau moves construction takes 85% of the solution time.

(Insert Table 4 around here)

8. CONCLUSIONS AND FURTHER WORK

This paper has considered a nurse-scheduling problem in which the conflict between feasibility and optimality is an issue that must be addressed by any successful solution approach. We have developed a GRASP approach that outperforms other heuristic approaches for this particular variant of the problem. The key to achieving this was the incorporation of a look-ahead facility based on a knapsack model of a relaxation of the problem, to ensure that the solutions produced by the construction phase were relatively easy to convert into feasible solutions during the improvement phase. The paper also adds to the body of evidence showing the importance of a suitable neighbourhood structure and evaluation/acceptance criteria during the improvement phase, as well as the benefit of a feedback mechanism to allow for a learning process from one cycle to the next. Feedback was used in two ways. Firstly, as the basis of a diversification strategy to ensure that the construction phase produced solutions from different areas of the search space, and secondly, to adjust the acceptance conditions for the improvement phase. Both played an important role in providing high quality solutions on the more difficult data sets.

The need to balance feasibility with optimality in all parts of the search was evidenced not only by the improvements resulting from the knapsack routine, but also by the introduction of the preference cost threshold with N_C and N_S. The results also show that the choice of construction heuristic is important, although it is not clear why the *last chance* decoder proved to be the best of the four. Three possible factors are its use of a look-ahead strategy, the more aggressive use of the best shift-pattern for each nurse, and the fact that the number of options at each stage in the construction is smaller than for the other three.

This work has shown that, for this particular problem, a GRASP approach can be very effective in solving highly constrained problems, as long as the feasibility of the solution is given sufficient consideration during the construction. It would be interesting to use a similar approach on other problems, in particular the use of relaxations or other problem specific information to guide the construction phase towards feasible, or easily repairable, solutions. We are currently working on these ideas with respect to a scheduling problem arising in the allocation of medical students to 'firms' of consultants in order to cover a given set of clinical disciplines.

REFERENCES

Abramson, D. (1991). "Constructing school timetables using simulated annealing: sequential and parallel algorithms", *Man. Sci.*, **37**, 98-113.

Aickelin, U. and K. A. Dowsland (2000). "Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem," *J. Sched.*, **3**, 139-153.

Aickelin, U. and K. A. Dowsland (2004). "An Indirect Genetic Algorithm for a nurse-scheduling problem," *Comput. Oper. Res.* **31**, 761-778.

Aickelin, U. and J. Li (2007). "An Estimation of Distribution Algorithm for Nurse Scheduling," to appear in *Annals of Oper. Res.*

Aiex, R.M., S. Binato, and M.G.C. Resende (2003). "Parallel GRASP with path-relinking for job shop scheduling," *Parallel Computing* **29**, 393-430.

Bellanti, F., G. Carello, F. Della Croce, and R. Tadei (2004). "A greedy-based neighbourhood search approach to a nurse rostering problem," *Eur. J. Oper. Res.* **153,** 28-40.

Berrada, I., J. A. Ferland, and P. Michelon (1996). "A Multi-objective Approach to Nurse Scheduling with both Hard and Soft Constraints," *Socio-Economic Planning Sciences,* **30**, 183-193.

Binato, S., W. J. Hery, D. M. Loewenstern and M. G. C. Resende (2001). "A GRASP for job shop scheduling", *Essays and surveys in metaheuristics*, **15**, 81-100.

Brusco, M.J. and L. W. Jacobs (1995). "Cost Analysis of Alternative Formulations for Personnel Scheduling in Continuous Operating Organizations." *Eur. J. Oper. Res* **86**, 249-261.

Burke, E., P. Cowling, P. De Causmaecker and G.Vanden Berghe (2001). "A Memetic Approach to the Nurse Rostering Problem", *Applied Intelligence*, **15**, No. 3, 199-214.

Burke, E., P. De Causemaecker, S. Petrovic, and G. Vanden Berghe (2003a), "Variable Neighbourhood Search for Nurse Rostering Problems", In Mauricio G. C. Resende and Jorge Pinho de Sousa (eds.), *METAHEURISTICS: Computer Decision-Making*, Chapter 7, Kluwer, pp. 153-172.

