DEVELOPMENT AND APPLICATION OF HYPERHEURISTICS TO PERSONNEL SCHEDULING

A thesis submitted to the University of Nottingham for

the degree of Doctor of Philosophy

by

Eric Soubeiga: Ingénieur d'Etat, DEA in Operational Research

School of Computer Science and Information Technology

The University of Nottingham

June 10, 2003

Abstract

This thesis is concerned with the investigation of hyperheuristic techniques. Hyperheuristics are heuristics which choose heuristics in order to solve a given optimisation problem. In this thesis we investigate and develop a number of hyperheuristic techniques including a hyperheuristic which uses a choice function in order to select which low-level heuristic to apply at each decision point. We demonstrate the effectiveness of our hyperheuristics by means of three personnel scheduling problems taken from the real world. For each application problem, we apply our hyperheuristics to several instances and compare our results with those of other heuristic methods. For all problems, the choice function hyperheuristic appears to be superior to other hyperheuristics considered. It also produces results competitive with those obtained using other sophisticated means. It is hoped that

- hyperheuristics can produce solutions of good quality, often competitive with those of modern heuristic techniques, within a short amount of implementation and development time, using only simple and easy-to-implement low-level heuristics.

- hyperheuristics are easily re-usable methods as opposed to some metaheuristic methods which tend to use extensive problem-specific information in order to arrive at good solutions.

These two latter points constitute the main contributions of this thesis.

Key words: Hyperheuristic, Heuristic, Local Search, Optimisation, Personnel Scheduling.

Acknowledgements

In my opinion, undertaking a PhD is the second biggest commitment after marriage in a young man's life. In my case this would not have been possible without the help of many people. I wish to thank them all for their valuable support.

I would like to thank the University of Nottingham for funding my PhD research. I started my PhD with Peter Cowling who helped me get funding for it. Graham Kendall joined Peter soon after in the supervision of my PhD research. I would like to thank both Doctor Graham Kendall (my principal supervisor) and Professor Peter Cowling (my external supervisor). I have enjoyed working with both of them and would like to thank them both for training me to do scientific research. Your support, help and advice have been very valuable to me.

I would like to thank Professor Edmund Burke for his help, support and professionalism.

I would also like to thank both administrative and academic members of the Automated Scheduling, optimisAtion and Planning (ASAP) research group. You have all contributed to making my PhD work within ASAP an enjoyable experience.

To Professor Gerd Finke for acquainting me with research in the field of combinatorial optimisation.

To my parents, my two brothers and my sister over in Burkina Faso.

To Dr Helen Ashman for providing me with data for the problem of chapter 7.

To both Dr Kath Dowsland and Dr Uwe Aickelin for providing me with data for the problem of chapter 8.

Many thanks to you all and may God bless us.

Contents

1 Introduction **1**

1.1 Structure of thesis 6

1.2 Academic papers produced 7

2 Related work: hyperheuristics **10**

2.1 Introduction 10

2.2 Single-heuristic techniques 11

2.3 Multiple heuristic or neighbourhood search techniques 15

2.3.1 Without learning 15

2.3.2 With learning 18

2.3.3 Adaptive problem solving in AI systems 24

2.4 Conclusion 27

3 Related work: exact & metaheuristic methods **30**

3.1 Introduction 30

3.2 Exact methods 31

3.3 Heuristic methods 33

3.3.1 Point-based methods 34

3.3.2 Population-based methods 39

3.4 Conclusion 45

4 Related work: personnel scheduling 47

4.1 Introduction 47

4.2 The general personnel scheduling problem 51

4.2.1 Recent surveys 52

4.2.2 Modelling flexibility in personnel scheduling 55

4.2.3 Additional flexibility using heuristics 60

4.2.4 When theory leads to the development of new algorithms . . . 72

4.2.5 Other exact methods 75

4.3 Specific applications 77

4.4 Summary 80

5 Hyperheuristics for personnel scheduling 84

5.1 Introduction 84

5.2 Designing hyperheuristics 86

5.2.1 Level 0: Problem representation 86

5.2.2 Level 1: Low-level heuristics 99

5.2.3 Level 2: hyperheuristic (high-level heuristic) 101

5.2.4 Guidelines for designing hyperheuristics 106

5.2.5 Summary 109

5.3 Hyperheuristics developed 109

5.3.1 Simple hyperheuristics 112

5.3.2 A choice function hyperheuristic 114

5.3.3 A simulated annealing hyperheuristic 131

5.4 Summary 134

6 Application to sales summit scheduling 137

6.1 Introduction 137

6.2 The sales summit scheduling problem 138

6.2.1 Problem description 138

6.2.2 Problem formulation 139

6.3 Experimental study 142

6.3.1 Problem instances 142

6.3.2 The low-level heuristics 146

6.3.3 Simple, simulated-annealing and choice function hyperheuristics147

6.3.4 Effectiveness and learning ability of the choice function hyper-heuristic . . . 153
6.3.5 Experiments with a smaller set of low-level heuristics . . . 160
6.3.6 Experiments with a larger set of low-level heuristics . . . 162
6.4 Conclusions . . . 165

7 Application to Presentation scheduling 169

7.1 Introduction . . . 169
7.2 Scheduling of project presentations . . . 170
7.2.1 Problem description . . . 170
7.2.2 Problem formulation . . . 172
7.3 Experiments . . . 175
7.4 Conclusions . . . 183

8 Application to nurse scheduling 186

8.1 Introduction . . . 186
8.2 The nurse scheduling problem . . . 187
8.2.1 Problem description . . . 187
8.2.2 Problem formulation . . . 188
8.3 Experiments . . . 194
8.4 Conclusions . . . 203

9 Conclusions **205**

List of Figures

1.1 General hyperheuristic framework 4

5.1 Design issues for the development of a hyperheuristic 87

5.2 General hyperheuristic framework when dealing with partial solutions 89

5.3 General hyperheuristic framework when dealing with complete solutions 92

5.4 The general framework of hyperheuristics developed 110

5.5 Choice function hyperheuristic framework 127

5.6 Simulated annealing hyperheuristic framework 133

6.1 Interplay between the choice function parameters, for DR 156

8.1 Evolution of Feasibility and Cost (Upper chart) and Feasibility only (Lower chart) over the number of heuristic calls during the choice function hyperheuristic search, when applied to instance 49. The Upper chart goes from heuristic call 2550 to 4500. 201

List of Tables

4.1 Shift Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver, H = heuristic, imp = implicit. 78

4.2 Days-off Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour, d = day] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver. 78

4.3 Tour Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver, H = heuristic, imp = implicit, SGP = Stochastic Goal Programme. 79

5.1 Conceptual differences between hyperheuristics which deal with complete solutions and those which deal with partial solutions. 96

5.2 A summary of key issues when designing a hyperheuristic 109

6.1 The different problem instances. Problem parameters are given in the format $(|S|, |D|, |T|, MinPrMeet, MinMeet, MaxMeet)$ representing respectively the numbers of suppliers, delegates, timeslots, the minimum number per supplier of priority-meetings, the minimum number per supplier of meetings and the maximum number of meetings per delegate. D is the average number of requested delegates per supplier. Γ_{50} is the number of delegates being requested by at least 50% of the suppliers. $Dif = \frac{\Gamma_{50}}{D}$ is the relative difficulty of the problem. The larger the value of Dif the more difficult the problem. The suffix 'p' in each case is the corresponding measure when only priority meetings are considered. 145

6.2 Experiments with 10 low-level heuristics, real-world data- instances are ordered in increasing difficulty (from left to right) 152

6.3 Experiments with 10 low-level heuristics, random data- instances are ordered in increasing difficulty (from left to right) 152

6.4 $freq(N_j)/freqB(N_j, \rho)$ and $\kappa(N_j, \rho)$ for RP-OI, $\rho = 10$ 157

6.5 $freq(N_j)/freqB(N_j, \rho)$ and $\kappa(N_j, \rho)$ for CFb-AM, $\rho = 10$ 157

6.6 Proportion of time used on *Idle* and *Nasty* low-level heuristics. E_{Idle}, E_{Nasty} and $E_{Idle/Nasty}$ denote E when *Idle* and *Nasty* are introduced alone and when both heuristics are introduced simultaneously. In each case $CFb - AM$ is applied to instance DR, and results are averaged over 10 runs of 600 second CPU. 158

6.7 Different Choice Function expressions, real-world data 159

6.8 Different Choice Function expressions, random data 159

6.9 Experiments with 4 low-level heuristics (1, 2, 3 and 5), real-world data- instances are ordered in increasing difficulty (from left to right) 161

6.10 Experiments with 4 low-level heuristics (1, 2, 3 and 5), random data- instances are ordered in increasing difficulty (from left to right) . . . 162

6.11 Experiments with 16 low-level heuristics, real-world data- instances are ordered in increasing difficulty (from left to right) 166

6.12 Experiments with 16 low-level heuristics, random data- instances are ordered in increasing difficulty (from left to right) 166

7.1 Initial solution is *ch. ml* is a manual solution produced by the problem owner. All algorithms in the upper part start with ch as initial solution. $HH(x)$ denotes algorithm HH starting with initial solution x. $RD1$ is a hand-made RD which was tailored for this problem. csit0 results marked with * are taken from [73] which used a 1Ghz PC with 128Mb RAM. 176

7.2 heuristic calls by $CFa{-}AM$. Format: # calls during 1st 100 calls/last 100 calls to best solution, overall rank/overall proportion of call . . . 181

7.3 Comparison between CFa and $RD1$ with different initial solutions . . 184

8.1 Hyperheuristic and metaheuristic performance on the nurse scheduling problem. For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost for feasible solutions. . . . 192

8.2 Hyperheuristic and metaheuristic performance on the nurse scheduling problem (continued). For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost for feasible solutions. Note that the average is given over the 52 problem instances. 193

8.3 Choice function vs simple hyperheuristics applied to the nurse scheduling problem. For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost. 198

8.4 Choice function vs simple hyperheuristics applied to the nurse scheduling problem (continued). For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost. Note that the average is given over the 52 problem instances. 199

Chapter 1

Introduction

$Hyperheuristic = Heuristics + Learning$

Many combinatorial optimisation problems of practical interest are computationally intractable and, consequently, are solved using heuristic and metaheuristic techniques [124, 215, 218, 227, 229, 271]. This essentially means that the search spaces are so large that it is not possible to exhaustively search them. Over the past couple of decades or so, there have been significant advances in the investigation of metaheuristics both from a theoretical and practical stand point. Examples of some metaheuristic applications can be seen in [7, 19, 261, 94, 37]. Very often, metaheuristic methods make use of detailed knowledge of the problem domain in order to arrive at good solutions (e.g. [9, 89, 46, 47]). While such approaches can result in high-quality solutions, the resulting metaheuristic techniques are often not re-usable for different problems or even different instances of the same problem [48]. In effect, in order to apply the same metaheuristic technique to a different problem or problem instance, we often need to carry out (sometimes extensive) re-development and implementation of the method. This may involve the adjustment of relevant parameters of the technique for the new problem. It may also involve the incorporation of different problem-specific heuristics. For example, it was noted in [8, 9] that a tabu search metaheuristic used in [89] for a nurse scheduling problem, relied heavily on some of the problem-specific information such as the coefficients of the objective function. As a result the tabu search of [89] was no longer effective when perturbations of the problem data were made [8]. Metaheuristics can appear to be brittle, precisely because of their bespoke, *tailor-made* nature. Hence problem-specific metaheuristics are often not readily applicable to a new problem [48, 69]. Another illustrative example can be found in [46, 47] where a metaheuristic approach is used for a nurse rostering system. The system produces excellent results for nurse rostering and is in use in over 40 Belgian hospitals. The methods would need significant programming effort though to use them for nurse rostering in different countries because the system is tailor-made for the rules, regulations and working practices that apply in Belgium. If those rules, regulations and working practices change then the system will need altering. It would need even more alteration if we wanted to address rostering problems for staff other than nurses. This situation is highly appropriate in circumstances where high quality schedules that satisfy a wide range of constraints are crucial (such as the nurse rostering problem described above). The difficulty is that such bespoke systems are expensive to implement and often more expensive to adapt to new problems or problem instances. There will always be a place for expensive problem-specific systems which can produce very high quality solutions to specific critical problems. However, there is also a place for much more general systems which can deal with a wide range of problems. In many optimisation scenarios we are not looking for very high quality solutions. Instead, solutions which are 'good enough - soon enough - cheap enough' (e.g. dynamic scheduling for factory flows [74], stock cutting problems [49, 50], etc.). There is a current school of thought in the metaheuristic and search technology community which asserts that one of the main goals of the field over the nex few years is to raise the level of generality at which metaheuristics can operate [48].

The work reported in this thesis is concerned with the investigation and development of the heuristic infrastructure which will allow us to operate efficiently and effectively within this more general framework. We investigate *hyperheuristics*, a term which we have coined[1] to describe *heuristics (or metaheuristics) which choose heuristics (or metaheuristics).* A hyperheuristic can be a (high-level) heuristic which, when given a particular problem instance and a number of low-level heuristics, selects and applies an appropriate low-level heuristic at each decision point. While a metaheuristic usually (but not exclusively) deals directly with solutions, a hyperheuristic deals with *solution methods* (e.g. heuristics). A metaheuristic can and usually does modify solutions directly. A hyperheuristic can only modify solutions indirectly, by way of an operator (a low-level heuristic). This places a hyperheuristic at a higher level of abstraction and generality than most current studies of metaheuristics. Of course, hyperheuristics can be metaheuristics. Indeed they usually are [48]. For example, a genetic algorithm has been employed successfully as a hyperheuristic in [140]. The point of using the term hyperheuristics (rather than metaheuristics) is that it tells us that we are attempting to find the right method or heuristic in a particular situation rather than trying to solve a problem directly. The overall

[1]The term 'hyperheuristic' or 'hyper-heuristic' was first used by the author's PhD supervisors and subsequently by the author himself in early 2000.

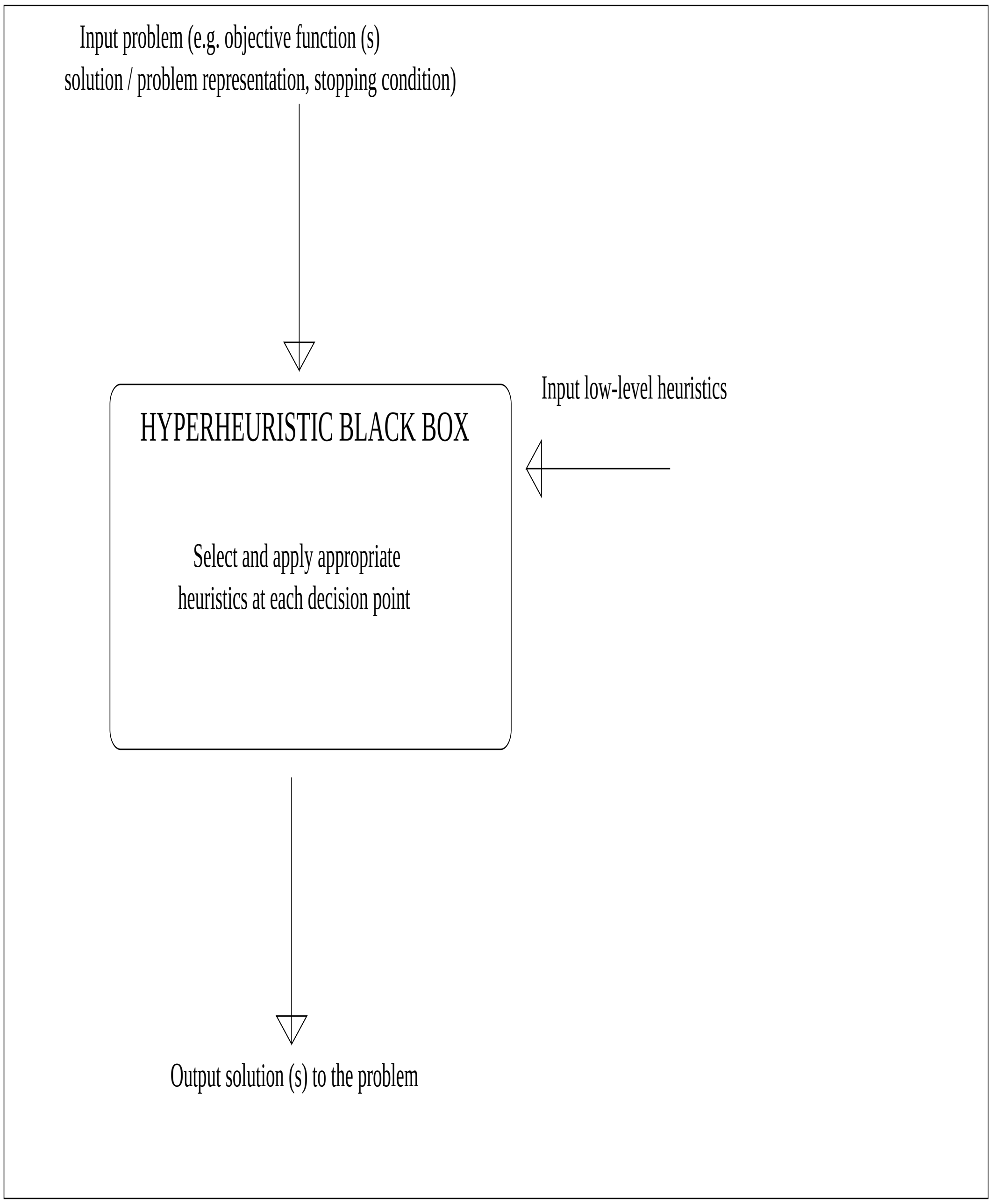

Input problem (e.g. objective function (s)
solution / problem representation, stopping condition)
HYPERHEURISTIC BLACK BOX
Select and apply appropriate
heuristics at each decision point
Input low-level heuristics
Output solution (s) to the problem

Figure 1.1: General hyperheuristic framework

goal of such research is to develop systems that can operate at a higher level of abstraction and generality than today's systems, but which can also produce solutions which are competitive with problem-specific systems. The aim is not (necessarily) to develop a method which would 'beat' existing algorithms for a given optimisation problem, but instead a method which is capable of performing *well-enough, soon-enough, cheap-enough* across a wide-range of problems and domains. Such a method could ultimately underpin cheaper optimisation systems which would be available to a wider range of users than is the situation today. A hyperheuristic can choose which low-level heuristic to apply at each decision point, until a stopping condition has been fulfilled. We are therefore not concerned with solving the problem *directly*, but simply recommending a solution method (e.g. heuristic) for the problem at hand. This is illustrated in Figure 1.1. A hyperheuristic can be thought of as being a black box which receives as input

- *a problem to be solved*: This includes a description of the problem / solution, the objective function(s), the stopping condition(s), etc.
- *a set of low-level heuristics for the problem*: A number of low-level heuristics are plugged into the black box, for the hyperheuristic to use (or choose from) at each decision point.

Then, when the stopping condition has been met, the hyperheuristic black box returns as output one or several solution(s) to the problem. Details as to how this process can be carried out constitute the core of this thesis (chapter 5).

The main achievements of this thesis is that it provides evidence that

- hyperheuristics can produce solutions of good quality, often competitive with those of modern heuristic techniques, within a short amount of implementation and development time, using only simple and easy-to-implement low-level heuristics.

- hyperheuristics are easily re-usable methods as opposed to some metaheuristic methods which tend to use extensive problem-specific information in order to arrive at good solutions.

1.1 Structure of thesis

The remainder of this thesis is structured around three parts.

In the first part, we review the literature for work related to hyperheuristics (chapter 2). We shall then give an overview of both exact and metaheuristic methods used for solving optimisation methods (chapter 3). Because all problems tackled during the work reported in this thesis involve the allocation of timeslots to resources (in this case people), we shall survey the literature of personnel scheduling for some of the most recent advances in this field (chapter 4).

In the second part of the thesis we start by discussing general design strategy issues when developing hyperheuristics. We present a description of the hyperheuristic methods developed and used in the thesis (chapter 5). We then illustrate the performance of our hyperheuristics through three different application problems taken from the real world. The first application problem (chapter 6) concerns scheduling business meetings organised by a commercial company at a sales summit. A meeting takes place between two types of individuals who are representatives of different companies, suppliers and delegates, who want to attend the summit. The problem is to allocate a number of meetings to each supplier subject to a number of constraints and with the aim of achieving a number of objectives. The second application problem (chapter 7) is about scheduling a number of students' project presentations at the University of Nottingham. A student's presentation involves not only the presenter (i.e. student), but also three members of academic staff who have to attend to assess the presentation. The problem is to allocate a timeslot to each student's project presentation. The problem has a number of constraints and objectives. The third application problem (chapter 8) is that of scheduling nurses at a UK hospital. Nurses must be assigned a work shift-pattern for their weekly duties. Again, a number of constraints and objectives must be achieved.

Finally, the third part of the thesis (chapter 9) will evaluate the overall performance of the hyperheuristics and make recommendations for future work.

1.2 Academic papers produced

As a result of the PhD research reported in this thesis the following papers have been produced:

Journal papers

1. P.Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics. *The Journal of Heuristics* 2002. Submitted.
2. G. Kendall, E. Soubeiga, and P.Cowling. Hyperheuristics: a robust optimisation method for real-world scheduling. *European Journal of Operational Research* 2003. In preparation.
3. G. Kendall, E. Soubeiga, and P.Cowling. The principles of hyperheuristics and their applications in the real world. *The Journal of Scheduling* 2002.

Submitted.

Selected Refereed Volumes

1. P.Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach for scheduling a sales summit. *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, Lecture Notes in Computer Science, pages 176–190, Konstanz, Germany, August 2000. Springer.

Fully refereed conference papers

1. P.Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. *Metaheuristic International Conference MIC'2001*, pages, 127–131, Porto, Portugal, July, 16-20 2001.

2. P.Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. *Second European Conference on Evolutionary Computing for Combinatorial Optimisation, EvoCop 2002*, Lecture Notes in Computer Science, pages 1–10, Kinsale, Ireland, April, 3-51 2001. Springer.

3. P.Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. *Parallel Problem Solving from Nature VII, PPSN 2002*, Lecture Notes in Computer Science, pages 851–860, Granada, Spain, September, 7-11 2002. Springer-Verlag.

4. G. Kendall, E. Soubeiga, and P.Cowling. Choice function and random hyperheuristics. *4th Asia-Pacific Conference on Simulated Evolution And Learning,*

SEAL 2002, Singapore, November, 18-22 2002. NTU Press. 667-671.

Chapter 2

Related work: hyperheuristics

When in doubt, generalise.

2.1 Introduction

In this chapter we survey the literature for work related to hyperheuristics. A hyperheuristic is a high-level heuristic which chooses between several low-level heuristics, using some learning mechanism. We survey the literature from the point of view of ideas using the concept of hyperheuristics. As well as reviewing the literature, this chapter serves as a unifying framework for past and future efforts in the field of hyperheuristics. Of course there may be methods which overlap between classes in our proposed framework.

The way the hyperheuristic method that is investigated in this thesis conducts the search is by combining different (low-level) heuristics. We distinguish between hyperheuristic approaches based on one low-level heuristic (special case) and those that use several low-level heuristics or neighbourhood structures. Note that the former category does not really correspond to hyperheuristics since it only uses one low-level heuristic[1]. Nonetheless we shall discuss some methods in this category which, in the author's opinion, use ideas related to hyperheuristics in many respects (e.g. use of a high-level strategy, exploration of heuristic space - as opposed to solution space). Of those methods which use several low-level heuristics we distinguish between those that contain an element of learning and those without. In general we shall describe *how* the choice of the low-level heuristics (or parameter settings for the base heuristic) is made.

2.2 Single-heuristic techniques

In this category, a base (low-level) heuristic is employed to develop a solution to the problem at hand. The use of the base heuristic, however, requires the specification of one or more parameters. Thus the problem is to find good settings for the parameter(s). This becomes a (heuristic) *parameter optimisation* problem. Hence a high-level local search method may be needed to (heuristically) search for good settings of the base heuristic parameters.

Smith [244] presented a paper in 1985 which used such an approach. The problem is that of bin-packing, that is the packing of different items of various sizes in a minimum number of bins. A base heuristic exists which tries to fill in an empty box with unpacked items taken in a certain order. The base heuristic's parameter set in this case is the order in which items are put into the boxes. The high-level local search used to optimise that parameter is a genetic algorithm (GA), whose

[1]This is analogous to saying that tabu search or simulated annealing, is a special case of population-based methods, even though the population size is 1.

chromosomes represent an ordered list of items to be placed into the boxes. Such an approach is often referred to as an indirect GA (as the GA encoding does not represent a solution but instead rules for producing a solution).

Syswerda [255] also used a similar idea to solve a problem of scheduling various resources for the U.S. Navy. The base heuristic parameter is also an ordered list of tasks to be placed into the schedule. A GA is used to search for a good ordering of the tasks. Note that when the parameter optimiser is a GA the resulting approach is sometimes referred to as a hybrid GA [103, 212](a GA is hybridised with another heuristic, in this case a base heuristic).

Kelly and Davis [156] used such a hybrid GA approach to classify new data records based on their weighted distance from the rotated members of the training set. The base heuristic is the K nearest neighbours, a statistical classification algorithm. Ling [180], too, used a hybrid GA to solve a timetabling problem.

In [243] a hybrid GA approach is also used in which the GA optimises parameters for several base heuristics as opposed to just one base heuristic. Other hybrid GA approaches include [102] for job-shop scheduling, [172, 173] for the scheduling of maintenance of electrical power transmission networks, [109] for the graph colouring problem, [228] for the bin-packing problem, [237, 238] for the line balancing problem (note that the approach is referred to as parameter optimisation), and [67] for the Methodist preaching timetabling problem (the heuristic parameters in this case represent the timeslots to be filled in).

Another paper which uses a high-level strategy to control a base heuristic is that by Burke and Newall [54]. The application problem is that of scheduling exams (also known as examination timetabling). In their adaptive framework [54], the base heuristic is a constructive heuristic which repeatedly chooses one examination and schedules it in one given timeslot if this does not result in the violation of any hard constraints. To do this, the base heuristic needs as input an ordering of the exams to be scheduled. The ordering of the examination is adaptively changed using a set of heuristic rules based on the penalty function. The method builds on the squeaky wheel optimisation method that was introduced by Joslin and Clements [153]. In squeaky wheel optimisation, a priority ordering of problem elements is given to a greedy constructive heuristic which builds a solution according to that given priority. The solution is then analysed to find areas that need improving. These trouble spots are then given higher priority and, with this new priority ordering, the constructive heuristic is re-invoked to produce a new (improved) solution. The principle of the method is that the squeaky wheel is the one that 'gets the grease' [153]. The idea of using a heuristically-driven priority ordering for a constructive heuristic can also be found in Smith [244] and Corne and Ogden [67]. The ordering in both [244] and [67] is produced by a genetic algorithm.

While the approaches described above all used a GA as a high-level parameter optimiser, Storer et al [250] developed another strand which includes other high-level local search techniques (i.e. other than a GA). The problem is still that of parameter optimisation but the inspiration came from the work by Panwalkar and Iskander [217] in which they surveyed machine scheduling rules. Panwalkar and Iskander[217] observed *'... that a combination of simple priority rules, or a combination of heuristics with a simple priority rule, works better than individual priority rules...'*. This was based on promising results published in earlier work such as that in [4] and [97]. In [4] the job-shop scheduling problem is solved using a priority rule based on various costs. In [97] the job-shop scheduling is solved using a rule based on the time available until due date. An interesting combination of priority dispatching rules in [217] is the weighted linear combination of individual rules/heuristics. The weight associated with each heuristic/rule can be regarded as a parameter. Parameterising these weights over a specific range describes an entire parameter space over which a local search method can be applied [251]. Such a parameter space was termed *heuristic space* by Storer et. al. in [250]. In the same paper the authors also proposed another search space termed *problem space* as opposed to solution space. Both heuristic and problem spaces require the use of some fast, base heuristic H at each iteration of the local search. In order to use the heuristic space the base heuristic H must be defined in terms of a parameter (or a set of parameters) which is (are) subject to some perturbation. The application of the base heuristic with different settings of the parameter(s) generates different solutions to the problem. Instead, the problem space method is based on some perturbation of the problem data to which the base heuristic is applied. Solutions of the perturbed problem are evaluated using the original data. In [250] different local search techniques based on hill-climbing were applied to both the heuristic and the problem space for the job-shop scheduling problem. The authors obtained good solutions competitive with the shifting bottleneck procedure [3]. The performance of the problem space method appeared higher than that of the heuristic space method. In [251] the job shop scheduling problem was solved using simulated annealing, genetic algorithms and tabu search applied to both the problem and heuristic space. Both the problem space and the heuristic space methods gave promising results, though the application of genetic algorithms to the problem space gave the best results of all. The heuristic space method appeared inferior to the problem space method. Both methods require parameter tuning. It is interesting to note that, within the problem and heuristic space strand, the problem space approach has enjoyed greater attention than the heuristic space, probably due to the findings in [250] and [251]. Thus we see that problem space is used in [81] for the synthesis of heterogeneous multiprocessor systems, [249] for the number partitioning problem, in [82] for datapath synthesis, in [5] for the synthesis of self-recoverable ASICs (application-specific integrated circuits), and in [175] for resource-constrained scheduling. To the author's knowledge the only paper using heuristic space, as borrowed from [250], is that by Ahmadi et al. [6] for the examination timetabling problem. It should be noted that the idea of problem space was also independently proposed by Charon and Hudry [58] under the notion of *noising method*. The idea is to add some 'noise' to the problem data, hence the term of perturbation. Other examples include [63, 248, 134].

2.3 Multiple heuristic or neighbourhood search techniques

While the previous category only discussed approaches that delat with a single heuristic, there have been methods which have dealt with two or more heuristics. These methods are discussed here. We distinguish between approaches equipped with some learning mechanism and those without.

2.3.1 Without learning

In this group of approaches several heuristics or neighbourhood structures are available which may be used at each decision point. Methods in this category are also special cases in that they do not provide any learning mechanism.

In some approaches, the choice of the heuristic is limited to one possibility. This means that the available heuristics are applied in a certain pre-determined order. For example one such approach is due to Mladenović and Hansen [194] who introduced the concept of Variable Neighbourhood Search (VNS). The idea in VNS is to use an ordered sequence of solution neighbourhoods. The search starts from the first neighbourhood and moves to the next in the sequence. The search is intensified by applying a local search to a solution generated at random in the nearest neighbourhood, and diversified by moving to the next neighbourhood in the sequence. Different local search methods can be used within VNS including Variable Neighbourhood Descent (VND). In VND a descent method is applied to the current neighbourhood. If this does not yield a better solution, descent is applied to the next neighbourhood in the sequence. VNS highlights crucial issues including, which neighbourhoods to use, in which order should they be applied, and which strategy should be used in changing neighbourhoods. The answer to these questions is problem-specific. VNS has been applied to many combinatorial optimisation problems. A recent survey of the method and its applications can be found in [137].

A similar approach to VNS is that employed by Dowsland [89] in which a three-phase tabu search is used to solve a problem of nurse scheduling. In the first two phases, several neighbourhoods and heuristics are employed in a certain order. When it is not possible to find a feasible (non-tabu) move within a given neighbourhood, the next neighbourhood is considered. We shall come back to this work [89] later (chapter 8).

In other approaches the choice of the heuristic is more flexible than the ones above and depends on the qualities of the current solution. These characteristics dictate the range of heuristics that may be applied to the solution in order to 'repair' some undesirable features. Such approaches are often referred to as 'iterative repair'. From this point of view one could say that these approaches involve some sort of semi learning in order to decide which (type of) heuristic to choose. For example

Zweben et al [279] used an iterative repair method in the context of scheduling. Iterative repair methods usually start from a complete but possibly flawed solution which is iteratively repaired in this case due to real-time events. This is in contrast to a constructive approach which incrementally constructs a complete solution using a partial one. Perhaps the oldest iterative repair method is the famous Kernighan-Lin [160] which is mentioned in the next paragraph. Iterative repair methods have also been used in [193, 247, 80, 225, 224, 60].

In [267], Vaessens et al. proposed a local search template in an attempt to help classify local search methods. Using their local search classification a hyperheuristic can be regarded as a *multi-level* local search method. In effect, a hyperheuristic makes use of several neighbourhoods and heuristics, which correspond to several levels of search. The hyperheuristics presented in this thesis use a single solution, as opposed to a population of solutions, and are therefore *point-based* local search methods. Vaessens et al. noted that local search methods *'...which use more levels in the local search template seem to be powerful and deserve wider attention...'*. This further provides some motivation for the work of this thesis. The work of Martin et al [186] is an example of multi-level local search method in which the TSP is solved using two neighbourhoods: At level 1, simulated annealing is used with 4-opt (disconnecting and reconnecting 4 vertices). At level 2, 3-opt is used to determine a local minimum which is then compared (in a simulated-annealing fashion) to the level 1 solution. The same method is reported in [187] and in [185] where it is applied to both the TSP and the Graph Partitioning Problem (GPP). It should be noted that Lin-Kernighan's TSP method [179] is a multi-level approach in which a restricted range of k-opt moves is used with a variable k. The idea of varying k was initially proposed by Kernighan and Lin in [160] for the GPP. The idea has been used in its original form or modified versions to solve various TSP problems.

Lin-Kernighan [179] is regarded as one of the most efficient heuristics for TSP.

2.3.2 With learning

In this category we include methods which learn to choose between several low-level heuristics. The main components in such approaches are the existence of several low-level heuristics, the existence of a high-level heuristic which uses some learning mechanism in order to choose between the various existing low-level heuristics. Hyperheuristics could be classified on the basis of the learning mechanism employed. We shall distinguish between hyperheuristics which use a GA during the learning process from those which use other mechanisms. This is because a great deal of work on hyperheuristics has been done using genetic algorithms.

Genetic Algorithm based hyperheuristics

In GA-based hyperheuristics the idea is to use the GA to evolve the solution methods, not the solutions themselves. Typically the chromosome represents a list of heuristics or rules that may be applied to the current solution. Such GA-based hyperheuristics are often called an indirect GA. Kitano [162] was an early paper (1990) which used a GA-based hyperheuristic. The problem is that of neural network design. Traditionally, in GA-based neural network designs, the chromosome directly encodes network configurations. Instead, Kitano used an indirect encoding, called *grammar encoding method*, which encodes a set of rules that generate networks. Thus instead of directly evolving networks (the traditional approach), Kitano evolved rules which generate networks. Kitano's approach proved much superior to the traditional approach.

Four years later, Fang et al. [103] and Norenkov [213] independently used such an approach. In [103] the GA-based hyperheuristic is applied to the open-shop scheduling problem. The chromosome represents a set of heuristics associated with the jobs to be scheduled. Thus the GA evolves heuristic choice instead of evolving actual schedules. The approach proved superior to the direct representation. In [213] the GA-based hyperheuristic is applied to a problem of design of computer systems. As above the chromosome encodes different heuristic methods to solve the problem, instead of the encoding the schedule.

Dorndorf and Pesch [87] also used a GA-based hyperheuristic to solve the job-shop scheduling problem. Though their approach did not produce solutions better than those of other approaches, it was observed that the GA-based hyperheuristic was more robust to problem changes as well as easy to implement.

GA-based hyperheuristics are also used in [258] in the context of printed circuit board assembly and in [212] in the context of scheduling. More recently, Terashima-Marin et al. [259] used such an approach to solve the examination timetabling problem. The GA chromosome of [259] represents a combination of various strategies, heuristics and conditions for switching between strategies for the examination timetabling problem. Results obtained by the hyperheuristic were overall better than those obtained using a graph-colouring based method.

Cowling et al. used a similar approach in [68] to solve a trainer scheduling problem [2]. The GA chromosome represents an ordered sequence of low-level heuristics to be applied to the problem. Results obtained by their hyperheuristic (Hyper-GA) are superior to those obtained by both a GA using a direct representation of the problem and a memetic algorithm which invokes a local search improvement procedure after

[2]Note that their paper refers to one of the papers published in this thesis.

the application of crossover and mutation operators. The superiority of the hyperheuristic approach of [68] was attributed to the ability of the hyperheuristic to find ways of combining those low-level heuristics which are effective across all instances considered. In [135] the trainer scheduling problem is solved using an enhanced version of the hyper-GA of [68] which uses an adaptive length chromosome.

Hart and Ross [138] used a GA-based hyperheuristic to solve the job-shop scheduling problem. The GA chromosome represents which method to use to identify conflicts among schedulable operations and which heuristic to use to select an operation from the conflicting set. Computational results showed that evolving solution methods was beneficial. Results obtained were promising when compared to those most recently published.

Hart et al. [139, 140] also used a GA-based hyperheuristic to solve a problem of chicken catching. The problem is decomposed into two sub-problems with each sub-problem solved using a GA. The GA chromosome in the first phase represents various heuristic rules used to split an order (for chickens) and to assign a load (of chickens) to a chicken catching squad. The result was a robust scheduling system which was capable of producing practical schedules for the factory.

Smith [245] also used a hyperheuristic approach in a memetic algorithm (MA) framework. As will be mentioned in chapter 3, memetic algorithms are genetic algorithms within which some sort of local search is applied to further improve on the solution (i.e. individual) after crossover and / or mutation have been applied. In his paper, Smith addressed the issue as to which local search method to apply, in how many iterations and in which fashion (i.e. whether single call or descent). To do this, the chromosome representation of the solution includes an encoding of which local search method to choose. Hence in Smith's approach, solutions and solution methods co-evolve simultaneously. The resulting MA-hyperheuristic approach was able to outperform both a conventional GA (i.e. no local search) and a static MA (i.e. the memetic algorithm in which the learning of which heuristic to apply has been disabled) when applied to a classic 64-bit problem in which 16 fully deceptive 4-bit sub-problems were considered (fully deceptive 4-bit problems - in which the aim is to set all 4 bit positions to '1' - are often used to analyse the performance of genetic algorithms; these problems are designed in such a way as to keep local optima far away from the global optimum [17]).

It should be noted that GA-based hyperheuristics need tuning of the genetic algorithm parameters (population size, selection scheme, crossover and mutation probability etc).

Non-GA based hyperheuristics

In 1961 Fisher and Thompson [107] published a paper on hyperheuristic methods. Fisher and Thompson [108] also published their work in 1963 at the same time as Crowston et al. [75]. In all three papers [107, 108, 75] the problem was that of job-shop scheduling. The learning method was based on some probabilistic weighting of the low-level heuristics, which represented various scheduling rules. In [108] the hyperheuristic combined two scheduling heuristics (rules) in a probabilistic learning algorithm. The hyperheuristic learning mechanism was in line with reinforcement learning ideas of reward-punishment [254, 154] in which the probability of choosing a heuristic was increased when the heuristic yielded an improvement and decreased otherwise. The hyperheuristic is compared to an *unbiased random process* which chooses either heuristic at random. The hyperheuristic proved to be superior to the unbiased random process.

The work by O'Grady and Harrison [214] is an extension of the previous idea in which a similar approach is applied to the job shop scheduling problem. The authors proposed a general framework which includes a large number of job-shop scheduling heuristics (rules).

Ross et al. [234] used a hyperheuristic approach to solve the bin-packing problem. The hyperheuristic's learning process used the learning classifier system XCS [276]. XCS is an extension to the learning classifier system ZCS [275]. A classifier represents a heuristic or rule for packing a bin. Here the idea is to use a classifier system in order to produce a good combination of several bin-packing heuristics and rules. The resulting system produced good solutions, better than those produced by individual heuristics. The system was also able to generalise well (when making modifications to the original problem). Naturally a series of parameters inherent to classifier systems needs tuning (rate of exploration/exploitation, GA parameters, percentage of training data, etc.).

Nareyek [211, 210] proposed a hyperheuristic approach which uses ideas based on reinforcement learning [154] in order to choose which heuristic to apply next. Each heuristic has an associated weight which can increase or decrease according to its performance. Various reward and punishment schemes (weight adaptation) are used when selecting a low-level heuristic. The hyperheuristic was applied to two optimisation problems (Orc Quest problem and the Logistics Domain) in order to compare the various weight adaptation schemes and good results were obtained in each case.