Burke, E., P. De Causemaecker and G. Vanden Berghe (1999). "A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem", in B. McKay et al. (eds.), *Simulated Evolution and Learning*, Springer, Lecture Notes in Artificial Intelligence, Vol. 1585, pp.187-194 .

Burke, E., P. De Causemaecker, G. Vanden Burghe and H. Van Landeghem (2004). "The State of the Art of Nurse Rostering," *J. Sched.*, **7**, 441-499.

Burke, E., G. Kendall and E. Soubeiga (2003b). "A Tabu-Search Hyperheuristic for Timetabling and Rostering," *J. Heuristics* **9**, 451-470.

Cheang, B., H. Li, A. Lim and B. Rodrigues (2003). "Nurse rostering problems – a bibliographic survey," *Eur. J. Oper. Res.* **151**, 447-460.

Dowsland, K. A. (1998) "Nurse scheduling with tabu search and strategic oscillation." *Eur. J. Oper. Res.* **106**, 393-407.

Dowsland, K. A. and J. M. Thompson (2000). "Solving a nurse-scheduling problem with knapsacks, networks and tabu search," *J. Oper. Res. Soc.* **51**, 825-833.

Drexl, A. and F. Salewski (1997). "Distribution requirements and compactness constraints in school timetabling", *Eur. J. Oper. Res.* **102**, 193-214.

Ernst, T., H. Jiang, M. Krishnamoorthy, and D. Sier (2004). "Staff scheduling and rostering: A review of applications, methods and models," *Eur. J. Oper. Res.* **153**, 3-27.

Feo, T. A., M. G. C. Resende and S.H. Smith (1994). "A greedy randomised adaptive search procedure for maximum independent set," *Oper. Res.* **42**, 860-878.

Fleurent, C. and F. Glover (1999). "Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory," *INFORMS J. Computing*, **11**, 198-204.

Fuller E. (1998). "Tackling Scheduling Problems Using Integer Programming". Master Thesis, University of Wales Swansea, United Kingdom.

Gupta, S. R. and J. S. Smith (2006). "Algorithms for single machine total tardiness scheduling with sequence dependent setups", *Eur. J. Oper. Res.* **175**, 722-739.

Gutjahr, W. J. and M. S. Rauner (2007). "An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria", *Comput. Oper. Res.* **34**(3), 642-666.

Isken, M. (2004). "An implicit tour scheduling model with applications in healthcare," *Annals Oper. Res.* **128**, 91-109.

Jaumard, B., Semet, F. and T. Vovor (1998). " A generalized linear programming model for nurse scheduling", *Eur. J. Oper. Res.* **107**, 1-18.

Laguna, M.and J. L. González-Velarde (1991). "A search heuristic for just-in-time scheduling in parallel machines," *J. Intelligent Manufacturing*, **2**, 253-260.

Laguna, M. and R. Martí (2001). "A GRASP for coloring sparse graphs", *Computational Optimization and Applications* **19**, 165-178.

Lim A., B. Rodrigues and X. Zhang (2006). "A simulated annealing and hill-climbing algorithm for the travelling tournament problem", *Eur. J. Oper. Res.* **174**, 1459-1478.

Martello, S. and P. Toth (1990). "Knapsack Problems," Wiley, Chichester.

Meyer auf'm Hofe, H. (2000). "Nurse rostering as constraint satisfaction with fuzzy constraints and inferred control strategies", *DIMACS Workshop on Constraints Programming and Large Scale Discrete Optimisation*, 67-100.

Michalewicz, Z. and D. B. Fogel. (2004). "How to Solve It: Modern Heuristics" Springer Verlag (Berlin and Heidelberg).

Moz, M. and M. V. Pato (2007). "A genetic algorithm approach to a nurse re-rostering problem", *Comput. Oper. Res.*, **34**(3), 667-691.

Parr, D. and J. Thompson (2007). "Solving the multi-objective nurse scheduling problem with a weighted cost function," to appear in *Annals of Oper. Res.*

Petrovic, S., G. Beddoe, and G. Vanden Berghe (2003). "Storing and adapting repair experiences in personnel rostering", in E.K. Burke and P. De Causemaecker (eds), *Practice and Theory of Automated Timetabling*, Fourth International Conference, Gent, Springer, *Lecture Notes in Computer Science*, **2740**, 149-166.