Allen and Minton [12] proposed a hyperheuristic method in the context of Constraint Satisfaction Problems. The learning mechanism was based on runtime performance predictors such as the estimated number of constraint checks. The rationale was that the heuristic with the fewest constraint checks is faster. The hyperheuristic chose the heuristic with the lowest estimated average number of constraint checks. The method proved to be better than interleaving the low-level heuristic. A similar work, developed by Lagoudakis and Littman in [170], used reinforcement learning. The learning mechanism was based on a Markov decision process. The hyperheuristic used two low-level heuristics.

Another strand of hyperheuristics was developed by Mockus et al. [195] in 1989. Their approach, named *Bayesian heuristic approach*, used the Bayesian approach [198][3] in an attempt to improve on a given (set of) heuristic(s) by randomising and optimising their parameters. We describe the Bayesian heuristic approach for a set of heuristics (note that in the case of a single heuristic, the problem becomes a parameter optimisation problem as described in section 2.2). Given a set of heuristics, each with a certain (unknown) probability of being chosen (by the hyperheuristic), the Bayesian heuristic approach determines the probability distribution of the heuristics, that is, the probability with which each heuristic should be called based on some historical performance of the heuristics. The probability distribution is chosen in such a way as to minimise the expected deviation from a global optimum (average-case analysis). The method has been applied to a variety of discrete optimisation problems. See [198, 203, 199, 195, 196, 197, 200, 202, 201] for further details. Throughout chapters 6, 7 and 8, we shall carry out different analyses of heuristic call frequencies, which is closely related to the probability distribution of the heuristics in the Bayesian heuristic approach.

A hyperheuristic approach using case-based reasoning (CBR) techniques as a learning mechanism is proposed in [52]. The case-based reasoning hyperheuristic

[3]In the Bayesian approach one is interested in the average-case analysis of an algorithm, as opposed to an exact method where the worst-case analysis is considered.

is applied to the examination timetabling problem. In the case-based reasoning paradigm, a set of previously solved problem instances are stored in the case base along with partial solutions obtained at different steps of the search. In order to use the CBR system as a hyperheuristic, corresponding solution methods are also included in the case base alongside these partial solutions. When a new problem instance is to be solved the most similar case in the case base is retrieved and the corresponding heuristic which gave the corresponding partial solution is chosen. Hence, at each decision step, a low-level heuristic is chosen based on the similarity between the current (partial) solution and the partial solutions stored as cases in the case base. The case-based reasoning approach was able to outperform individual heuristics used in the case base. Note that [52] references one of our own papers.

Finally some of the hyperheuristics developed in this thesis use a *choice function* to learn which heuristic to choose. We call these hyperheuristics choice function hyperheuristics (see chapter 1) [69, 70, 73, 72, 157, 71, 159, 158]. We will discuss these later in Chapters 5 to 9.

It should be noted that all hyperheuristic methods presented in subsection (2.3.2) allow for switching between heuristics during the solution process. The choice of the low-level heuristics is dynamic and adaptive.

2.3.3 Adaptive problem solving in AI systems

Some methods used in AI (planning and scheduling) systems are similar to hyperheuristics that learn to choose between several low-level heuristics. Here we give two examples of such approaches before elaborating on the differences between these methods and hyperheuristics.

Gratch and Chien [132] proposed a technique for adaptively modifying a given method for solving a given problem (problem solver). The problem is as follows. Given a parameterised base problem solver for solving a given problem, learn to adapt (optimise) the parameters of the problem solver over a certain number of training problem instances, so that the thus-optimised (or adapted) problem-solver will perform well enough on other problem instances. The base problem solver is a broad general procedure for solving the problem. Its parameters can be viewed as corresponding to a range of strategies or heuristics that can be used. Therefore optimising the values of the parameters corresponds to choosing between several strategies. Each strategy has an associated utility value which reflects the expected worth of the corresponding strategy or heuristic. The goal of their search problem, called *adaptive problem solving* is: given a distribution of problem instances find some values of the parameters (which corresponds to finding a combination of different strategies or heuristics) that maximises the overall predicted performance of the problem solver (in terms of solution time) over the distribution of the problem instances. This approach is similar to the heuristic space approach except that here, parameter settings are only allowed to change from one problem instance to another. They are kept constant during the search for a solution to the problem instance.

Fink [106] also proposed a technique for choosing among problem-solving methods. The problem (for which he proposed his technique) is as follows. Given the past performance of a particular method for solving a particular problem (problem-solver) expressed in terms of number of problem instances solved with success, with failure and unsolved within a given time bound, how well will the same method solve a new instance of the problem within a certain time bound? Fink used a statistical approach to solve this problem. His technique is aimed at selecting between several problem solvers and a time bound to apply the selected method before solving a new problem. In his technique a database of problem solver performance is maintained. Each time a new instance is to be solved we must choose which problem solver to apply to the new problem and for how long (time bound). The way a problem solver is selected is similar to roulette wheel [126] with the probability of choosing a (statistically) good problem solver being higher than that of choosing a poor problem solver. This approach is similar to the multiple-heuristic hyperheuristics described above except that here, again the switch from one low-level problem solver to another is only allowed from one problem instance to another. Once chosen the selected problem solver remains constant during the search for a solution to the problem instance being solved. The following passage in Fink's paper [106] reads *'... We do not provide a means for switching a method or revising the selected bound during the search for a solution. Developing such a means is an important open problem...'*. This provides a further motivation for the work of this thesis.

Based on the above two papers a clear distinction must be made between problem-oriented hyperheuristics, the ones described in [132] and [106] for example, and solution-oriented hyperheuristics (any other hyperheuristic in this thesis). Problem-oriented hyperheuristics choose between problem solvers. When selected, the chosen problem solver is applied to the entire problem instance and its performance will not be assessed until (at the earliest) the problem instance has been solved. Instead, solution-oriented hyperheuristics choose between heuristics or neighbourhood structures. Solution-oriented hyperheuristics choose and may apply several different low-level heuristics for a single problem instance. While problem-oriented hyperheuristics are concerned with the selection of an appropriate problem-solver in order to solve an entire instance of the problem at hand, solution-oriented hyperheuristics are concerned with the selection of an appropriate operator (e.g. heuristic) which can modify the current solution of a given instance of the problem. The main difference between the two hyperheuristics is that in the former hyperheuristic, once selected the problem-solver is applied to the problem instance throughout the hyperheuristic execution. Whereas the latter allows for switching between different operators during the execution of the hyperheuristic. The passage in Fink's paper [106] presented in the previous paragraph further illustrates the difference between these two types of hyperheuristics. Problem-oriented hyperheuristics require training on a number of problem instances in order to be tuned to the class of problem instances to be solved. On the contrary, solution-oriented hyperheuristics may or may not require training as will be explained in chapter 5 which addresses design issues for hyperheuristics. This thesis is concerned with solution-oriented hyperheuristics. From this point on we shall use the term hyperheuristic to refer to solution-oriented hyperheuristics. While problem-oriented hyperheuristics work at the macro level (i.e. choosing a problem solver in order to solve one entire problem instance), solution-oriented hyperheuristics work at both the macro and the micro levels (i.e. choosing between different heuristics *during* the solution process for a given problem instance). Of course, both types of hyperheuristics operate in the *space of heuristics*. Whereas most applications of metaheuristics, which have control over the way the low-level heuristic modifies the solution, operate in the *space of solutions*.

2.4 Conclusion

In this chapter, we have surveyed the literature on hyperheuristics and other related efforts. As well as being a survey, this chapter provided a unifying framework for past and future work in the field. Our classification is based on whether the high-level heuristic uses one or more low-level heuristics, and, in the latter case whether it is equipped with some learning mechanism which helps choose the low-level heuristics. In this case we have a hyperheuristic, that is a high-level heuristic which chooses between several low-level heuristics using some learning mechanism which is responsible for its adaptiveness, which results in its robustness.

Since the first hyperheuristic publication that we know of in 1961 [107], there has been an increased interest in the development of generic or semi-generic methods aiming at adaptively choosing between several heuristics or possibilities. We note that little work was done during the following three decades (from the 1960's till the 1990's). The work by Kitano [162] in 1990 was the beginning of modern hyperheuristic research. Thus, in the last decade there has been an increased interest in this area of research. We think that is due to the increasing complexity of problems tackled by operations research and artificial intelligence researchers along with the frequency with which these problems occurred and re-occurred, especially the class of scheduling problems. Hence the need for re-usable software programs has become apparent, especially in industrial applications [48]. On the other hand the proliferation of local search methods [216] meant that more and more heuristics or possibilities were available when solving a given problem. This resulted in a twofold situation: develop a generic (as in re-usable) method which is able to appropriately[4] choose between several alternatives for a given problem (or class of problems).

One of the main goals of using hyperheuristics is the aim of achieving robustness in terms of *good-enough, cheap-enough, soon-enough* solutions across a wide range of problems and domains. This would help overcome the difficulty posed by the use of bespoke *tailor-made* metaheuristics which often are not readily re-usable for other applications. However it would be difficult to achieve this quality without a certain level of learning ability, which might be needed in order for the hyperheuristic to

[4]note that an appropriate choice implies one which is adaptive to the change of some features of the problem

solve different problems in an effective manner. Learning is a crucial element in the ability of the hyperheuristic to cope with various regions of the search space, various problem instances and various problem domains. Ideas from machine learning in general and reinforcement learning in particular may be borrowed [246, 57, 154, 254, 61]. Conversely, hyperheuristics can be used in the field of machine learning. Collaboration between researchers from both communities should be welcomed.

It is already contended that hyperheuristic development is going to play a major role in search technology over the next few years [48]. It seems that the potential for scientific progress in the development of more general optimisation systems such as hyperheuristics, for a wide range of application areas, is significant [48]. Hyperheuristics appear worthy of further investigations.

Chapter 3

Related work: exact & metaheuristic methods

3.1 Introduction

In the previous chapter, we gave an overview of hyperheuristics and other related methods. Here we give an overview of alternative methods for solving optimisation problems. It is not intended to be an exhaustive survey of the field. For more detailed treatments, see [124, 215, 218, 227, 229, 271]. We distinguish between exact methods, aimed at finding optimal solutions, and heuristic methods aimed at finding reasonable (but not necessarily optimal) solutions. Some of the methods discussed in this chapter are investigated in research papers described in the next chapter on personnel scheduling. Although in this thesis we are concerned with heuristic optimisation methods, we shall briefly discuss exact optimisation methods. We shall then address some of the most widely used heuristic approaches to optimisation problems. We conclude by highlighting how metaheuristics can be employed in a hyperheuristic framework.

3.2 Exact methods

Many optimisation methods use mathematical programming techniques. Here we give a short description of some of the most frequently used. See [242, 278] for a more detailed treatment.

Lagrange multipliers This approach is used for optimisation problems involving possibly non-linear constraints and objectives. The basic idea is to transform such a constrained optimisation problem into an unconstrained one using the so-called Lagrange function and then solve the resulting unconstrained optimisation problem using classical optimisation methods (e.g. employing derivatives). In this method one generally relaxes some, but not all, constraints. The method is often used in conjunction with linear programming (next paragraph). [242, 278, 30] give further details.

Linear programming This is perhaps the most popular technique used in mathematical programming, due to the fact that many important applications can be modelled as linear programming problems. Also, concepts and insights derived from linear programming constitute the basis for much of the general theory of mathematical programming. Both the constraints and the objectives are linear. The simplex method is the first method developed to solve linear programming problems. The simplex method is an exact local search which works by repeatedly moving from one (basic) feasible solution to an adjacent until the optimal solution is found. The method was introduced by Dantzig in 1947 [76, 78].

Cutting plane The simplex (and other linear programming method) becomes inefficient when variables of the linear programming problem are required to be integer. One of the most well-known methods for mixed integer programming is the cutting plane method. The relaxation of the integer linear programming problem is solved to optimality and further constraints are added to the simplex tableau in order to 'cut off' the non-integer parts of the solution. The method was developed by Gomory [127, 128, 129, 130].

Explicit enumeration For pure integer 0-1 problems an alternative method is to enumerate all possibilities of 0-1 values assigned to the variables. However, because of the large number of such possibilities, it is useful to apply suitable exclusion rules in order to keep the number of possibilities as small as possible (smart enumeration). This method can also handle non-linear constraints. See [242, 278] for further details.

Branch and bound A good practical approach when dealing with a mixed integer linear programme (that is, not all variables must be integer) is branch and bound [59]. Like the explicit enumeration method, branch and bound starts with a solution to the relaxed linear programme and, for each integer variable of the mixed linear programme which is not integer in the optimal solution of the relaxed problem, each of the two options (branches) of rounding that value up or down is added as a constraint to a further linear programme and the resulting solution evaluated (bound). This helps reduce the number of options to pursue, hence the term branch and bound. Branch and bound methods are also applicable to pure integer linear programmes. Although branch-and-bound is used with linear programmes, there are other examples such as the assignment problems or the travelling salesman problem where branch-and-bound is used to obtain bounds.

Dynamic programming The mathematical programming techniques described above are usually applied to static problems where the passage of time is ignored. Some problems, however, involve sequences of decisions to be made dynamically over time and dynamic programming techniques can be efficient for such problems. A number of important discrete dynamic programming models can be described using networks (vertices and edges) for which various graph-theoretic algorithms have been developed (e.g shortest route problems). In dynamic programming it is assumed that if all decisions to date are optimal, and we look over all possible decisions for the current state, then the best one will also be optimal. Note that dynamic programming need not refer to time (e.g. knapsack problems). [28, 29] are recent books on dynamic programming.

3.3 Heuristic methods

Exact methods become impractical when applied to NP-hard problems (these problems are characterised by the fact that the computing time required to solve such problems to optimality increases exponentially with the size of the data). In such a context one resorts to the use of heuristics. A heuristic uses domain knowledge to solve a given problem. Unlike exact methods, heuristics do not guarantee optimality of the solution. However, heuristics are fast methods which could deliver good solutions. The last 20 years or so have seen the development of a number of heuristic methods for combinatorial optimisation known as *metaheuristics* or modern heuristics. Here, we give an overview of some of the most widely known metaheuristic methods often applied to combinatorial optimisation problems. One of the most basic local search methods is *hill climbing* or *iterative improvement* which repeatedly moves to a solution better than the current one until it finds a local optimum (i.e. a solution which is better than all others in its neighbourhood). Because only improving moves are accepted, hill climbing tends to get stuck fairly quickly in a local optimum, which may be much worse than the global optimum. To overcome this, modern heuristics (or metaheuristics) are equipped with some way of escaping local optima. The idea is to accept a solution even if it is worse than the current one in order to find better solutions later on in the search process. The main features of metaheuristics are intensification (thorough investigation of promising regions of the search space, which might lead to a short term improvement of the current solution) and diversification (exploration of parts of the search search not yet covered, which might require a short term worsening of the current solution). These features are complementary and are therefore required for an effective metaheuristic search. The effectiveness of a metaheuristic will depend on how these two features are balanced. We distinguish between point-based (or trajectory) metaheuristics and population-based ones. Whereas in the former category only one solution is maintained at a time in the latter a population of solutions are maintained. We present below an overview of some of the most popular metaheuristic methods. A special emphasis on Tabu Search, Simulated Annealing and Genetic algorithms is given as, in practice, these are the three most widely utilised approaches for combinatorial optimisation.

3.3.1 Point-based methods

Tabu Search Tabu search is an adaptive memory based technique originally proposed in 1977 by Glover [118] though it took over a decade for the method to become popular. In order to implement a tabu search algorithm for a given problem, one must define both a search space and a neighbourhood structure. Tabu search works by controlling the way solutions are iteratively changed within the local search framework. When performing local search (i.e. iterative improvement) there is the possibility of getting stuck in local optima. Tabu search tries to prevent this by accepting even non-improving moves. However this may lead to cycling back to previously visited solutions. To avoid this, a distinctive feature of the method is to maintain a short-term memory structure which disallows those most recently applied moves (hence the term 'tabu'). Such moves are made 'tabu' for a number of iterations. There is a variety of methods as to how restrictive the tabu list can be. Another parameter defines the length of the tabu list. That is, how many move attributes should be 'remembered' at a time. It is also possible to use several tabu lists, for example one for each type of neighbourhood move [89]. Although tabu list are important in tabu search, they may prohibit attractive moves, even if there is no risk of cycling. It is therefore often essential to overrule the tabu status of certain promising moves. These are known as the 'aspiration criteria'. The most common aspiration criterion is to accept a move, even if it is tabu, provided that that move results in a better solution than the best found so far. Tabu search has enjoyed a great deal of research attention over the past 15 years or so. Many people believe the Lin-Kernighan heuristic for the TSP [179] (1971) is a tabu search approach, which predates Glover's 1977 paper [118]. The seminal papers for tabu search are often considered to be [119], [120], [121] and [136]. Also an important book by Glover and Laguna [125] is often cited as the key reference for tabu search users. Further issues can be considered in an implementation of a tabu search algorithm. For example in addition to short-term memory issues, there are issues related to recency memory (intensification) and frequency memory (diversification). In sophisticated tabu search implementations it is also possible to allow for infeasible solutions and surrogate objectives. Indeed tabu search is often hybridised with other metaheuristics [109, 219].

Simulated Annealing Simulated annealing was first introduced as a search strategy by Kirkpatrick et al. in 1983 [161]. The idea originated from the physical annealing process of metals published by Metropolis in 1953 [189]. As in tabu search, in order to implement a simulated annealing algorithm for a given problem, one must define both a search space and a neighbourhood structure. The idea of simulated annealing search is to always accept the solution under consideration if it is better than the current solution (intensification), otherwise a worse solution may be accepted with a certain probability (diversification). Over the course of the search the probability of accepting a worse solution gradually decreases. This ensures that as the search progresses, the algorithm focuses on areas of the search space which are likely to contain good (local or global) optima. The nice thing about simulated annealing is that it is easy to implement. An even more important advantage of the method is that there are known theoretical convergence results. Indeed, if the algorithm is allowed to run for sufficiently long iterations, it is guaranteed to find the (a) global optimum. The key in a simulated annealing implementation is how the temperature is decreased during the search (the 'cooling schedule'). There are two types of cooling schedules. Static schedules which must be predefined (i.e. before the algorithm is run), and dynamic cooling schedules, which are essentially adaptive (based on information obtained while the algorithm is being run). A criticism of simulated annealing is that it is completely memoryless (the algorithm disregards historical information gathered during the search). For this reason, simulated annealing is often discarded in favour of tabu search. However, there are no proofs of convergence in the literature for tabu search. A number of algorithms similar to simulated annealing have been reported in the literature. This includes Threshold Accepting which deterministically accept solutions only if they are better than a given threshold [92, 206]. The Noising method of Charon and Hudry [58] can also be regarded as a simulated annealing related algorithm where an objective function, to which some 'noise' is added, plays the same role as the simulated annealing's temperature. Like the temperature in simulated annealing, the noise is gradually reduced so as to end up with the problem's original objective function. Simulated annealing has been applied to a wide variety of problems in operations research. [163] contains a review of such applications. [2] contains excellent chapters discussing theoretical issues surrounding simulated annealing. See [93, 101] for some of the most recent issues related to simulated annealing.

Greedy Randomised Adaptive Search Procedures a Greedy Randomised Adaptive Search Procedure (GRASP) is a multi-start metaheuristic technique for combinatorial optimisation problems, in which each iteration consists of a constructive phase followed by an improvement phase. In the constructive phase, a constructive algorithm is used to create a solution from scratch. The constructive heuristic can be applied with different random seeds to create different starting solutions. The constructive heuristic is a greedy approach which tends to select the best candidate elements taken from a restricted candidate list in order to build a complete solution. It is also adaptive in that it updates the evaluation of remaining candidate elements each time a candidate element has been selected. The solutions thus obtained by the constructive heuristic are not necessarily optimal, even with respect to simple neighbourhoods. The second phase of GRASP is therefore invoked in order to improve on the starting solution produced at the end of the first phase. This second phase uses a local search approach. It is therefore necessary to define a neighbourhood structure and a search space. Usually the second phase utilises simple neighbourhood structures. Overall the GRASP algorithm will repeatedly apply Phase 1 and Phase 2 using different seeds at phase 1 (so as to produce different starting solutions). The best solutions found are reported at the end of the algorithm. A major advantage of GRASP algorithms is their ease of implementation. Indeed the method requires few parameters and the neighbourhood structures used are quite simple. GRASP is often hybridised with path relinking techniques which were initially developed for intensification purposes in tabu search. Recent accounts of the methods can be found in [104, 222]. [105] contains a recent survey of GRASP.

Iterated Local Search Iterated Local Search is another point-based search method in which a local search method is repeatedly applied to different solutions (similar to GRASP). Unlike GRASP, the essential idea in Iterated Local Search lies in focusing the search not on the whole solution space but on a smaller subset of the solution space which contains solutions that are locally optimal with regards to the local search method. The key in Iterated Local Search is in the sampling of the reduced set of local optima. Iterated Local Search starts with an initial solution to which the local search method is applied. Then, from that local optimum, Iterated Local Search will repeatedly perform a perturbation of the local optimum and apply local search to that perturbed local optimum and so forth. The local search method may be viewed as a black box which is repeatedly applied to perturbed local optima. When a local optimum to the perturbed local optimum is obtained, an acceptance criterion is applied to decide which of the new and previous local optimum is accepted. How effective Iterated Local search is will depend on the local search method, the perturbations carried out and the acceptance criteria. Like GRASP, Iterated Local Search is fairly easy to implement. Of course the method is memoryless (but memory can be incorporated (e.g. [252]). [182] gives a thorough investigation of Iterated Local Search methods.

Guided Local Search Guided Local Search was developed by Tsang and Voudouris [272, 266]. The basic principle in GLS is to change the objective function value when a local optimum has been reached so that other areas of the search space can be explored. GLS has been used in [266, 272, 273].

Variable Neighbourhood Search Variable neighbourhood search has already been discussed in the previous chapter (section 2.3.1).

Other perturbation methods This category includes metaheuristics which uses perturbation in order to escape local optima. The perturbation can be implemented at various levels: problem data perturbation, heuristic perturbation, all of which have been discussed in the previous chapter (section 2.2). Note that GLS can be regarded as a perturbation method in that the objective function is perturbed when one reaches a local optimum.

3.3.2 Population-based methods

Unlike point-based methods, the methods described in this subsection maintain a population of solutions.

Genetic Algorithms Genetic algorithms were first introduced in the late 1950's [115, 116, 35] though John Holland is often mentioned for carrying out much of the seminal work on GA's [144] (later re-edited in [145]). Genetic algorithms mimic the evolution of biological species in nature. A population of strings is used, which is often referred to as chromosomes. Strings can be recombined by way of genetic crossover and mutation operators. The genetic algorithm search is guided using the objective function for each individual (string) in the population. Individuals with higher fitness (i.e. strings which represent better solutions) are given more opportunity to breed. Individuals are thus interbred according to the principle of *survival of the fittest.* This principle encourages the creation of better individuals as the algorithm progresses from one generation to the next. In a general genetic algorithm framework, an initial population of solutions is chosen. The algorithm then repeatedly applies crossover and mutation to selected individuals and evaluates the fitness of the offspring(s). The algorithm then selects a new population based on the old population and the offsprings obtained. In order to implement a genetic algorithm one must define a number of parameters. This includes how the initial population is obtained (usually at random) and how many solutions are considered (population size). When should crossover operations be performed? Also what types of crossover should be considered (e.g. 1-point, 2-point, crossover, etc.)? When should mutation operator be applied? What types of mutation operators should be applied? How are individuals selected for crossover and mutation? How is the new population selected (strict selection based on ranking or tournament selection)? There are various stopping conditions including a preset number of generations, CPU time, stop after the population diversity falls below a certain threshold. Even the way solutions are represented needs to be clearly defined. One of the most popular representations is binary (0-1 alphabet) but there are other possibilities. Two of the most recent books on GA's often encountered in the literature are those by Davis [79] and Goldberg [126]. A more recent account of genetic algorithms can be found in [230].

Scatter Search and Path Relinking Scatter search is a search method which also maintains a population of solutions at a time. The key in scatter search is to construct solutions by combining others. Scatter search starts with a set of points, called reference points, which are good solutions previously obtained using some other search method (e.g. tabu search, iterative search). Here, 'good' solution does not just mean with respect to the objective function, but it might also refer to the diversity of the population of solutions considered (solutions are 'scattered' all over the search space in order to ensure maximum diversity). The scatter search algorithm systematically produces combinations of the reference points in order to create new points. Scatter search consists of five steps also known as methods. In the first step, a set of solutions is produced which contains solutions as diverse as possible. This step is known as 'Diversification generation method'. Then a second phase, the 'Improvement method', is applied which tries to improve on the quality of the solutions taken from the previous phase. A 'Reference Set Update method' is invoked next. This consists of selecting the b best solutions following the Improvement phase. It is these solutions that are used to generate smaller subsets of solution for recombination. This is known as the 'Subset Generation Method'. Once the subsets have been formed, a 'Solution Combination method' is applied to solutions in each subset in order to produce new combinations of these solutions. The solution combination method in scatter search is similar to the crossover operator in genetic algorithms. A similar approach to scatter search is 'path relinking'. In path relinking, one tries to link up two or more solutions. For example if solution B was obtained from solution A after a number of moves, it might be possible to observe the moves which led to B when starting from A. This path of moves from A to B, once identified can help observe certain features in the involved moves. These features can be recombined in order to help discover new moves which might lead to even better solutions or to solutions that are not better, but from which it might be possible to explore new areas of the search space. Both scatter search and path relinking are used in conjunction with an existing search method. Application examples can be found in [171, 232, 122, 123].

Ant Colony Optimisation (ACO) Ant Colony Optimisation is another population-based technique for optimisation problems. The method was developed in analogy to the biological organisation of real ants searching for food in nature. Thus, Ant Colony Optimisation is based on the indirect communication of a colony of artificial agents (the ants) mediated by artificial pheromone trails. Pheromone trails in ACO are expressed as numerical values which the ants use to probabilistically construct solutions to the problem at hand. Pheromone trails are adapted during the search to reflect search experience gained by the ants. By design, Ant Colony Optimisation techniques are constructive heuristics. An Ant Colony Optimisation starts with a population of empty initial solutions. The algorithm proceeds by iteratively adding elements to the existing partial solutions in order to form complete feasible solutions for the problem being solved. The problem is usually represented by a graph with vertices and edges, where the vertices represent states of the problem and the edges the possible connections between states. A colony of ants concurrently and asynchronously build solutions by moving through adjacent vertices of the problem on the construction graph thus building paths. At each state of the problem (represented by a vertex in the construction graph) the ant must decide which vertex to visit next, which corresponds to which element to add to the current partial solution. Once an ant has built a complete solution or while the ant is building the solution, the ant evaluates the current solution (which may or may not be partial) and deposits a certain amount of pheromone on the connections (or edges) it has used. This pheromone will direct the search of future ants. To prevent early convergence, Ant Colony Optimisation algorithms maintain a certain level of pheromone evaporation, that is, the pheromone deposited by previous ants decreases over time. It is also possible for Ant Colony Optimisation algorithms to perform what is known as 'daemon' actions, which are actions that cannot be carried out by individual ants but are instead centralised decisions. The key in a successful implementation of an ant Colony Optimisation algorithm is how pheromone is updated, which has a direct impact on the balance between intensification and diversification of the search. Ant systems, which are the early form of Ant Colony Optimisation algorithms, were introduced by Dorigo et al [85, 86]. Further publications can be found on http://iridia.ulb.ac.be/ mdorigo/ACO/ACO.htm maintained by Marco Dorigo.

Genetic Programming Genetic programming is another population-based search method. In genetic programming one breeds a population of computer programs over a series of generations. The starting point is a set of (possibly thousands) randomly created computer programs, which represent the initials solutions. The genetic progamming approach uses Darwinian principles of natural selection, mutation, crossover (or recombination in general), gene duplication, gene deletion and other biological mechanisms. A genetic program first generates an initial population of computer programs (e.g. composition of the functions and terminals of the problem). It then repeatedly executes and evaluates the fitness of each program in the population. Then a new population is created by applying a certain number of operations to certain programs. The programs are selected based on their fitness. The operations applied are mainly reproduction (copy the selected programs to the new population), crossover, mutation and architecture-altering operations. The algorithm then returns the best solution (computer program) created over the generations. Genetic programming is described in detail in [164, 165, 166, 167, 168]

Asynchronous Teams (A-Teams) Asynchronous Team (A-Team) is also a population based approach. In an A-team, artificial agents, each with different skills, operate on different individuals of different populations. A-team is a multi-agent and multi-population approach. Individuals in the different populations are solutions to the optimisation problem to be solved. Agents here are autonomous. Agents have the same work cycle which consists of three phases: select a solution from a population, alter the selected solution (hence the skills needed by the agent to perform this alteration), insert the altered solution in a population (not necessarily the same population). In an A-team approach, agents must work asynchronously, thus the agents work in parallel all the time, each at its own speed. Agents can compete or cooperate with one another though it is conjectured that cooperation should perform better than competition. Agents can be classified in three types according to the type of alteration they perform on a solution; construction, improvement and destruction agents, which respectively construct, improve and destroy solutions. The main idea underpinning the convergence of the A-team approach is that good solutions can be reached provided that improvement agents select solutions randomly with a bias for solutions of higher quality, the destruction agents select solutions randomly with a bias in favour of solutions of lower quality. [235] and [256] provide a good overview of A-teams.

Evolutionary Strategies (ES) ES are closely related to GA's. Originally [114] ES used only mutation, a single individual in the population and were mainly used to optimise real-valued variables. More recently ES have used a population containing more than one individual. They have also used crossover and have been applied to discrete variables [143]. However their main use remains that of real-valued function optimisation using mutation rather than crossover. Mutation is carried out by applying a random number with a Gaussian distribution to the current value. See [18] for a survey of ES methods. Some of the key references include [17] and [112]. It is in his PhD thesis that Schwefel [241] described what is regarded as the seminal work in evolutionary strategies. More recent papers include [111], [110] and [113].

Memetic Algorithms (MA) MA's are basically a combination of an evolutionary strategy with some local search technique. In an MA framework, some local search technique is applied to individuals in the population in the hope to further improve their fitness. A good introduction to MA's can be found in [66]. The term memetic algorithm was first used in [205] which is regarded as the seminal paper in the field of memetic algorithm. There is a web site dedicated to memetic algorithms, http://www.ing.unlp.edu.ar/cetad/mos/memetic_home.html, which is maintained by Pablo Moscato.

3.4 Conclusion

Heuristics are the only option for large real world problems. Over the past two decades or so, metaheuristics have enjoyed an increasing popularity. Recent evidence of this includes [216] which contains a recent bibliography of metaheuristics. [32] reviews metaheuristic methods for combinatorial optimisation. Other recent work in this topic includes [221, 267]. Recent books on metaheuristics include [229], [215], [227], [271] and [124].

A key in the implementation of a search method is the balance between intensification and diversification. Different metaheuristics have different trade-offs between these two essential components of approximate search methods. Thus, different heuristics will have different strengths and weaknesses [48]. It is therefore not surprising that efforts have been made to develop hybrid methods which draw on the advantages of different techniques while leaving out their respective disadvantages. Memetic algorithms are a good example of such efforts, in which evolutionary techniques are combined with local search. There are also hybridisation efforts between different local search approaches (e.g. scatter search with Iterated Restart procedures such as GRASP [231], tabu search combined with scatter search and path relinking [133], ACO with local search [84, 253]).

Another way to exploit the strengths and weaknesses of different heuristics is to combine them in a hyperheuristic framework, which does not aim at solving directly the problem, but, rather, at recommending an appropriate heuristic at each decision point.

Chapter 4

Related work: personnel scheduling

4.1 Introduction

Personnel scheduling is concerned with the determination of appropriate workforce requirements, workforce allocation and workforce duty assignments for an organisation in order to meet internal and external requirements. This involves allocating people (personnel) to timeslots and possibly locations. This problem is often extremely difficult to solve [268, 99]. Providing the right people at the right time at the right cost whilst achieving a high level of employee satisfaction is a critical problem for organisations [99]. Not surprisingly, personnel scheduling has been the subject of much investigation in the literature over the past 30 years with a survey in every decade [21, 265, 26, 24, 99]. Different applications of this problem have been presented in the literature. We can distinguish between general and specific applications. The terms workforce (or manpower, labour, staff or personnel) scheduling (or rostering, timetabling) are often used interchangeably to refer to the general problem. In this paper, we propose to review the literature on the general problem of personnel scheduling, though we shall make mention of specific applications as well. In this review, we focus on various issues covering modelling, scheduling environments, solution techniques, and theoretical studies. The general personnel scheduling problem is typically encountered in service organisations (e.g. call centres, airport ground personnel, etc.). Several frameworks have been proposed to help classify various approaches used to tackle the general problem of personnel scheduling including [21], [265], [34] and [99].

In this chapter, we are concerned (essentially) with the choice of solution technique which addresses the constraints (work regulations) and achieves the objectives (costs, employee preferences, etc.). We first define some of the generic terms used in personnel scheduling jargon [268, 99].

A *shift* is a period of work with a known beginning and ending time within a period of 24 hours. Hence generally an employee works at most one shift per day. For example a typical shift in administration (in the UK) is 9:00 am to 5:00 pm. Split shifts morning/evenings are common in transport and the hotel industry.

A *line of work or workstretch or tour* is a block of shifts (or a block of consecutive days-off and days-on) spanning over a longer-term period (week, fortnight, month). For example a typical workstretch in administration (in the UK) is Monday to Friday. A tour generally follows a certain shift pattern.

The general problem of personnel scheduling can be classified into the following three groups [21, 24].

1. Days off scheduling: the determination of days on and off work for each em-

ployee (usually over a 7-day period).

2. Shift scheduling: selecting, from a potentially large set of candidates, which shifts are to be worked, together with the number of employees needed for each shift in order to meet demand (usually across a daily planning horizon).

3. Tour Scheduling: the creation, for each employee, of a line of work defined as a sequence of days-on and days-off and, for each day-on, which shifts to work (usually across a weekly planning horizon).

The latter problem is the most general as it deals with both days-off and shift scheduling problems simultaneously. An integer programme (IP) was first proposed by Dantzig [77] to formulate the general personnel scheduling problem as follows.

$$Minimise \quad Z = \sum_{k=1}^{n} x_k \tag{4.1}$$

Subject to

$$\sum_{k=1}^{n} a_{qk} x_k \geq r_q, \quad q = 1, 2, ..., m \tag{4.2}$$

$$x_k \in \mathbf{N}, \quad k = 1, 2, ..., n \tag{4.3}$$

where

x_k = number of employees assigned to *schedule* (or *workstretch*) k;

r_q = demand in terms of number of employees required to work in the q^{th} planning interval (also known as time period);

n = number of schedules considered;

m = number of planning intervals scheduled over the the planning horizon;

$a_{qk} = 1$ if time period q is a work period in *schedule* k, 0 otherwise.

Equation (4.1) represents the objective of minimising workforce size. Equation (4.2) expresses the idea that demand (in terms of number of people needed) must be satisfied (this is also known as the coverage constraint). Equation (4.3) ensures the integrality of variable x_k.

Here the term *schedule* should be considered in the broad sense as, depending on the context, it may mean shift or tour or days-off pattern, which corresponds to a problem of shift or tour or days-off scheduling respectively. Although the objective is to minimise workforce size, other objectives include minimising total labour hours, labour costs, unscheduled labour costs, over-staffing, understaffing, number of schedules with consecutive days-off, number of different work schedules used, maximising customer service, or some combination of those [24].

Remark: The above mathematical model, which was first introduced by Dantzig's [77], is also known as the Set Covering Formulation (SCF) or the Generalised Set Covering Formulation (GSCF). Dantzig's model [77] - which dates back to 1954 - is of great importance in personnel scheduling. As will be seen throughout the survey, most mathematical models for the labour scheduling problem are based on Dantzig's set covering formulation, or an extension of it.

Often, a given problem of personnel scheduling is defined within a certain context, known as the scheduling environment. The scheduling environment is generally described in terms of the following:

- *operating hours:* This corresponds to the maximum length of a day of work for the organisation (also known as the opening hours of the organisation)

- *planning period:* This gives the length of the planning period (also known as the planning interval). The planning interval is dictated by the workforce requirements, which is given in terms of number of employees needed for each single period (hence planning period) - for example 15 minutes, 60 minutes.
- *number of breaks:* This gives the number of planned work interruptions per employee - for example a meal break or a relief break.
- *Work contract:* This indicates whether the organisation employs, full-time (FT) and / or part-time (PT) workers. In the case where both FT and PT workers are considered, there usually is a maximum ratio of PT over FT (to ensure a certain minimum presence of FT workers).

The remainder of this chapter is structured as follows. In section 4.2 we shall focus on the general personnel scheduling problem. This is followed by a brief overview of specific applications in section 4.3. We summarise our work in section 4.4.

4.2 The general personnel scheduling problem

This is also known as the labour scheduling problem. In this section we present articles dealing with the general personnel scheduling problem. We start with some of the most widely cited surveys on personnel scheduling (subsection 4.2.1). This is followed by a review of the literature on flexibility modelling (subsection 4.2.2). We show that flexibility can also be achieved and / or enhanced when incorporating simple heuristic procedures (subsection 4.2.3). In subsection 4.2.4, we highlight how theoretical studies have led to the development of new algorithms. Finally, subsection 4.2.5 reviews other exact methods.

4.2.1 Recent surveys

Baker [21] presented an early survey on personnel scheduling in a cyclical environment in 1976. The whole problem of planning the workforce consisted of three steps: The determination of staff requirements, the determination of the number of employees for each shift or shift pattern (as suggested by the author, shift patterns might be differentiated by the placement of the meal breaks and/or relief breaks), the determination of the number of employees for each work pattern. The two latter steps are often referred to as shift scheduling and days-off scheduling problems respectively and are modelled as set covering problems. While some researchers solved the days-off scheduling problem before solving the shift scheduling problem (top-down approach), Baker noticed that it is possible to solve both problems the other way around (bottom-up approach). This clearly shows the existence of a strong interrelationship between the two problems as mentioned by the author, although the problems were typically dealt with separately.

Tien and Kamiyama [265] presented a survey on manpower scheduling algorithms in 1982. The manpower scheduling problem was decomposed into 5 separate but related stages, namely, the determination of temporal manpower requirements, total manpower requirement, recreation blocks, recreation/work schedule and shift schedule. They also developed models at each stage to categorise problem formulation suggested by the various algorithms. Several applications of manpower scheduling are covered. This includes specific applications in sanitation, transportation, law enforcement, nursing and other areas. Most of the articles reviewed used a mathematical programming model for some of the stages. Also some solution methods were used to solve many stages simultaneously. In comparison with Baker's classification in [21], we see that Tien and Kamiyama identified 2 more stages. As Baker suggested in [21], the problems occurring at each stage (whether 3 or 5 stages) of the general manpower scheduling problem should be solved in an integrated fashion to find the best global solution, rather than separately as was the case at the time of publication of his paper. This approach was widely adopted by the time Tien and Kamiyama published their survey. Tien and Kamiyama suggested that future research should focus on the mathematical programming aspects of the general personnel scheduling problem, as this is the most popular formulation to the problem (typically the set covering problem).

A decade later Bradley and Martin [34] presented another survey on personnel scheduling algorithms with an emphasis on applications in hospitals. Rather than adopting the 5-stage decomposition suggested by Tien and Kamiyama [265] they suggested a decomposition of the problem into 3 stages: staffing (the problem of determining how many personnel must be employed to provide a predetermined level of service), personnel scheduling (the problem of determining who works what shift and who has which days off) and allocation (the problem of assigning scheduled personnel to work sites). The entire model of the problem was called the staff planning and utilisation model. This decomposition makes it easier to classify solution methods which are now specific for each stage (this was not the case in Tien and Kamiyama's decomposition). Bradley and Martin also suggested that schedules be classified by the type of schedule developed (cyclical or non cyclical) as well as by technique (heuristic, mathematical programming, self-scheduling). We note that the authors' approach is very similar to that of Baker, 14 years earlier, thus confirming the relevance of Baker's work.