Resende, M. G. C. (2001). "Greedy Randomized Adaptive Search Procedures (GRASP)," *Encyclopedia of Optimisation*, Kluwer Academic Press, **2**, 373-382.

Rosenbloom, E.S. and N. F. Goertzen (1987). "Cyclic nurse scheduling," *Eur. J. Oper. Res.*, **31**, 19-23.

Thompson, J. M. and K. A. Dowsland (1996). "Variants of simulated annealing for the examination timetabling problem", *Annals of Oper. Res.*, **63**, 637-648.

Wright, M. B. (1991). "Scheduling cricket umpires", *J. Oper. Res. Soc.* **42**, 447-452.

Zhu, Y and A. Lim (2006). "Crane scheduling with non-crossing constraint", *J. Oper. Res. Soc.* **57**, 1464-1471.

Tables

Table 1. The combinations of parameters investigated.

Heuristic	N	w_c	w_p	Total combinations
Cover	3,12,20	—	—	3
Combined	3,12,20	1,2,...,5	1,2,...,5	75
Holistic	3,10,60	1,2,...,5	1,2,...,5	75
Last chance	3,6,10	1,2,...,5	1,2,...,5	75

Table 2. Comparison of 4 knapsack heuristics.

Heuristic	n	w_c	w_p	Cost	% feasible cycles.
Cover + kn.	20	—	—	23	76
Combined + kn.	3	2	5	6	89
Holistic + kn.	3	1	3	4	87
Last chance + kn.	3	2	5	2	95

Table 3. Average cost of best solutions obtained from 10 runs of 100 GRASP cycles for each of the four heuristics using the extended neighbourhood, plateau moves, preference cost threshold and knapsack based diversification.

Data Set	*Cover*	*Combined*	*Holistic*	*Last chance*
2	0.2	0.1	0	0.1
20	0	13.5	9.7	0
25	0	1	0	0
31	3.6	1.2	3.2	0.4
33	0.8	0	0	0
42	0.1	0	0.6	0
44	0	10.8	5.1	0
46	0	0.6	0	0

Table 4. Comparison of best results for all 52 datasets.