Bechtold et al. [24] compared the performance of nine heuristic solution methods for personnel tour scheduling. Because the problem environment differs from one case to another, the authors compared the solution methods for a given environment where the main criterion is to minimise the total labour hours scheduled subject to the satisfaction of labour requirements. They considered full time employees working either five consecutive or non-consecutive days per week. All daily shifts are nine hours with a meal break during the fifth hour. This occurs in an operating environment of sixteen hours a day, seven days a week. The authors distinguished between Linear Programming (LP) based methods and constructive methods. The LP-based methods solve the personnel scheduling problem by obtaining an LP solution which is then modified to eliminate non-integer variables. Constructive methods start with no employees and iteratively allocate employees to work schedules until all schedule requirements are satisfied. Two LP-based methods and one constructive method gave better results than all other methods when applied to a broad range of labour requirements, distributions with different amplitude levels for each labour requirement distribution. The authors recommended the integration of all three methods in a Decision Support System that service organisations can use.

The latest survey of personnel scheduling was carried out by Ernst et al. [99]. The authors decomposed the general personnel scheduling problem into 6 stages namely, demand modelling (determination of staff requirements), days-off scheduling, shift scheduling, tour scheduling, task assignment, and staff assignment. Various application areas are also given.

4.2.2 Modelling flexibility in personnel scheduling

By the 1990's the literature for personnel scheduling had demonstrated that the use of flexibility in designing employee schedules can result in substantial improvements in manpower utilisation. That flexibility resides mainly in the placement of breaks during the shift. The first attempt came from Gaballa and Pearce [117] who considered break-placement flexibility by including a separate break variable for every period for which a break is allowed for each shift in their IP formulation. However their formulation involved more variables than the usual set-covering formulation, thus creating further problems of size. In light of this, Bechtold and Jacobs [25] proposed an IP formulation in which break-placement flexibility is not expressed in an explicit manner (as was the case with the previous authors [117] who defined explicit break-variables) but rather implicitly through the inclusion of extra-constraints whose role is to link the shift variables with the break-placement flexibility. Once both the optimal number of employees for each shift and the optimal number of (meal) breaks for each planning period is determined, a procedure is used to allocate breaks to employees. Their formulation has more constraints than the set-covering formulation but substantially fewer variables. The implicit formulation required less CPU time when applied to different data sets with different demand patterns and shift lengths using an IP solver (MPOS).

Another attempt to improve on the modelling of the labour scheduling problem came from Thompson [260] who proposed a new formulation for the tour (shift) scheduling problem. Thompson's formulation aimed at overcoming two limitations contained in two previous formulations (Dantzig [77] and Keith [155]): the difficulty of setting the desired workforce sizes in each planning interval so as to maximise profits and the assumption that a surplus employee is of equal value for all planning intervals. In Thompson's formulation the objective function is to maximise marginal benefit due to additional labour capacity and minimise usual labour costs. In addition to the decision variables on the number of employees for each tour (shift), Thompson's IP formulation includes a decision variable on the excess of employees over a certain minimum workforce size for each planning interval. The author applied a simulated annealing heuristic to all 3 formulations on a large set of problem data involving different combinations of customer arrival and customer service configurations. Particularly, Thompson's formulation generated more profit than the two others and allowed for a certain scheduling flexibility in terms of workforce size. During the same year (1995), based on the work of Moondra [204] and that of Bechtold and Jacob [25], Thompson [263] developed a mathematical programming model for the shift scheduling problem. The model was constructed on the principle of using variables for shift types rather than individual shifts. Then variables representing starting and finishing periods for all shift types as well as meal-breaks were linked by different types of constraints that ensured consistency of meal-break periods with the shift types. Thompson used his model to schedule cashiers in a grocery store, employing an IP solver (SAS-OR). His approach turned out to be superior to that of Bechtold and Jacobs [25] in terms of computing time, problem size and flexibility (regarding the placement of breaks).

Aykin too [15] proposed an implicit IP model similar to that of Bechtold and Jacobs [25] for the shift scheduling problem. As in the latter model, Aykin's model involves break variables (number of employee having a break during a given planning period). These break variables are also linked to shift-type variables by a set of constraints. Aykin's model however is more flexible and robust as it can easily cope with more than one break within a shift span as well as overlapping break windows [25] (when the break window of one shift is contained in the break window of another).

Both models have fewer variables than that of the set-covering model. During the same year Jacobs and Brusco [150] independently proposed an implicit model for the tour scheduling problem that allows daily shifts starting-time flexibility. The idea is similar to that for the implicit shift scheduling problem proposed in [25] in that tour-type variables (a tour-type is defined by specifying the shift start-time band associated with it), rather than explicit tour variables, are introduced and linked to shift-type variables (characterised by start-time band) through a set of constraints. As expected, the resulting integer programming formulation has fewer variables than the initial set-covering model. The authors applied their approach to a problem of scheduling toll collectors in an Illinois tollway company and were able to solve larger size problems to optimality than with the set-covering formulation (problems with several million variables using the set-covering formulation were reduced to a couple of thousands variables using the implicit model).

A further effort on the modelling of labour scheduling came from Aykin [16] in 2000, who compared different modelling approaches for the labour shift scheduling problem. The labour shift scheduling problem has traditionally been modelled using the set covering approach proposed by Dantzig [77]. The size of the resulting model is very large for many real-world problems and some researchers have proposed new models of significantly reduced size. This is the case of Bechtold and Jacobs in [25] and Aykin in [15]. Both approaches are based on the original set covering model but with significantly less variables and less non-zero elements in the constraint matrix than the set covering model. Also both models have more constraints than the original one. The author compared the two new approaches on 220 problem instances presenting different demand patterns, different relief and lunch break window sizes and shift start-time patterns. He also considered both a cyclical (24 hours operating time) and an acyclical environment. Aykin's approach has more variables but significantly less constraints and less nonzero elements in the constraint matrix than the approach of Bechtold and Jacobs. Aykin's approach turned out to be more reliable (in terms of the number of problems solved to optimality) and faster (computation time) than the latter. Coincidentally, Brusco and Jacobs [44] published a study during the same year 2000 concerned with the same issue of developing a model of reduced size for the tour/shift scheduling problem. In the set covering model, every feasible tour is explicitly represented as a variable and the possible meal-breaks are known in advance and represented in the constraint matrix. Instead, Brusco and Jacob represented shifts and meal-breaks implicitly as variables and a series of backward and forward constraints which links the meal-break variables with the shift variables and the shift variables with the number of days-on of the week, which thus creates a tour. This is known as an implicit integer-programming formulation (imp. IP). Overall the model size is much smaller than that of the set covering problem. This makes it possible to aim for optimal solutions even for very large scale problems. The authors also proved the usefulness of the model on real-world problems when post-optimisation comparisons of different managerial scenarios (policies) are considered.

Another labour scheduling solution method was presented by Alfares [10] who proposed a two-phase algorithm for solving the cyclical days-off scheduling problem. The cyclical days-off scheduling problem is that of determining the number of workers who will work the different weekly work-patterns considered containing a period of two consecutive days off. Once the problem is modelled as an IP (set covering problem), the minimum workforce size is calculated using Vohra's [270] formula. In a second phase a constraint binding the variables representing the number of workers for each work-pattern to the minimum workforce size is added to the relaxation of the IP and the resulting augmented programme is solved. The author compared his two-phase method to that of integer programming and that of Bartholdi et al. [22] linear programming. The author's methods proved to be more efficient (faster) than both that of an IP solver and Batholdi et al. [22] when applied to 1250 problem instances (with different demand patterns).

Berman et al. [27] tackled a problem of scheduling workforce and workflow in mail processing centres of the US Postal Service. Workers had to be assigned to work stations and each station had its own amount of work (mail) arriving at the stations in different amounts over time. The arrival of work was modelled as a Markov chain. The amount of work at each station could be inventoried and the resulting workflow could be controlled (scheduled) to be processed during a certain period of the day. Workers were allowed to switch from one station to another at different levels of performance (qualifications) during the same shift (one station during the first half of the shift and the other station during the other half, with a meal-break in the middle of the shift). Further flexibility constraints (capacity constraints, time window constraints for work completion etc.) were considered. The authors adopted an IP formulation whose decision variables were represented by the number of workers of each type, working the first half of their shift at one station and the second half at another station, to be scheduled for each operating day of the week. Computational results showed that adding flexibility to the flow of work and to the use of the workforce resulted in substantial reductions of labour costs.

Brusco [36] proposed a dual all-integer cutting plane approach for solving personnel tour scheduling problems. Personnel tour scheduling is very often formulated as an IP (GSCF [77]). When the size of the model is not too large, an IP solver is used to find an optimal solution. Most commercial IP solvers use a branch-and-bound technique. Brusco proposed the use of Gomory's dual, all-integer cutting plane [129] technique to solve the IP. Brusco made three modifications to Gomory's initial technique. These modifications concerned the rule used to select the source row, the incorporation of an additional constraint (the objective cut) and a procedure to overcome the potential problem of oversized elements (a large integer value) in the tableau. The author also considered a second version of the all-integer cutting plane technique which incorporated an advanced start based on the solution to the relaxation of the IP. Both versions were tested against a branch-and-bound technique used in a commercial IP solver. All three techniques were applied to 144 different personnel tour scheduling test problems with different labour demand patterns, 12-hour or 16-hour operating days divided into hourly planning intervals, and 5 consecutive work days per week with a meal-break or a meal-break and a relief break flexibility for full-time workers (part-time shifts did not contain any break). Both cutting plane techniques were superior to the branch-and-bound technique in terms of number of test problems solved to optimality and computing time.

4.2.3 Additional flexibility using heuristics

It should be clear from section 4.2.2 that exact approaches play an important role in the solution to the general personnel scheduling problem, perhaps due to historical reasons connected to Dantzig's set covering formulation [77]. This explains why much effort has been put in the development of efficient mathematical formulations to the problem. The main goal in developing such models is to achieve flexibility: placement of meal and relief breaks during the shift, workforce size, types of shifts used, shift start times, work location, work completion time-windows etc. The advantage of using 'modern' formulations is twofold. Not only this results in substantial savings or reduction in labour costs, but also such models can solve problems of larger size than when using the original set covering formulation (or its generalisation). However the resulting 'flexible' models still involve either more decision variables or more constraints than Dantzig's set covering formulation [77]. Where exact algorithms fail to solve large-size problems, there is room for heuristic methods. Different heuristic and metaheuristic approaches have been developed in order to solve personnel scheduling problems. One of the main advantages of heuristic approaches is that they can help achieve that very flexibility mentioned above.

Simulated annealing

Brusco and Jacobs [37] used a simulated annealing heuristic to solve a cyclic staff scheduling problem. The problem was modelled using the original integer linear programme proposed by Dantzig [77]. The simulated annealing heuristic used two types of neighbourhood move: 'add one employee to' and 'drop one employee from' the schedule. Simulated annealing was compared with several methods including a linear programming-based heuristic, a construction/improvement heuristic and a pairwise interchange heuristic. Experiments were run on different sets of problem instances with different demand patterns. The simulated annealing heuristic was found to be superior to all the other heuristics in terms of solution quality (convergence to near-optimal solutions), robustness, and speed.

Brusco et al. [45] solved a weekly tour scheduling problem at United Airline stations. They formulated the problem using the generalised set covering formulation [77]. Because of the company requirement to produce solutions within a few minutes, the authors considered a three-stage approach based on an existing personnel scheduling software used by the company. Thus a first problem of (daily) shift scheduling was modelled using GSCF. The relaxation of the problem was solved using a shift generation heuristic which produces a set of shifts, based on column generation, with the corresponding number of employees. The company software used those shifts to construct an initial tour schedule. A second module improved on the initial tour schedule. This was a simulated annealing heuristic which used two types of neighbourhood moves ('add' and 'remove' employees). A final stage was to pass the best simulated-annealing tour schedule to the company software for conversion into an actual schedule (assignment of break, days-off etc.). Application of their methods to United Airline stations generated enormous savings in terms of Full-Time Equivalent employees and the authors suggested the use of such a method for similar problems in other service organisations.

Brusco and Jacobs [38] carried out a cost analysis of a continuous and discontinuous formulations for the tour scheduling problem in a continuously operating system. The tour scheduling problem was modelled using the generalised set covering formulation with an additional constraint on the ratio between PT and FT employees, and the objective was to minimise labour cost. In the continuous formulation shifts may overlap between 2 consecutive days; This is not permitted in the discontinuous formulation. The LP-relaxation for both formulations were computed and the resulting lower bound compared in the case of a low PT/FT ratio and a high PT/FT ratio. It appeared that the discontinuous formulation potentially results in an excess of labour cost over the continuous formulation. This was confirmed when using a simulated annealing heuristic applied to both formulations.

Thompson [261] used a simulated annealing heuristic to produce shift schedules in the context of non-continuously available employees, that is, employees who are not permanently available to be scheduled. The problem was modelled using mathematical programming and the simulated annealing heuristic comprised 5 different routines which were applied successively over and over again until a stopping criterion is met. The routines involved the use of neighbourhood moves based on adding or dropping shifts and combinations of these two basic moves. Different shift selection rules were used to apply the neighbourhood moves and these different variants of the routines were experimented with in order to find the best combinations of the variants. Further improvements were made when the heuristic generated multiple schedules instead of a single one. The heuristic was then capable of generating near-optimal schedules in a small proportion (9%) of the time required to generate optimal solutions using an IP solver.

Easton and Rossin [96] also analysed the effect of overtime scheduling policies for service organisations. Using a base case situation in which employees worked 40 hours a week (8hours per day, 5 days a week) the authors considered different overtime shifts/tour lengths (shifts longer than 8hours, tours longer than 40 hours). The tour scheduling problem was formulated using GSCF [77]. Using a heuristic solution method based on column generation and simulated annealing, the authors conducted experiments involving general demand patterns, in different operating environments with various policies. It turned out that overtime can help achieve significant savings in terms of workforce size, schedule efficiency and total cost.

Brusco and Jacobs [39] tackled a problem of personnel tour scheduling with restrictions on shift starting times for an American airline company. The problem was modelled using the classical set covering problem with additional constraints reflecting the restriction on the maximum number of both full-time and part-time shift starting-times as well as the ratio between part-time and full-time workers. Due to the large size of the problem, the authors proposed a two-stage method to solve the problem. In the first stage the tour scheduling problem was reduced into a (daily) shift scheduling problem and, using the dual simplex algorithm a constructive heuristic was employed to select (add) a shift to the daily work schedule. In the second stage an initial tour schedule was first constructed using a constructive heuristic which added workers until all demand was satisfied. A simulated annealing procedure was used to improve on the initial solution. The procedure used two types of neighbourhood move: 'add one worker' and 'drop one worker'. Different rules for selecting the shifts and their starting times (in stage one) and for adding/dropping a worker (in stage two) were considered. Application of the authors' methods to the company's problem instances produced savings (in terms of full-time workers equivalent and hence in dollars) over the method currently used by the company.

Lesaint et al. [176] addressed a problem of workforce scheduling for British Telecommunications plc (BT). BT engineers had to be allocated to a certain number of tasks (maintenance) to be performed in different locations and different periods of time. Several problem-specific constraints were considered including constraints on off-hours and other predefined breaks for engineers, and constraints on matching tasks to skill levels. Some precedence constraints existed between certain tasks. The objectives were the maximisation of the productivity of the workforce, maximisation of service quality, best utilisation of skills, minimisation of operational costs, maximisation of workers' preferences. The authors employed a two-phase solution method. The first stage was a constructive heuristic which gradually allocated tasks to the workers. This was done based on a tree search and involved some backtracking in case of infeasibilities. The second stage was a simulated annealing heuristic which used a 'relocate' neighbourhood move. Relocate alters the current solution by randomly selecting an engineer's tour and attempting to change the position of a randomly chosen task for the engineer. The feasibility of a schedule was modelled as a Constraint Satisfaction Problem (CSP) and verified using a constraint programming solver. Implementation of the authors methods yielded solutions of better quality than those obtained by BT's current workforce scheduling system. This also generated important annual savings estimated in millions of US dollars.

Tabu search

Easton and Rossin [95] tackled a tour scheduling problem formulated as a stochastic goal programme when labour requirements for each planning period was a random process (due to random demand). The authors proposed that the deterministic goal programme formulation be replaced with a stochastic goal programme. In the deterministic goal programme approach ideal labour requirements for each period were estimated (using for example marginal analysis techniques) and input to the deterministic goal programme. If the estimates are erroneous, the resulting scheduled workforce (i.e. after solving the deterministic goal programme) may be costly and oversized. Unlike deterministic goal programmes, stochastic goal programme simultaneously optimise the service level, minimise workforce size by incorporating the different possible required workforce sizes associated with their respective probability in the deterministic goal programme. Using tabu search, the authors compared both deterministic goal programme and stochastic goal programme formulations on a large set of problem instances and the latter outperformed the former in terms of labour cost and workforce size.

Genetic algorithms

Cai and Li [56] tackled a tour scheduling problem where employees had different skills. More precisely there were 2 types of job, each with a certain type of workers (skills) and there was a third type of worker who can work either type of job. The number of workers of this latter type was bounded. The tour scheduling problem was modelled as an IP with 3 objectives (first minimise labour costs, then maximise labour surplus, and then balance staff distribution). The problem was solved using a genetic algorithm with crossover and mutation operators. Infeasibilities due to the application of the GA crossover were heuristically repaired. The algorithm gave good results when applied to various real-world problem instances.

Easton and Mansour [94] employed a unified mathematical programming model for a family of deterministic and stochastic labour scheduling problems modelled as Generalised Set Covering Problem, Deterministic Goal Programme or Stochastic Goal Programme. A distributed genetic algorithm which consisted of evolving different populations simultaneously in a network was used to solve these labour scheduling problems. The distributed genetic algorithm was applied to three different sets of published test suites. The authors compared their methods with tabu search, branch-and-bound and simulated annealing. They found that the distributed genetic algorithm outperformed the latter methods in terms of mean error maximum error and percentage of least cost solution.

Other heuristic methods

Many solution methods in this category are multi-stage approaches. The idea is to solve the personnel scheduling problem in different phases. The output of one phase being fed into the input of another. In each phase the solution technique used may or may not be an exact algorithm. The main problem in such multi-stage approaches is that the quality of the final solution(s) is very much dependent on the quality of those solutions obtained during the intermediate stages. There is therefore no guarantee of optimality at the end of the overall solution process.

An example of multi-stage approach is that used by Love and Hoey [183] who tackled a problem of labour scheduling for a chain of fast-food restaurants. They represented their problem using linear programming and decomposed the linear programme into two subproblems which are both solved as minimum cost network flow problems. No computational results were provided.

Schindler and Semmel [239] tackled a problem of labour shift scheduling at Pan American World Airways. Baggage handlers had to be assigned to daily part-time/full-time shifts. The problem formulation was based on the classical set covering problem with additional constraints reflecting the maximum ratio of part-time workers over full-time workers. Another constraint was not to allow the existence of butting part-time shifts (that is when one shift starts within a minimum number of 15-minutes period after another shift ends). Shifts can start at different (but given fixed) 15-minute intervals. Two types of part-time shifts (4-hour and 5-hour) were considered. Full-time shifts contained three breaks (one meal-break and one relief-break before and after the meal break). The author used a two-step method to solve the integer programme. The first step scheduled shifts without consideration of the different breaks. Then in a second step a second IP, similar to the first, was solved in order to determine the different breaks within the shifts. Implementation of the authors' approach reduced the deployment of staff, used the existing staff more efficiently thus reducing costs (the shift scheduling problem was solved manually before). No computational results were presented.

Rafaeli et al. [226] presented a 'weight' and 'improve' algorithm for the general problem of resource allocation where the resources can be people. The problem was formulated using a mathematical programming model with linear constraints and was represented as a graph where the nodes represent the tasks to be allocated to the resources and an arc exists between any pair of conflicting tasks (a task is allocated to one resource and only one task can be processed by each resource at a time). A greedy algorithm based on weights determined for each task-resource assignment was used to generate an initial solution. The solution was further improved by a second heuristic which switched some assignments of tasks to other resources. Computational experiments showed that the algorithm performed better than other greedy algorithms reported in the literature.

Jarrah et al. [151] proposed an integrated approach for solving large-scale tour scheduling problems. In their approach, the first problem was that of scheduling daily shifts. Their shift scheduling model was a combination of Dantzig's set-covering formulation [77], the implicit formulation of Bechtold and Jacobs [25] and the lower bound procedure of Burns and Carter [55]. Because of the large size of the resulting formulation, aggregate variables were introduced as well as surrogate constraints, which resulted in a new (augmented) formulation. In the augmented formulation all integer variables (except for the aggregate ones) were relaxed and the mixed integer programme thus obtained (master problem) was solved using an IP solver. Given the fixed optimal values of the aggregate variables, the augmented formulation was decomposed into seven independent shift scheduling problems, each of which were solved to optimality. A heuristic procedure was invoked in case the solution obtained were not integral. Once shifts are determined, two procedures are used to assign breaks to shifts and shifts to tours respectively. The authors applied their approach to a problem of staffing at General Mail Facilities where workers sort mail on a daily basis. Twenty-eight different problem instances of large sizes were used and for all of them their method found optimal solutions, without using the repair heuristic, in a reasonable amount of time.

Bailey et al. [19] suggested an integrated approach for project task and manpower scheduling problems. The problem was formulated using an IP which relates the staffing level requirements with the start times and duration of the project tasks as constraints. A dynamic-programming based heuristic was then used to solve the problem. This resulted in labour cost and total cost savings over the traditional two-step heuristic procedure which first determines the start times and duration of the project tasks before calculating the number of employees per period.

Ashley [14] tackled a problem of personnel scheduling at a university library. The problem was to determine weekly schedules for library staff. Each member of staff must be assigned to desks at certain times throughout the week. Staff members had different periods of availability and there were constraints on the workload of each staff. The problem was modelled as an IP and the objective was to minimise the total number of uncovered (unfilled) slots. The author used a spreadsheet system to solve the problem. Not only did this generate savings in the time used to compute a schedule manually but it also produced higher quality solutions.

Tsang and Voudouris [266] introduced a Fast Local Search (FLS) combined with a Guided Local Search (GLS) and applied it to a workforce scheduling problem. FLS is a hill climbing method which heuristically ignores moves used in the past without any improvement and GLS is a method which diversifies the search to other regions each time a local optimum is reached. The authors applied FLS+GLS to British Telecom's workforce scheduling problem. They compared FLS with a simple hill climbing and noted that the activation bits used in FLS to ignore certain moves helped to speed up the method with no convincing evidence that solution quality was sacrificed.

Brusco and Johns [42] proposed a heuristic method for the discontinuous tour scheduling problem (i.e. shifts are not allowed to overlap between 2 consecutive days). The tour scheduling problem was first modelled as a GSCF [77] and, instead of solving the LP-relaxation of the GSCF and counting on rounding and improvement procedures to determine good integer solutions, the authors' heuristic imposed integer restrictions on subsets of tour variables. More precisely the tours that begin (end) in a given hour may be constrained to be integer while others are continuous thus allowing information concerning demand in all time period to influence the values of the integer tour variables. Experiments were carried out in the case where all employees are FT and in the case where some employees are PT. In both cases the authors' heuristic compared favourably with some of the best LP-based heuristics reported in the literature.

Thompson [262] tackled a problem of scheduling telephone operators for a telephone company. Each worker chose a certain number of daily shifts that s/he would like to work. The problem was to assign daily shifts to workers. Thompson modelled the problem as a mathematical programme and the objective function was to minimise the number of unassigned shifts as well as to satisfy personnel in order of seniority (priority was given to senior workers first). The author used a heuristic method to generate solutions on a PC. Both management and workers saw the resulting decision support system as an improvement over the previous manual procedure. No computational results were provided.

DuCote and Malstrom [91] described a Decision Support System (DSS) to model personnel scheduling in a manufacturing environment. The problem was to schedule workers to a certain number of tasks and to a certain number of timeslots during which these tasks would be performed depending on the location of the work (work centre). The cost of a schedule depends on the worker qualification, the work centre and the time period of the work. The DSS consisted of four modules, the last of which heuristically assigned workers to work centres by both worker category and time period. The heuristic took account of different factors including worker interchangeability, new hires, extra shifts, layoffs, overtime, weekend work and unexpected absences.

Brusco and Jacobs [40] conducted an experimental analysis and a case study as to which set of shift starting-times to choose for scheduling labour tours. The labour tour scheduling was formulated as an IP based on GSCF [77] with two additional sets of constraints on the restriction of the number of (daily) shift starting-times. The objective was to minimise the workforce size. Experiments were conducted on real-world sets of labour requirements for a one-week planning horizon divided into hourly planning intervals. Only full-time workers without meal/relief breaks were considered. Two workweek alternatives (5 days-8 hours per day and 4 days-10 hours per day) and 3, 4 or 5 starting-times were considered. It turned out that restricting the number of starting-times to 4 or 5 did not result in a substantial increase of workforce. The authors also solved all IP's where all 24 starting-times were considered. This was possible using the cutting plane technique proposed by Brusco [36]. However a bad selection of the set of starting-times resulted in high workforce volumes. The authors then considered a case study of analysing different starting-times policies for customer representatives at Motorola's LMPS Radio Network Solutions Group call centre. The case study demonstrated that starting-time decisions must be examined in relation to other scheduling policies and, due to the resulting large size of the problem, a constructive heuristic approach was employed to solve the tour scheduling problems.

Lin [178] tackled a problem of personnel scheduling at a telephone call centre.

The problem was to determine daily schedules for telephone call operators for a monthly horizon. There were 3 types of shifts (day, night and evening) and various types of constraints were considered (shift type precedence constraints, staffing requirements, staff day-off requests, functional constraints). Lin developed a 3-phase solution method which first calculated hourly call forecasts and staff requirements. It then determined daily workforce sizes and assigned meal breaks. The 3rd phase used the Burns and Carter algorithm [55] to calculate the monthly roster (days-off scheduling, and shift assignment). Implementation of the resulting workforce system not only helped save a lot of time but also satisfied more constraints than the manual scheduling method.

4.2.4 When theory leads to the development of new algorithms

Van den Berg and Panton [268] investigated the theoretical existence conditions of a case of personnel scheduling where both continuous and forward rotating shift assignments are considered. The authors used Tien and Kamiyama's [265] 5-stage decomposition of manpower algorithms. The fifth stage of this decomposition is concerned with the problem of shift assignment. The shift assignment is continuous if the same shift is worked every day of the workstretch (a period of consecutive days of work). A forward rotating shift assignment is one where different shifts are worked within the same workstretch such that later shifts in the workstretch have a later starting time than earlier ones. The two cases of continuous and forward rotating shift assignments were regarded as a very common requirement in workforce scheduling. The authors used a network model upon which an algorithm was developed to search for a continuous and forward rotating shift assignment. The algorithm was capable of finding such continuous and forward shift assignments in a high proportion of cases solved. Note that this is the first paper presenting a theoretical study on the existence conditions of continuous and forward rotating shift assignments.

A related study was conducted by Lau [174] who also investigated the complexity of the shift assignment problem (SAP). The author tackled the problem of manpower shift scheduling from a theoretical point of view. Using the decomposition framework of Tien and Kamiyama [265] for manpower scheduling algorithms, the author proved that SAP is NP-hard when shift change constraints are considered. Lau also developed a greedy algorithm for the monotonic Changing SAP (CSAP). In CSAP a worker is not allowed to work on an earlier shift than that of the previous day (which corresponds to the forward rotating shift assignment in [268]). The algorithm was extended to solve more complex SAP's including cyclic schedules, consecutive same shift constraints, spare demands and non-monotonic shift changes.

Brusco and Johns [41] proposed a pre-emptive goal programming (PGP) method for solving a tour scheduling problem. The problem was usually solved with the primary criterion of minimising labour cost. Brusco and Johns' work aimed at obtaining solutions with an even distribution of labour surplus at the same optimal labour cost. In the first instance the GSCF was used with the objective of minimising labour cost. Then a second formulation was utilised with the objective of minimising the maximum of the ratio of labour surplus over labour requirement (demand) while constraining solutions to keep the same cost as that of the generalised set covering formulation (GSCF) optimal solution. Experimental results showed that the authors' approach provided solutions not significantly worse than those of the GSCF but also generated solutions with a significantly lower variance than those of

GSCF.

Narasimhan [208] tackled a problem of days-off scheduling involving different categories of workers (qualification levels) whereby a high-level worker can do the work of a low-level worker. In his problem, employees worked up to 5 days a week and had at least 2 days off. There was a constant weekday demand (required number of workers) and a different constant weekend demand. The problem was to assign days-off to each worker in such a way as to minimise the workforce size while respecting the work demand required per category as well as cumulatively. The author used a multi-step algorithm which first calculated a lower bound for the workforce size for each (worker) category as well as cumulatively. The algorithm then determined weekends off as well as other days off for each category. It then assigned shifts to workers. An example was given to illustrate the resulting algorithm.

In [209], Narasimhan extended his methodology to multiple shift scheduling on 4-day and 3-day workweeks. A related approach is that of Hung and Emmons [149] where a 3-4 compressed workweek was considered. In a 3-4 compressed workweek, employees cyclically work 3 days and have 4 days off the first week and then work 4 days with 3 days off the second week and so forth.

Hung [148] studied a problem of workforce scheduling under annualised hours. The idea of annualised hours involves hiring workers to work for a given number of hours per year. This avoids varying workforce size for frequent hiring, firing, training, especially when the demand is seasonal. In this annualised hours context, Hung proposed a workforce scheduling algorithm similar to that used in [149], [208] and [209]. It first calculates the minimum workforce size and then iteratively assigns workers to work more (less) days during busy (slow) weeks. Several application examples were given to illustrate the scheduling algorithm.

4.2.5 Other exact methods

Beaumont [23] solved a real-world staff scheduling problem similar to the problem of determining working time as well as days-off for repair-people travelling between and servicing faulty lifts whilst attempting to maintain a certain service level (number of staff needed at any time to meet demand). The problem is formulated as a mixed IP. The problem size was reduced by omitting some redundant variables and constraints and the problem was solved using an IP solver. The obtained solution was used to compare with the operating cost of the company's approach.

Hueter and Swart [147] tackled a problem of labour scheduling for a fast-food chain of restaurants. In their personnel scheduling problem, labour requirements were determined for each 15-minute interval of the day using simulation and forecasting techniques. The problem was then modelled as an IP and solved using an Lagrangean multipliers. Their labour management system had been used in many stores (restaurants) and resulted in important savings as well as improvement of quality of customer service. No computational results were provided.

Brusco and Johns [43] analysed the effect of different policies for scheduling a multi-skilled maintenance staff at a paper mill factory. Each type of work could be performed by workers with varying levels of productivity. This is known as cross-training. Given a cross-training policy (represented by a matrix of levels of productivity for each worker class and for each work category) the problem of scheduling daily shifts was formulated as an integer linear programme with a 30-minute break for each worker. 36 cross-training policies were considered based on the possibility of cross-training in one or two secondary skill class(es) (the primary skill class is the one for which the worker was initially employed) at 100% or 50% productivity with a symmetric or asymmetric matrix (in a symmetric matrix the productivity level of employee skill class x working job of skill class y is the same as that of employee skill class y working a job of skill class x). Different demand patterns were also considered. Experimental results revealed that policies assuming a 100% productivity for secondary skill class produced the minimum cost of labour utilisation. The authors recommended the use of policies with asymmetric configurations assuming a 100% productivity for secondary skill classes. It also appeared that partially cross training employees in work categories can result in significant cost savings.

Billionnet [31] tackled a hierarchical workforce scheduling problem in which a higher qualified worker could be assigned to lower-qualification work but not vice versa. The problem was to determine the days off for each worker. It was modelled using MPL (Mathematical Programming Language) and solved using an IP solver. Results obtained showed that integer programming was an effective approach as it produced good results on various instances in a short amount of time. The model was extended to take account of further constraints or objectives in particular maximising the number of consecutive off-days for each worker. In this latter objective the choice of formulation turned out to be of crucial importance for obtaining good results in reasonable computing time. Although integer programming seemed to be suitable for their workforce scheduling problem, it would be interesting to apply their approach to large-size problems in order to test the robustness of the method.

Alfares et al. [11] proposed an integrated approach for solving project operations and personnel scheduling simultaneously. Project operations scheduling determines a schedule (calendar) of the different operations of a project to be performed within a certain period of time subject to temporal and precedence constraints on each operation (task). A problem of scheduling personnel occurred since the different tasks of the project had to be performed by workers. Both problems were usually solved using a two-phase approach that first solves the project scheduling problem and then the personnel scheduling problem based on the results of the first problem. The authors modelled both problems as a unique IP and solved the resulting programme using an IP solver. Computational results showed that their approach was superior to the two-phase one in terms of total project cost, labour cost and labour utilisation.

Kumar and Arora [169] addressed a workforce scheduling problem at a US newspaper company. The authors decomposed the complex heterogeneous system of the newspaper into homogeneous classes based on the type of activity and the type of section in the newspaper (i.e. news, display, advertisements). This in turn determined the different equipment sizes required. Based on these requirement sizes, the workforce size was calculated for each shift. The implementation of the workforce planning model, along with some equipment innovations, was expected to generate significant savings to the company.

4.3 Specific applications

In addition to the general labour scheduling problem detailed above, there have been a number of specific applications. Within each application, personnel scheduling problems have been solved using various techniques, often developed especially for that specific application. Thus each specific application domain has developed in its own right. Specific personnel scheduling applications together with some example references include:

1. Transportation: This deals with the scheduling of crew members for buses, airplanes and trains. A comprehensive survey can be found in [33]. More

Authors/year	Scheduling environment	Model	Method
Bechtold and Jacobs'90 [25]	(24-hr, 0-60min, 1b, FT)	imp. IP	IPS
Schindler and Semmel'93 [239]	($\geq 8.5hr$, 15min, 3b, FT/PT)	IP	IPS
Thompson'95 [263]	(12/15/16/20hr, 15/30min, 1/2b, FT/PT)	imp. IP	IPS
Aykin'96 [15]	(24-hr, 15-min, 3b, FT)	imp. IP	IPS
Thompson'96 [261]	(15-hr, 15/30-min, 0b, FT/PT)	IP	H
Thompson'97 [262]	upon employee's preferences	IP	H
Hueter and Swart'98 [147]	(-, 15-min, -, -)	IP	IPS
Brusco and Johns'98 [43]	(24-hr, 30-min, 1b, FT)	IP	IPS
Aykin'00 [16]	(24-hr, 15-min, 3b, FT)	imp. IP	IPS

Table 4.1: Shift Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver, H = heuristic, imp = implicit.

Authors/year	Scheduling environment	Model	Method
Alfares'98 [10]	(-, 1-d,-, -)	IP	IPS
Billionnet'99 [31]	(3/4/5-d, 1-d, - , FT)	IP	IPS
Alfares et al.́99 [11]	(5/6/7-d, 1-d, -, FT)	IP	IPS

Table 4.2: Days-off Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour, d = day] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver.

Authors/year	Scheduling environment	Model	Method
Love and Hoey'90 [183]	(18hr, 30min,-, -)	IP	IPS
Bechtold et al.'91[24]	(16hr, 1hr, 1b, FT)	IP	IPS + H
Brusco and Jacobs'93 [37]	(24hr, 1hr,1b, FT)	IP	H
Jarrah et al.'94 [151]	(20hr, 30min, 1b , FT/PT)	IP+ lower bound	IPS + H
Brusco et al.'95 [45]	(24hr, 15min, 2b, FT/PT)	IP	H
Brusco and Jacobs'95 [38]	(24hr, 1hr,1b, FT/PT)	IP	IPS + H
Brusco and Johns'95 [41]	(12-16hr, 1hr,1b, FT/PT)	IP	H
Thompson'95 [260]	(18hr, 1hr, 1b, FT)	IP	H
Bailey et al.'95 [19]	(-, 1d, -, -)	IP	H
Brusco and Johns'96 [42]	(16hr, 1hr,1b, FT/PT)	IP	H
Jacobs and Brusco'96 [150]	(24hr, 1hr, -, -)	IP	IPS
Easton and Rossin'96 [95]	(16hr, 1hr, 1b, FT/PT)	SGP	H
Beaumont'97 [23]	(24-hr, 20min, 1b, FT/PT)	IP	IPS
Berman et al.'97 [27]	(24hr, 30min, 1b, FT/PT)	IP	IPS
Easton and Rossin'97 [96]	(12, 16, 20hr, 1hr, 1b, FT)	IP	H
Brusco and Jacobs'98 [39]	(24hr, 15min, 0b, FT/PT)	IP	H
Brusco'98 [36]	(12/16hr, 1hr, 2b, FT/PT)	IP	IPS
Easton and Mansour'99 [94]	(16hr, 1hr, -, FT/PT)	IP	H
Brusco and Jacobs'00 [44]	(7d-24hr, 1hr, 1b, -)	IP	IPS
Cai and Li'00 [56]	(24hr, 1hr, 1b, FT)	Multi-criteria IP	H
Brusco and Jacobs'01 [40]	(24hr, 1hr, 0b, FT)	IP	IPS + H

Table 4.3: Tour Scheduling environments: The scheduling environment in the second column is given in the following format: (operating hours [hr = hour] , planning period [min = minute], number of breaks [b = break], existence of Full Time [FT] and/or Part Time [PT] employees). IP = Integer Programme, IPS = IP Solver, H = heuristic, imp = implicit, SGP = Stochastic Goal Programme.

recent articles include [13, 62].

2. Health care: The most frequent problem is that of scheduling hospital nurses. A problem of scheduling nurses is also studied in this thesis. A survey can be found in [34]. More recent papers include [89, 152, 191, 9].

3. Protection and emergency services: This includes the scheduling of police officers, ambulance drivers, etc. Papers in this application include [257, 98].

4. Government: [177, 207].

5. Venue management: [88, 188].

6. Financial services: for example bank and accounting firm personnel [184, 83].

7. Hospitality and tourism: for example hotel personnel[223, 181].

8. Retail: for instance [146, 141].

9. Manufacturing: for example [1, 100].

10. Educational institutions: though this is often classified as timetabling; for example [236, 73].

11. Miscellaneous applications: Religious institutions [67], judicial institutions [240], fast food personnel [183], security personnel [192], media personnel [131], and other kinds of commercial companies [69].

4.4 Summary

Personnel scheduling is a very wide field with hundreds of articles reporting different modelling and solution techniques used in various applications, some of which have developed into fields in their own right (e.g. crew scheduling, nurse scheduling). In this survey we have focussed on the general personnel scheduling problem also known as the labour scheduling problem. Labour scheduling comprises shift, days-off and tour scheduling problems. One important element in solving these problems is the environmental conditions (operating hours, planning periods, existence of breaks for employees, existence of part-time employees in addition to full-time ones). In Table 4.1, Table 4.2 and Table 4.3 we report some of the environmental conditions encountered in the literature for shift, days-off and tour scheduling problems respectively. We also mention the type of solution method utilised (exact or heuristic). We note that both exact and heuristic based methods have been used.

As in other real-world applications, models developed for personnel scheduling are sometimes a simplification of the reality, though in most cases they remain realistic enough to produce practical solutions. Hence most personnel scheduling application solutions were actually implemented. In most cases the implementation of the solution resulted in significant savings. Not surprisingly, many major organisations now have a team or department specialised in just workforce scheduling (e.g United Airlines, Lufthansa, British Telecom, etc.). When heuristic methods are used, 'add' and 'drop' an employee are among the most widely used moves. For large organisations both modelling and implementation of the solution require a considerable amount of time due to several complex considerations inherent to the real world.

Research efforts on labour scheduling can be grouped into three categories which lay the foundations for current and future trends in the field:

The need for powerful mathematical models which allows flexibility: This is due to the peculiarity of this field. Due to the fact that in personnel scheduling one deals with people (rather than machines as in machine scheduling) one has to accommodate, employee preferences, work regulations (which are often the result of workers' unions and / or government regulations) and high-quality customer service. Efforts in this direction include [25, 260, 150, 15, 44, 16].

The need for improvement on existing exact solution methods: This is only possible when dealing with fairly small problem sizes. Examples include the work by Brusco [36].