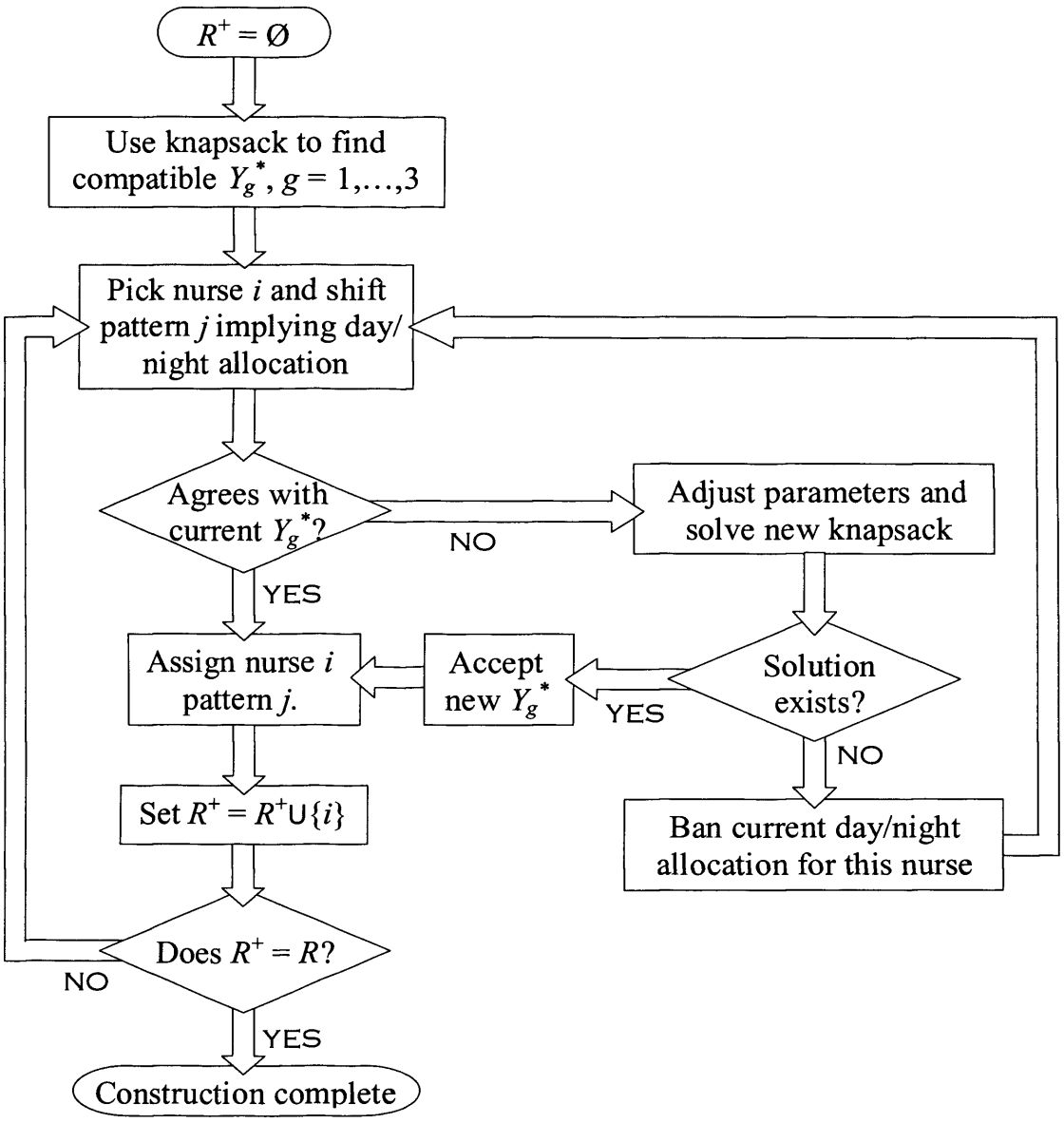
Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
IP	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
GRASP	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
Tabu	8	49	50	17	11	2	11	14	3	2	2	2	2	3	3	37	9	18	1	7	0	25	0	1	0	48
GA	8	50	50	17	11	2	11	15	3	4	2	2	2	3	3	38	9	19	1	8	0	26	0	1	0	48
EDA	8	56	50	17	11	2	14	15	14	2	2	3	3	4	4	38	9	19	10	7	1	26	1	1	0	52

Dataset	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
IP	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
GRASP	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
Tabu	2	63	15	35	62	40	10	38	35	32	5	13	5	7	54	38	22	19	3	3	3	4	27	107	74	58
GA	2	63	141	42	166	99	10	48	35	41	5	14	5	8	54	38	39	19	3	3	4	6	30	211	-	-
EDA	28	65	109	38	159	43	11	41	46	45	7	25	8	8	55	41	23	24	6	7	3	5	30	109	171	67

N.B. The GA did not find any feasible solutions for datasets 51 and 52.

Figures

Figure 1. Flow chart showing implementation of the knapsack model.