Finally the use of heuristic method appears to be very promising as it allows for better handling of large problem sizes. Furthermore it is sometimes possible to obtain good-quality solutions quickly. Examples include [261, 45, 95, 266]. Simulated annealing seems to be the most widely used metaheuristic method perhaps due its ease of implementation [142]. As the survey shows (see section 4.2.3), the use of heuristic methods in personnel scheduling can help enhance flexibility in terms of placement of meal and relief breaks during shifts, workforce size, types of shifts used, shift start times, work location, work completion time-windows etc. A typical problem in many service organisations is the fact that demand is often highly variable (see section 4.1). In this situation, exact algorithms can only produce solutions that are optimal for the original problem data. Because heuristic methods produce approximate solutions they are less dependent on the accuracy of the problem data than exact methods. An additional advantage of heuristic methods is that they require shorter implementation times.

A number of personnel scheduling heuristic methods have been developed and it is not always clear which heuristic(s) to use. A step towards answering this question is the investigation of hyperheuristic methods, that is, heuristics which recommend an appropriate heuristic chosen amongst other heuristics in order to solve a given personnel scheduling problem. By using hyperheuristics, not only are we able to address the issue of choosing an appropriate heuristic amongst a given number of them, but also we are able to preserve all qualities associated with heuristic methods as discussed above (i.e. flexibility enhancement). As we shall see in later chapters, there is room here for hyperheuristics to be employed. Indeed, one of the aims of this thesis is to raise the level of generality of (meta)heuristics by developing hyperheuristics which can manage several low-level heuristics and cope with different constraints in different domains. Chapters 6, 7 and 8 will demonstrate this.

Chapter 5

Hyperheuristics for personnel scheduling

5.1 Introduction

In chapter 1, we gave an introduction to the concept of hyperheuristics including a general hyperheuristic framework as illustrated in Figure 1.1. For a given problem and a given number of low-level heuristics for that problem, the hyperheuristic approach that we discuss in this thesis selects and applies an appropriate low-level heuristic at each decision point. This process continues until a stopping condition has been met. The hyperheuristic then outputs the best solution(s) found during the process.

The key question is how the process is actually carried out, or, in other words, how should we design and develop a hyperheuristic?

In order to generate an automated method for a given NP-hard optimisation problem, a software developer must produce a solution technique which takes into account the context within which the technique will be used. For example:

- Will the resulting software program be used for other types of problems?
- Does the problem owner place much emphasis on solution quality?
- What are the problem owner's criteria for acceptable solution quality?
- Is the problem owner prepared to invest a lot of time in the development of the technique?
- Is computational time an issue?

These issues are often discussed in the scientific community and are also known to be part of soft operational research (Soft OR) / system dynamics; in which the nature of concept definition and comparison is highly qualitative, as opposed to quantitative [233, 269].

In this thesis, we are looking into developing a technique which can be easily applied to different problems; i.e. we require a method which was not designed with one particular problem (or problem instance) in mind but is, instead, applicable to a wide range of problems and domains. This is not to say that we intend to develop a panacea. The aim is not to solve all problems with one method, but is to raise the level of generality from its current low-levels. The No Free Lunch Theorem [277] implies that the former objective cannot be achieved anyway. The more general method should require as little customisation as possible when applied to a new problem, perhaps at the expense of reduced but still acceptable solution quality (when compared with made-to-measure bespoke metaheuristic techniques). It should therefore be a quick-to-implement method.

5.2 Designing hyperheuristics

In this section we discuss design issues related to the development of hyperheuristics. The number of design choices possible is so large that it is difficult to consider every possibility. As will be seen throughout this section there is a wide range of choices possible for each topic discussed, and one has to limit these choices to a small number in order to allow for a thorough investigation of the chosen hyperheuristic design and thus gain valuable insights. To discuss these design issues, and to define directions for hyperheuristic research, it is useful to imagine the hyperheuristic paradigm as having three levels of abstraction. We depict this in Figure 5.1.

5.2.1 Level 0: Problem representation

Partial and complete solutions

At the lowest level (level 0), we must first determine whether one deals with *partial* solutions or *complete* solutions. These notions are best explained by way of example. An example of a hyperheuristic which deals with partial solutions is presented in [234]. The problem is that of bin packing. A given number of items must be packed in bins. Bins are of different sizes and the aim is to pack all items in the minimum number of bins. A complete solution to the problem is one in which *all* items have been placed in bins, (an optimal solution being one in which the number of bins used to pack all items is minimum). A partial solution to the problem is one in which *not all* items have been packed. We should be clear about whether the hyperheuristic will be dealing with partial or complete solutions during its execution. It should be noted however that the focus of our attention is less on the actual nature of the solution (whether partial or complete) than on the hyperheuristic process.

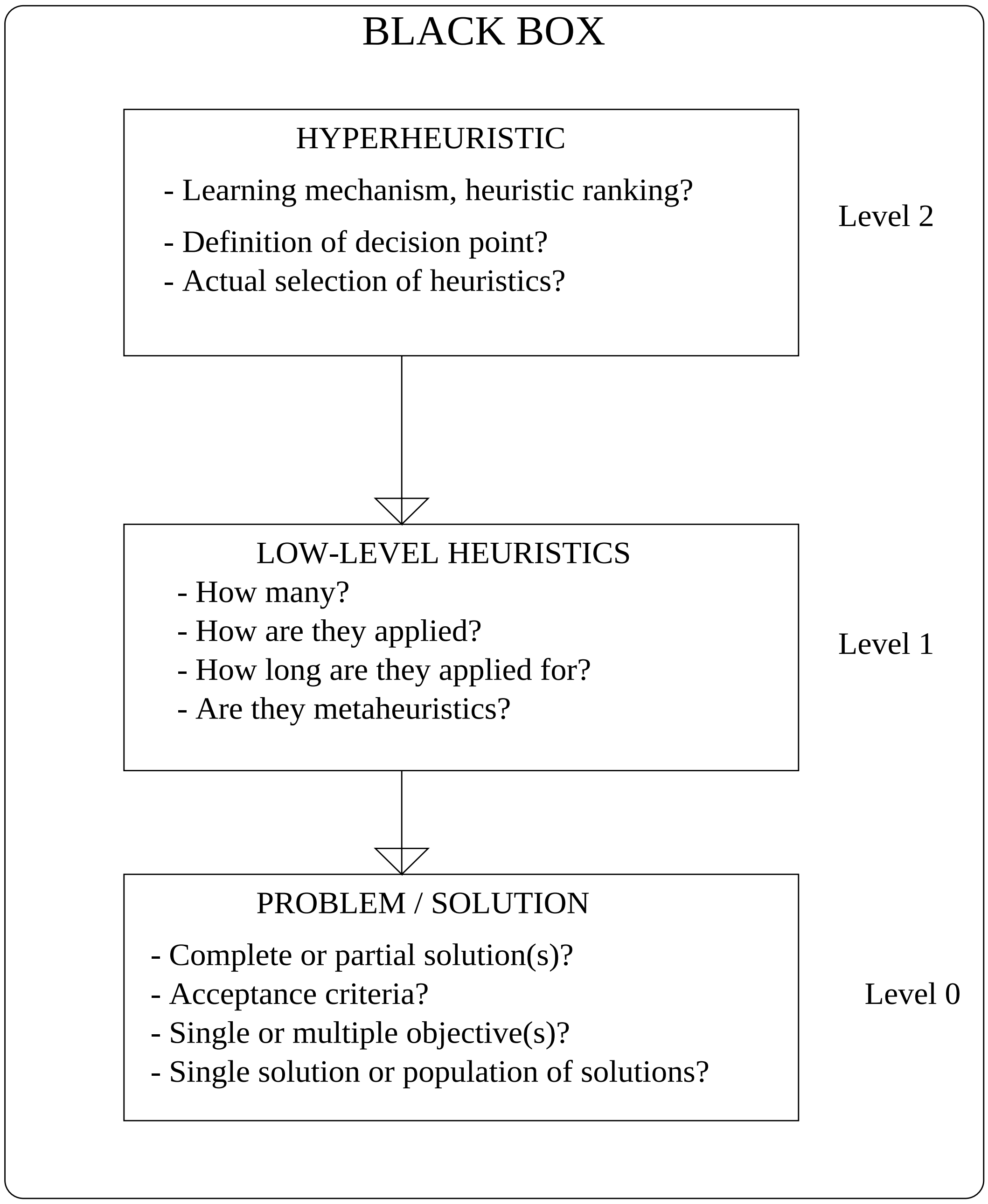

BLACK BOX
HYPERHEURISTIC
- Learning mechanism, heuristic ranking?
- Definition of decision point?
- Actual selection of heuristics?
Level 2
LOW-LEVEL HEURISTICS
- How many?
- How are they applied?
- How long are they applied for?
- Are they metaheuristics?
Level 1
PROBLEM / SOLUTION
- Complete or partial solution(s)?
- Acceptance criteria?
- Single or multiple objective(s)?
- Single solution or population of solutions?
Level 0

Figure 5.1: Design issues for the development of a hyperheuristic

Consequently, by partial solutions we mean the intermediate incomplete solutions formed during an incremental and constructive process. This is explained in the next two subsections.

Hyperheuristics dealing with partial solutions

In the hyperheuristic approach of [234], the hyperheuristic starts with an empty partial solution, that is, no items are packed yet. Then during the process, the hyperheuristic deals with partial solutions. It selects and applies an appropriate bin-packing heuristic to a given partial solution, then, if the resulting solution is not complete, the hyperheuristic must select and apply another (possibly the same) heuristic to the resulting partial solution, and so on until a complete solution has been obtained.

The question of whether we are dealing with a partial or complete solution is crucial when developing a hyperheuristic. Effectively, depending on whether the hyperheuristic will be manipulating (by way of the low-level heuristics) partial or complete solutions, the sort of information collated in order to select a low-level heuristic will be different. Thus in [234], what really matters is information about the state of the problem, e.g. what percentage of small, medium and large items have been packed. Figure 5.2 illustrates a general framework for this type of hyper-heuristic.

A general hyperheuristic procedure in this case (i.e. when dealing with partial solutions) can be illustrated in the following pseudocode:

```
Do
    Select a low-level heuristic and apply it to the current problem state
```

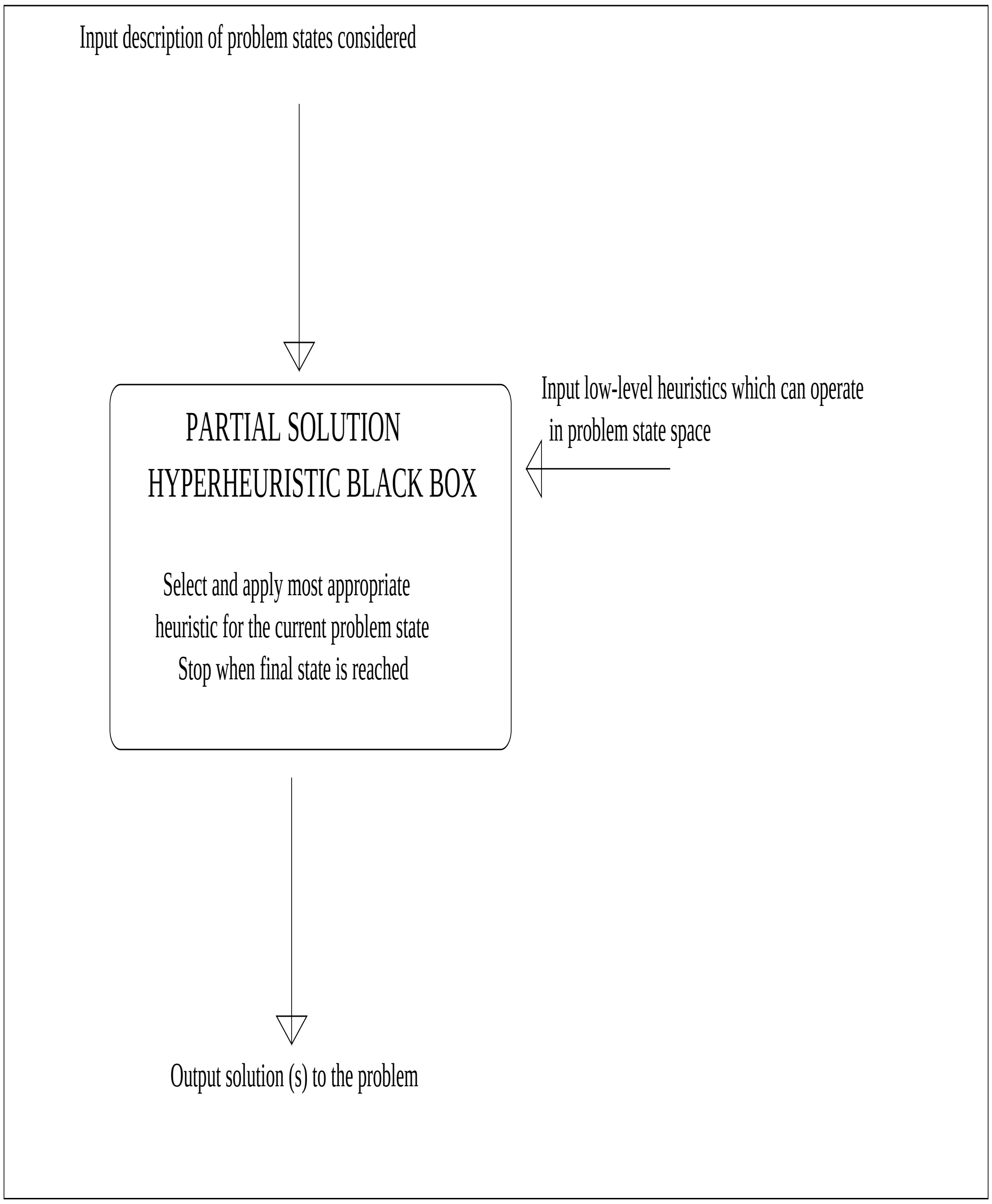

Input description of problem states considered
PARTIAL SOLUTION
HYPERHEURISTIC BLACK BOX
Select and apply most appropriate heuristic for the current problem state
Stop when final state is reached
Input low-level heuristics which can operate in problem state space
Output solution (s) to the problem

Figure 5.2: General hyperheuristic framework when dealing with partial solutions

(partial solution).

Until Final problem state.

Here, the hyperheuristic process starts from an initial problem state, which could be, for example, an empty solution. The final state corresponds to a complete solution to the problem. Therefore the hyperheuristic repeatedly selects and applies a low-level level heuristic to the current problem state. This process continues until the final state has been reached. Of course, the key here is to select the low-level heuristic that is in some sense the most suitable for the current problem state [48]. In order to identify which heuristics are most suitable for which problem states it is necessary to *train* the hyperheuristic using a training set of problem instances. Then a *test* set of problems can be used to assess the performance of the hyperheuristic on new problem instances. This type of hyperheuristic approach is therefore useful when dealing with a large number of problem instances. For example in [234], the authors had over 900 bin-packing problem instances available. Training and test sets of problem instances are also frequently used in AI planning systems [106, 132].

Hyperheuristics dealing with complete solutions

If there are not enough problem instances available for training and testing of the method, it might be more useful to consider hyperheuristics which deal with complete solutions - It should be mentioned that dealing with complete solutions will often mean dealing with 'almost' complete solutions. Note that this is the case in most metaheuristic implementations. Indeed, at the beginning of the search process, a complete solution is constructed. Then, through a series of improvements (local search) a final solution is obtained, which is (hopefully) of better quality than the initial solution. In this thesis we develop hyperheuristics which deal with (almost) complete solutions. A general hyperheuristic procedure in this case (i.e. when dealing with complete solutions) can be illustrated in the following pseudocode:

```
Do
    Select a low-level heuristic and apply it to the current (complete) solution.
Until Stopping condition is met.
```

Here, the hyperheuristic starts with an initial complete solution, which, for example, could be obtained using a constructive procedure. The constructive procedure starts with an empty solution and returns a complete solution which may or may not meet the problem's constraints (i.e. feasibility). The hyperheuristic can try to improve on the current solution using different local search operators (e.g. heuristics) and neighbourhood structures. The hyperheuristic repeatedly selects and applies a low-level level heuristic to the current complete solution. This process continues until a stopping condition has been met. The stopping condition must be input by the user. It can be given in terms of the number of iterations, the maximum amount of CPU time allowed, the maximum number of iterations or maximum amount of CPU time without consecutive improvement, when a solution has an objective value below (above) a certain threshold for a minimisation (maximisation) problem, etc. When the stopping condition has been met, the hyperheuristic process stops and returns the best solution(s) found during the search. Of course the output solution is, like the initial solution, complete. Note therefore that an initial solution must be input to the hyperheuristic black box. Figure 5.3 illustrates the framework of this type of hyperheuristic.

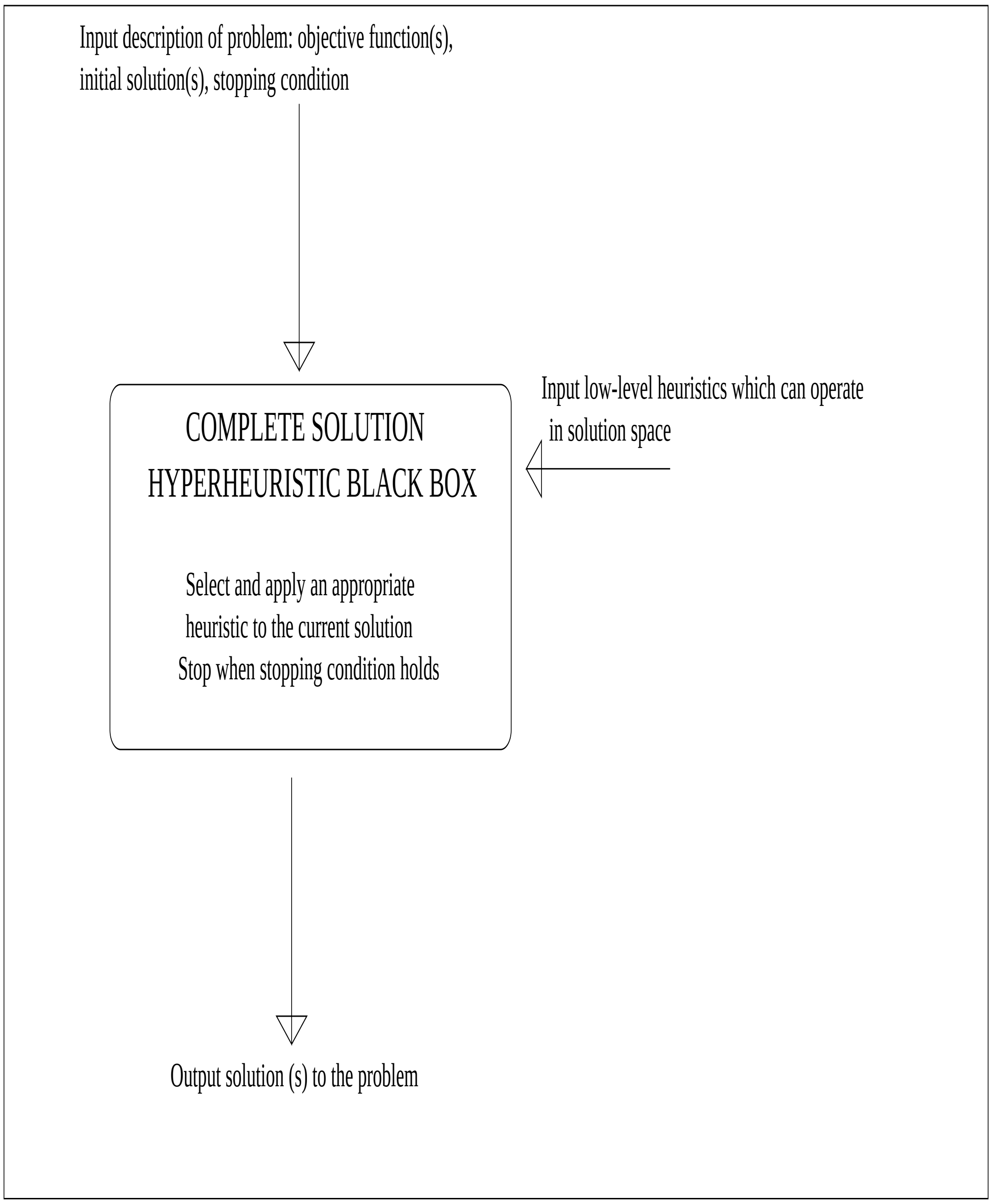

Input description of problem: objective function(s),
initial solution(s), stopping condition
COMPLETE SOLUTION
HYPERHEURISTIC BLACK BOX
Select and apply an appropriate
heuristic to the current solution
Stop when stopping condition holds
Input low-level heuristics which can operate
in solution space
Output solution (s) to the problem

Figure 5.3: General hyperheuristic framework when dealing with complete solutions

Remarks

1. It should be noted that while hyperheuristics which deal with partial solutions need to go through a training phase before being applied to new problems, hyperheuristics which deal with complete solution do not require such a training phase but they do require a constructive phase. Once an initial solution has been produced, the (complete solution) hyperheuristic is directly applied to whatever problem(s) is(are) at hand. The training in the case of a hyperheuristic which deals with (almost) complete solutions is *online* or *implicit* as opposed to the *offline* or *explicit* training required by a hyperheuristic which deals with partial solutions. When the hyperheuristic deals with complete solutions, the low-level heuristics used operate in the solution space. Whereas when the hyperheuristic deals with partial solutions, the low-level heuristics used operate in the space of problem states.