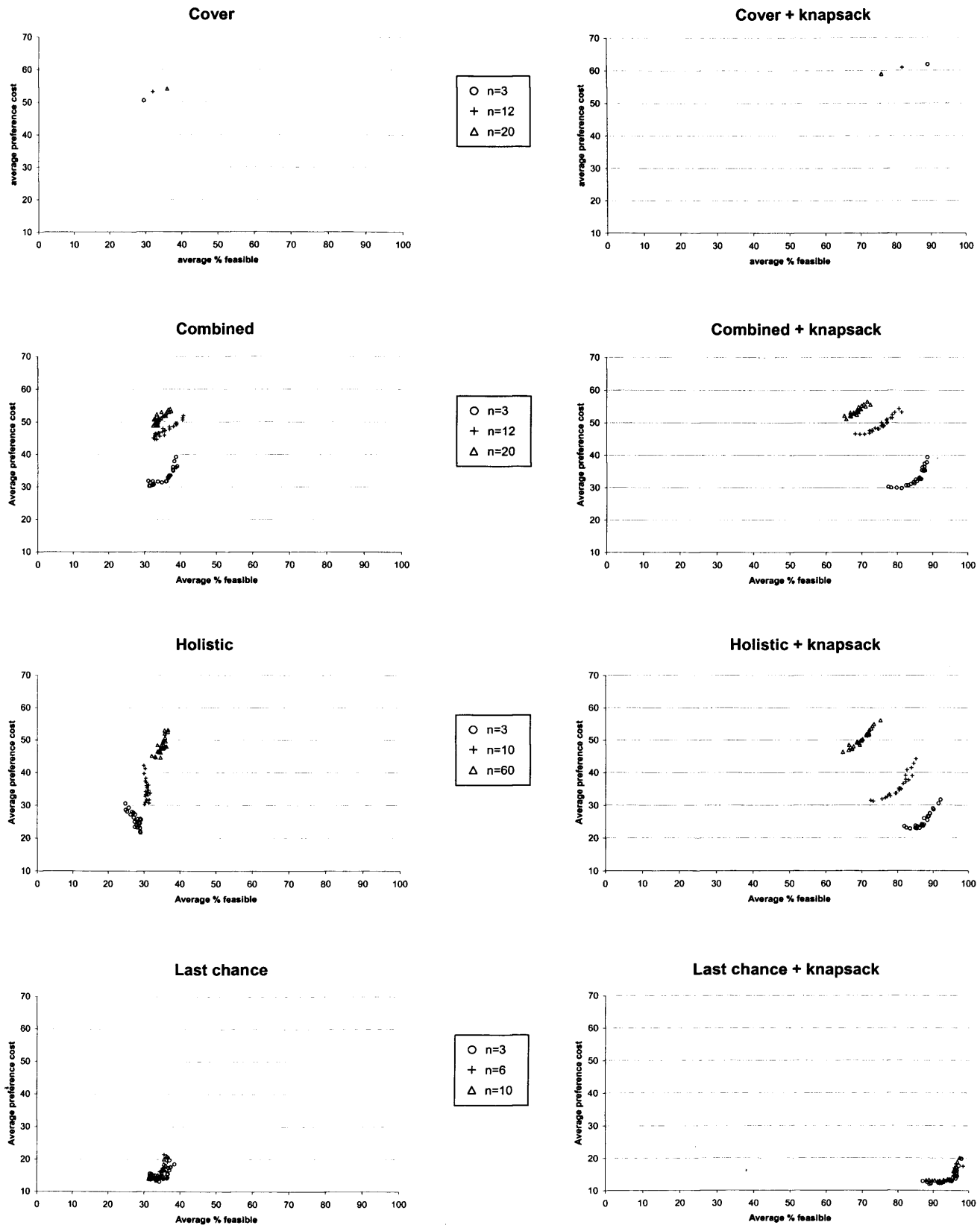


Figure 2. Plots of feasibility against average preference cost for the eight heuristics.

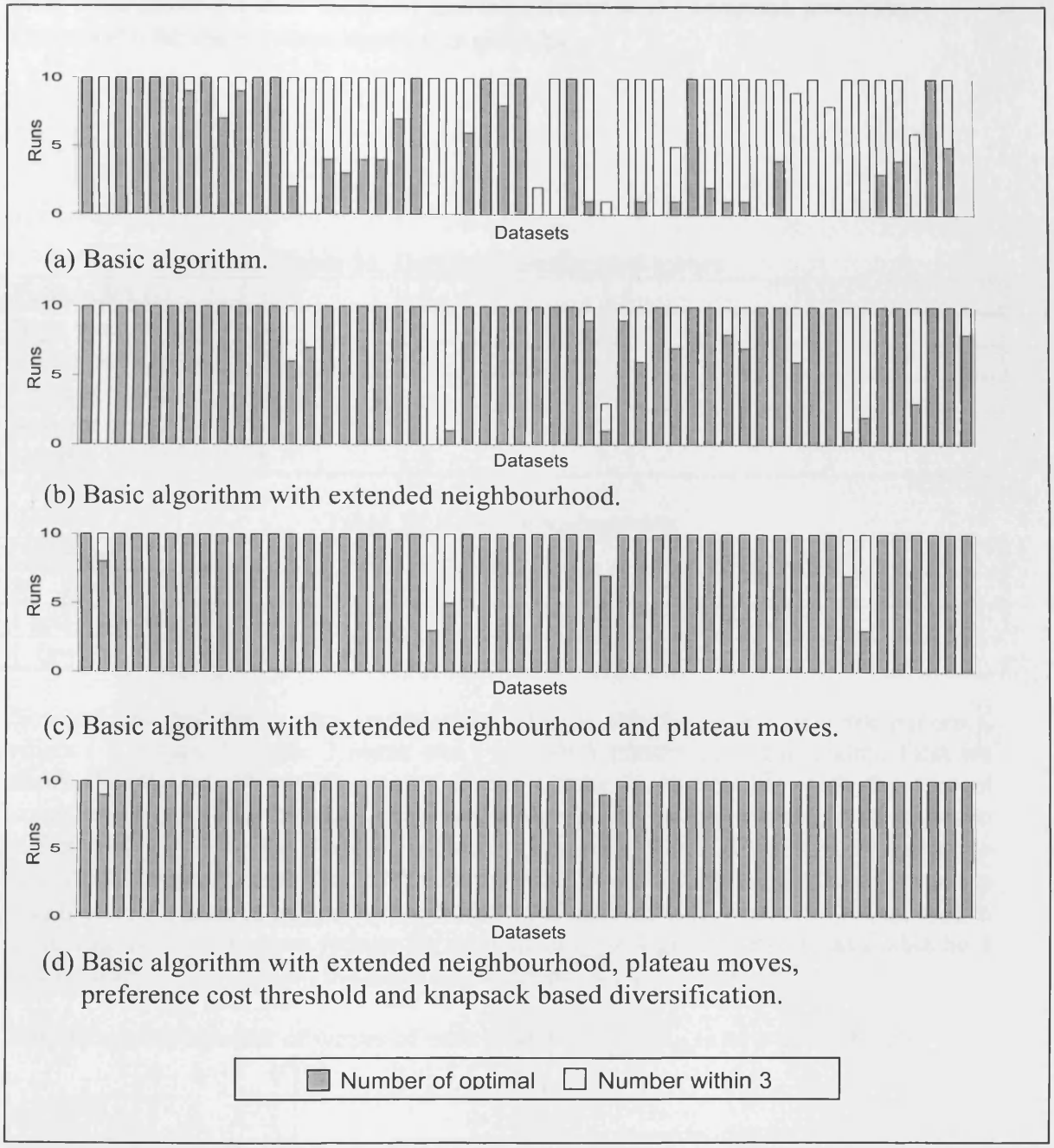


Figure 3. A breakdown of the results for all 52 datasets from 10 runs using different local search extensions.

Appendix One. Worked example of the knapsack look-ahead procedure.

Suppose we have a situation in which the set of nurses as yet unallocated who are free to work days or nights (i.e. the set R - R') are as given in Table 5a, the shifts that they must cover are as given in Table 5b, and the solution to the knapsack problems Cons($P(R')$) for the previous iteration is given by

$$Y^* = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

Table 5a. Details of unallocated nurses

Type index (t)	1	2	3	4
Days worked (d_t)	3	5	4	3
Nights worked (e_t)	3	4	3	2
Number grade 1	0	2	2	0
Number grade 2	1	3	2	2
Number grade 3	1	4	3	3

Table 5b. Cover Requirements

Grade (g)	Days required	Nights required
All (g=3)	26	14
1 and 2 (g=2)	19	10
1 (g=1)	6	7

Now assume that the greedy construction process selects nurse i to work pattern j, where i is a type 4, grade 2 nurse and j is a shift pattern covering nights. First we check the current Y^* matrix to see if our choice is compatible with the current knapsack solution. As the matrix represents a possible feasible allocation of nurses to nights, and $y_{24}^* = 0$, the current solution is not compatible. Thus it is necessary to resolve the knapsack problems to determine whether or not the suggested allocation is feasible i.e. we need to search for a solution to Cons($P(R' \cup \{i\})$) where i is assumed to be on nights. We therefore reduce the number of type 4 grade 2 nurses available by 1 and subtract e_4 (i.e. 2) from the night requirements for $g = 2$ and $g = 3$.

The cumulative number of nurses of each type available, n_{tg},is now given by the matrix

$$N = \begin{pmatrix} 0 & 2 & 2 & 0 \\ 1 & 3 & 2 & 1 \\ 1 & 4 & 3 & 2 \end{pmatrix}$$

The target values E_g are as given by the new night requirements (7,8,12) and the capacities of the knapsacks defined by the right-hand sides of constraints (5_g) are given by the vector (10,10,15). The tree relating to the search for a feasible solution is shown in Figure 4.

At the top level of the search, node A represents the knapsack problem KS_3 relating g = 3, where the target value $E_3 = 12$, knapsack capacity = 15 and the upper bounds on

the variables are given by the bottom row of matrix N. The top segment of the Figure, i.e. the region denoted g=3, shows a tree search for all feasible solutions to this problem with objective value at least 12, using bounds based on the linear programming relaxation of the problem to prune the tree. Note that the types have been ordered in decreasing e_t/d_t order and the branches relating to each variable are ordered in decreasing value order. This improves the efficiency of the search.

The first solution is found at node B and is given by $Y_3^* = (1,1,1,1)$ i.e. 1 nurse of each type working nights and the remainder on days. Having reached the solution we spawn a new tree to search for solutions to the problem $(KS_2|Y_3^*)$ starting at node B′. The target value is now 8 and the knapsack capacity is 10. The vector of upper bounds (given by the right-hand side of constraint set (8_2)) = (1,1,1,1) and the vector of lower bounds (given by the right-hand side of (9_2)) = (1,0,0,0). Thus y_{12} is fixed at $y_{12} = 1$. Substituting the fixed value leaves us with a problem in the three remaining variables with an adjusted target of 8-3 = 5 and adjusted knapsack capacity of 10–3 = 7. The tree search corresponding to this problem starts at root node B′ and the first feasible solution is found at E corresponding to $Y_2^* = (1,1,0,0)$.

We now spawn a new tree to search for solutions to $(KS_1|Y_2^*)$. The target is 7 and the knapsack capacity is 10. The upper bounds are given by (0,1,0,0) and the lower bounds by (0,0,0,0). Thus all variables except y_{32} are fixed at 0 and as y_{32} has an upper bound of 1 the problem is clearly infeasible as we cannot meet the target value. We therefore return to node E and continue searching the tree corresponding to $(KS_2|Y_3^*)$. A second solution at this level in the hierarchy is found at F with $Y_2^* = (1,0,1,1)$. This spawns a new problem $(KS_1|Y_2^*)$ with upper bounds (0,0,1,0) and lower bounds (0,0,0,0). Once again we have a problem in just one variable which is clearly infeasible so control is returned to node F. Continuing the search at this level shows that there are no more feasible solutions to $(KS_2|Y_3^*)$ and control is returned to point B and the search for solutions to problem A continues.

The next solution is at C corresponding to $Y_3^* = (1,0,3,0)$. This spawns a new problem for $(KS_2|Y_3^*)$ at C′ with upper bounds (1,0,3,0) and lower bounds (1,0,2,0). This becomes a problem in y_{23} with a target of 5 and knapsack capacity = 7. This problem is clearly infeasible as the lower bound on y_{23} causes the knapsack capacity constraint (5_2) to be violated. Control therefore returns to C. The next solution occurs at D corresponding to $Y_3^* = (0,3,0,0)$ with upper bounds (0,3,0,0) and lower bounds (0,2,0,0). The tree for the resulting problem in y_{22} starts at D′ yielding a feasible solution at G corresponding to $Y_2^* = (0,2,0,0)$. This spawns problem $(KS_1|Y_2^*)$ at G′ with upper bounds (0,2,0,0) and lower bounds (0,1,0,0) and yielding a feasible solution at H. Thus there is a feasible solution that is compatible over all grades given by

$$Y^* = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

We therefore make the allocation of nurse i to pattern j. Had the search failed to find a feasible solution nurse i would have been added to the set R^{fixed} (tagged as days only), the matrix Y^* restored to its previous set of values and n_{24} and n_{34} reduced by 1 and D_2 and reduced by 3.

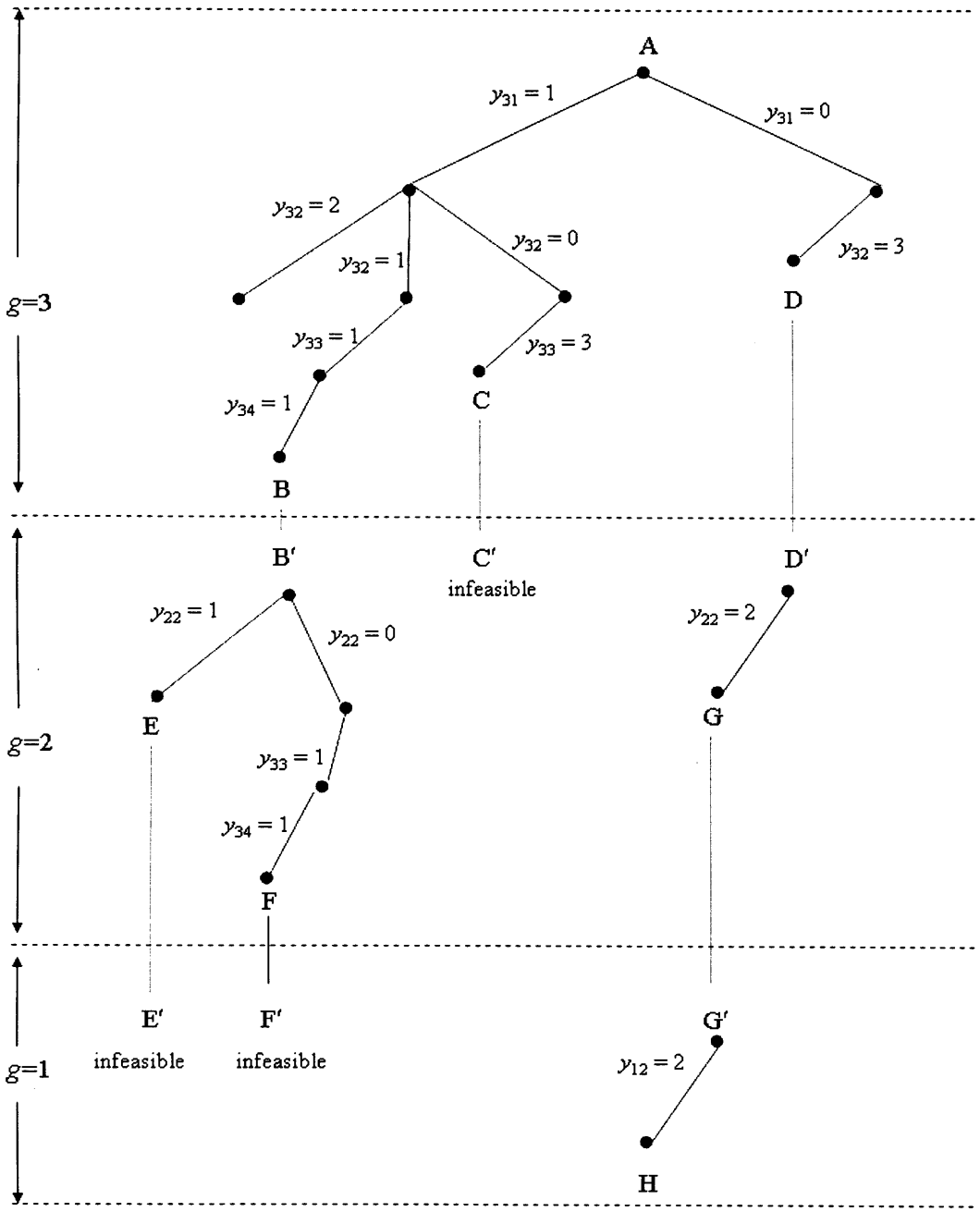


Figure 4. Search tree for finding a feasible solution