2. When considering hyperheuristics which deal with complete solutions, the hyperheuristic can switch from one low-level heuristic to another during the search process. Whereas when dealing with partial solutions, the time it takes to switch between heuristics can be longer. For example in [234], the hyperheuristic which deals with partial solutions considers the possibility of switching between heuristics not after an item has been placed in a bin, but after a bin has been filled (in their problem, this takes several items). In AI planning systems, the time it takes to consider a possible switch can be even longer. For example in [106, 132], the switch is only considered after an entire problem instance has been solved. This issue of decision point is further discussed in level 2 below.

3. Note that when dealing with partial solutions, it is not always appropriate to use the objective function during the hyperheuristic solution process. In

effect the objective function, which is used to assess the quality of complete solutions is not suitable for evaluating partial solutions. Since, essentially, the hyperheuristic is *constructing* a solution from scratch (using different low-level heuristics), there is no need to evaluate partial solutions obtained during the process using the objective function of a complete solution. Of course different ways of evaluating partial solutions can be used. For example, in Greedy Randomised Adaptive Search Procedures (GRASP), a *cost* is used which evaluates the state of partial solutions obtained when adding constructive elements in order to form a complete solution [231]. The objective function (which is for a complete solution) in this case is only needed at the end of the solution process in order to assess solution quality. The *cost* function used in GRASP evaluates partial solutions which do *not* have *all* the elements that make up a complete solution. Such solutions cannot be evaluated using the objective function for complete solutions because the objective function *assumes* that *all* the elements that make up a complete solutions are present. For example, in the sales summit scheduling problem of chapter 6 (which was briefly described in chapter 1), one of the objectives of the problem is to minimise the number of delegates scheduled to attend the summit. It is clear that if this objective of minimising the number of delegates was used to evaluate partial solutions during the execution of a hyperheuristic which deals with partial solutions (and in fact during any constructive solution process), the empty solution (which is a partial solution in which no delegate has been scheduled for meetings) would be optimal, whereas complete solutions would be far from optimality as they involve a certain number of delegates. This again highlights another difference with a complete solution hyperheuristic which does not construct a solution, but instead, *iteratively improves* on an initial solution (by using different low-level heuristics). Consequently, it is necessary to have some sort of (complete) evaluation function throughout the search process. A hyperheuristic which deals with partial solutions is a *constructive hyperheuristic*. The hyperheuristic methodology discussed in this thesis (which deals with complete solutions) is a *local search hyperheuristic*.

4. Whether the low-level heuristics operate in the solution space (complete solutions) or the space of problem states (partial solutions), the hyperheuristic operates in the space of heuristics as is the case in [250, 251]. In other words, whether the hyperheuristic is a *constructive* or a *local search* hyperheuristic, it operates in the *heuristic space*, not the solution space. Of course, most metaheuristic studies operate in the solution space.

5. Table 5.1 highlights the main conceptual differences between hyperheuristics which deal with partial solutions and those which deal with complete solutions. One of the important points made in the table is about the re-usability or generality of both types of hyperheuristic. Because partial solution hyperheuristics require training for each class of problem, they are less easily (readily) re-usable than complete solution hyperheuristics which can be self-adaptive. In effect, the nature of incremental solution construction gives rise to a large number of different problem states (as explained above). Heuristics that are adaptive to such problems would be difficult to design, and training techniques might therefore be useful.

6. It is possible to combine both types of hyperheuristics so that in a first phase a hyperheuristic which deals with partial solutions can be invoked in order to *construct* a solution from scratch. The solution thus obtained using the constructive hyperheuristic can then be used as the starting point of a second phase during which a hyperheuristic which deals with (almost) complete solu-

	Complete solutions	Partial solutions
Initial solution	constructed	usually empty
Training phase	No (implicit)	Yes (explicit)
Objective function	Yes	Other measures
Frequency of decision points	High	Low
Low-level heuristics	operate in solution space	operate in state space
Stopping condition	user-defined	(automatic) final state
Re-usability	Easy	Less (training required for each problem)

Table 5.1: Conceptual differences between hyperheuristics which deal with complete solutions and those which deal with partial solutions.

tions is employed to *improve* on the initial solution using different *local search* operators (e.g. low-level heuristics).

7. As already mentioned, the remainder of this thesis is concerned with complete solution hyperheuristics. We discuss below further design issues related to such hyperheuristics.

Solution acceptance criteria

Another issue which needs discussing concerns the criteria for accepting a solution. We distinguish two types of acceptance criteria.

- A solution may be accepted regardless of whether it is better or worse than the previous one (all moves, AM).
- A solution may be accepted only if it is better than the previous one (only improving, OI).

Note that OI can easily get stuck in local optima, due to the fact that it does not accept a solution that is worse than the current. We might therefore expect AM to discover promising areas more often than OI if given enough time. The resulting AM hyperheuristic should therefore produce better results than its OI variant, although this is not always guaranteed. In effect, accepting all moves (AM) may also hinder intensification of the search, if the moves it allows are too destructive. In such situations, OI, which maintains focus on restricted portions of the search space, will encourage moves which can produce better solutions. Of course, it is possible to use a mixture of AM and OI during the search. For example simulated annealing always accepts better moves (OI) and sometimes accepts worse moves (AM), though the AM moves which worsen the solution are accepted with a certain probability [142]. We also present a simulated annealing hyperheuristic at the end of this chapter which uses both OI and AM during its search.

Objective function(s)

It is also important to be aware of the number of objectives of the problem. If the problem has multiple objectives, is it possible to exploit this in order to produce even better hyperheuristic results? For example it is conceivable to imagine a hyperheuristic which selects an appropriate low-level heuristic in order to improve on one particular objective, rather than the overall objective function (possibly so as to generate more varied Pareto local optima). Or should all individual objectives simply be aggregated in one objective function? Indeed there is considerable scope for research here. There are already attempts to hybridise hyperheuristics and multi-objective Pareto optimisation techniques [53] based on results produced in Chapters 6, 7 and 8. Although this thesis is not concerned with the investigation of multi-objective techniques in hyperheuristics, we discuss how a particular type of hyperheuristic which we have developed can cater for multi-objective optimisation problems (section 5.3).

Point-based or population-based solution(s)

One option which might be worth considering is whether the hyperheuristic maintains a population of solutions to the application problem or not. It should be pointed out that point-based methods (i.e. one solution) are usually less time-consuming than population-based approaches. On the other hand, population-based methods can produce good quality solutions by combining good features taken from different individuals. In this thesis, we consider point-based local-search hyperheuristics for the following reasons.

- Population-based methods usually require a number of parameters which need tuning. These parameters include the size of the population (i.e. the number of individual solutions maintained at a time), crossover and mutation rates (if using these operators), solution representation[1] (including length of chromosome), reproduction scheme, selection of individuals for the next generation, etc. The definition of these parameters often involves extensive tuning with the resulting parameter settings being problem-specific [124]. This, of course, hinders the goal of raising the level of generality sought in this thesis.
- A number of population-based hyperheuristics have already been developed (e.g. [68, 87, 103, 135, 213, 259]). The research programme presented in this thesis is motivated (partly) by the desire to explore novel and untried research directions.
- Ideally a hyperheuristic method should be parameter-free, that is, all parameters are self-tuned and do not therefore require user specification. This goal motivated the choice of point-based methods.

[1]though this is needed for point-based methods as well

5.2.2 Level 1: Low-level heuristics

Which heuristics and how many?

The first question here is with regard to the size of the set of low-level heuristics used. How many heuristics should the hyperheuristic employ? If there are too few heuristics, it might be difficult to ascertain the benefit of using a hyperheuristic. Indeed the choice of heuristics in this case should not be too difficult, and it might be worth considering much simpler ways of selecting between a small number of heuristics. On the other hand, if there are too many low-level heuristics, the hyperheuristic may require too much time in order to learn how to select an appropriate heuristic. Experiments on different sizes of the set of low-level heuristics are carried out in chapter 6. Another important question is *which* heuristics to implement or to input to the hyperheuristic balck box (if already implemented). Of course the user can implement his / her own low-level heuristics and plug them into the hyperheuristic black box. Simple low-level heuristics (e.g. 'add', 'drop', 'swap' objects) can be easily implemented and input to the black box. The idea is that some heuristics come from existing (manual) solution approaches, together with a small number of 'add', 'drop', 'swap' etc. operators until there is enough richness so that good solutions can be reached from 'okay' solutions in only a small number of steps. Of course, low-level heuristics are problem-specific. It is also possible to develop systems which link to a maintained database of low-level heuristics, which could be dragged-and-dropped into the hyperheuristic black-box. Each heuristic in the database could be described in simple English terms so that the user can understand what they do. The idea of storing previously built heuristics in a database is not new. The machine learning paradigm of case-based reasoning has recently been used to select heuristics for course timetabling problems [51, 220]. In [51, 220], the case-based reasoning system maintains a case base of information regarding the performance of different heuristics on timetabling problems previously solved.

How to apply the chosen heuristics?

It is important to know how the chosen low-level heuristics are going to be applied. For example the chosen heuristic can be applied once. The heuristic could also be repetitively applied as long as it yields a better solution, that is, until it reaches a local optimum with regard to the corresponding neighbourhood (steepest descent). One way of controlling this is to apply a selected heuristic in a steepest descent fashion when exploiting the heuristic (high-level intensification) and in a single call fashion when exploring the space of heuristics (high-level diversification).

Low-level (meta)heuristics

Finally it should be pointed out that the low-level heuristics plugged ino the black box can be, themselves, metaheuristics. For example we could have a hyperheuristic which chooses between several variants of the same metaheuristics, e.g. different variants of a simulated annealing algorithm each with different parameters. The key for the hyperheuristic here would be parameter selection for each variant of the low-level simulated annealing algorithms. It is also perfectly conceivable to imagine a hyperheuristic system which decides to run, say a tabu-search approach for a certain duration and which then switches to a simulated annealing algorithm or a genetic algorithm and so on. However, it is not clear at all how we assess the contribution of each metaheuristic towards solution quality. The resulting system may be so complex that it is difficult to clearly identify which metaheuristic was most useful. In this thesis we consider the use of simple heuristics. The use of sophisticated metaheuristics at the low level counters the idea of developing cheap and easy-to-implement systems. The point here is that simple low-level heuristics pave the way to cheap and easy-to-implement systems.

It should be acknowledged however that this view is taken from an application standpoint. From a research point of view, hyperheuristics could be employed in order to compare different sophisticated metaheuristics. In fact, when significant progress is made in this area, it may be appropriate to investigate the intelligent selection of metaheuristics (or even low-level hyperheuristics!).

5.2.3 Level 2: hyperheuristic (high-level heuristic)

Learning mechanism

A key ingredient in implementing certain hyperheuristic approaches is the learning mechanism, which guides the hyperheuristic in the way in which low-level heuristics are selected. The survey of hyperheuristics (chapter 2) discussed the use of learning mechanisms. Different types of hyperheuristics will have different ways of selecting the low-level heuristics, e.g. genetic-algorithm hyperheuristics [68, 138], learning classifier systems [234], etc. In this thesis, we use a *choice function* as a learning mechanism for the hyperheuristic. Note that a number of hyperheuristics in which the low-level heuristics are chosen at random (i.e. such hyperheuristics have no learning) can be implemented. The point in using non-learning hyperheuristics is to compare them against hyperheuristics which are equipped with a learning mechanism. A comparison between simple hyperheuristics (without learning) and sophisticated hyperheuristics would help determine whether the incorporation of a learning mechanism in a hyperheuristic is beneficial. Both our simple and sophisticated choice function hyperheuristics will be presented in the next two sections.

An effective hyperheuristic learning mechanism must achieve an appropriate balance between the exploitation of the search experience gathered so far and the exploration of unvisited or relatively unexplored parts of the search space. There are several ways in which such a balance can be achieved, which comes down to when should different types of heuristics be applied? For example when dealing with local search hyperheuristics, it is clear that always selecting the best performing low-level heuristics (the ones that improve on the solution) will lead to local optima with respect to the corresponding neighbourhoods which may be of low quality[2]. It is therefore important to allow for a learning mechanism which selects 'bad' low-level heuristics at certain points during the hyperheuristic search in order to escape from the local optima and explore other areas of the search space. As will be explained below, the choice function hyperheuristic provides a way of addressing the balance between exploitation and exploration. Of course, randomisation is another means of exploration of different parts of the search space. This is also taken into account in our hyperheuristic methods which are presented in the next section.

There should be a clear distinction between the control over the execution of the low-level heuristics and the control over the way solutions are chosen (or neighbourhood moves). While in the former situation, the search is taking place in the space of heuristics, in the latter situation the search is taking place in the space of solutions to the problem. The hyperheuristic has no direct control over the solution space, within which low-level heuristics operate. The hyperheuristic only has direct control over the low-level heuristics. Consequently, the hyperheuristic has no direct intensification and diversification components as such. Instead, the hyperheuristic has exploitation and exploration components. Exploitation and exploration can be

[2]Getting quickly to good local optimum is often desirable.

viewed as being analogous to intensification and diversification respectively. However intensification and diversification do not take place in the solution space (as is the case with most metaheuristic approaches) but instead in the heuristic space.

Decision points

Another issue concerning the design of hyperheuristics is the determination of decision points. A decision point is a point at which a heuristic must be chosen, or in other words a point at which a heuristic trial takes place. This can be done in a number of ways. To remain general, a single trial of a low level heuristic can be defined as v trials where v is a positive integer. The discussion regarding decision points becomes therefore a discussion regarding the value of v. v can be constant throughout the hyperheuristic search. For example if $v = 1$ for the entire duration of the hyperheuristic solution process we are in presence of a hyperheuristic which repeatedly chooses a low-level heuristic and applies it once. Another possibility is to maintain a variable v. The value of v can be for example the number of steps it takes to apply a chosen low-level heuristic until no further improvement is possible (steepest descent). v can be chosen to be a large number, (e.g. 20000). This would be useful for example in situations where the chosen low-level heuristic operates in a very large neighbourhood of solutions. In such large neighbourhoods it is not easy to assess the performance of the chosen low-level heuristic in just a few trials. v can also be determined to be specific for each low-level heuristic and even in this case, each heuristic-specific v can be constant or variable. The value of v can be a function of a number of factors such as the amount of CPU time given to a particular heuristic or group of heuristics, if the solution produced has reached a certain threshold, if the change in the objective function value is within or outside a predefined interval, etc.

Again the number of possibilities is enormous. In this thesis we choose $v = k$ when exploiting the search and $v = 1$ when exploring the space of heuristics; where k is the number of steps (trials) necessary to reach a local optimum with respect to the neighbourhood of the chosen heuristic. This means that v is heuristic-specific. Also, it seems likely that the hyperheuristic prediction is more accurate with a smaller v as there will be more decision points, and therefore more statistical data points to consider.

Remark In the case where v is large there may be further issues to consider. For example the notion of change in the objective function value must be cleary defined. In effect, the change in the objective function value can be calculated as being the difference between

- best solution found by current trial and best solution to date
- best solution found by current trial and best solution found by previous trial
- mean solution value over current trial and mean solution value over previous trial
- mean value of solutions sampled in current trial and mean value of solutions sampled in previous trial
- mean solution value of accepted solutions in current trial and mean solution value of accepted solutions in previous trial.

Note that the third option differs from the fifth option in that the former is averaged over all v solutions whether or not they have been updated whereas the latter is based on trials resulting in an update.

Heuristic selection

Once the decision point has been determined, we must focus on the actual selection of the heuristics. Of course this is related to the learning mechanism. The question here is whether the hyperheuristic should systematically use the top low-level heuristic as suggested by the learning mechanism or not. Indeed the hyperheuristic may decide to choose another heuristic in the top d heuristics suggested by the learning mechanism. In this thesis we shall choose the top heuristic suggested by the choice function learning mechanism. This was suggested by Nareyek [211, 210]. The other reason for this choice is that it is difficult to ascertain the role played by the learning mechanism if its recommendations are not always followed by the hyperheuristic (this does not mean that a hyperheuristic which does not always follow the recommendations of the learning mechanism cannot perform well).

Remark

The problem of designing a hyperheuristic framework in which a high level heuristic controls a number of low-level heuristics, has a very large solution space, namely the set of all possible design choices at all three levels. Indeed, the number of possible ways in which to combine various choices of the design of a hyperheuristic as presented in Figure 5.1 is very large. Of course there are other problems, such as designing of aircrafts, computers and other complex civil engineering structures, that also have large design spaces. However for many of these other problems, it is possible to quickly evaluate candidate solutions (i.e. design choices) by means of extensive simulations and verification facilities. For example, it is possible to use strategies which 'generate and test' different candidate solutions proposed by humans or artificial expert systems. Unfortunately, such simulation and verification facilities are not available when designing hyperheuristic frameworks. It is therefore necessary to prune the space of design solutions, leaving a smaller and more easily searched sub-space [256]. The hyperheuristics presented in the next section constitute one such sub-space. The design issues discussed in this section highlight the large number of design choices possible.

5.2.4 Guidelines for designing hyperheuristics

We present below general guidelines for how to build a hyperheuristic.

1. The first thing to do when one is considering the development of a hyperheuristic is to carry out some sort of qualitative analysis of the issues at stake. Do we really need a hyperheuristic? What sort of problems will the hyperheuristic be applied to? How often do we have problems that need solving? What sort of solution quality do we need? How many problem instances are they? This is known as Soft OR or system dynamics and should take place before any quantitative study [233, 269]. At this stage there are no definite answers.

2. If the number of problem instances to be solved at one time (here, time should be taken in the broad sense - e.g. a day) is relatively little, (e.g. 1 to 100) it might not be worth developing a hyperheuristic which requires initial training. This is because the number of problem instances to be solved may not be large enough to allow for an effective learning during the training phase (i.e. not enough statistical data points). Also there may not be enough problem instances in the training set to justify the need for such training. If there is relatively a small number of problem instances to be solved it might be a good idea to consider local search hyperheuristics. For each of the few problems

to be solved, the hyperheuristic will iteratively attempt to solve the problem without an initial training phase. Because local search hyperheuristics such as the ones developed in this thesis do not require an initial training phase, they can be applied not only to different instances of the same problem but also to different problems.

3. If there is a large number of problem instances to be solved at one time (e.g hundreds or even thousands of problem instances) it might be very time consuming to apply a local search hyperheuristic to each of these problem instances. The problem with local search hyperheuristics which do not require an initial training phase is that the learning achieved during the solution process of one problem instance is not passed on to other problem instances, which results in the hyperheuristic having to learn again every time it is applied to a different problem instance (the hyperheuristic 'forgets' what it has learned at the end of the solution process for each given instance). This is not desirable when dealing with a very large number of problem instances. If there is a very large number of instances of the same problem it might be beneficial to consider a hyperheuristic which learns how to apply good heuristics during a training phase. The advantage here is that the learning takes place across all instances of the training set and what was learnt in one instance can be used in another. At the end of the training phase the hyperheuristic can be applied to a test set in order to assess its performance. When the hyperheuristic requires initial training, it might be useful to consider partial solutions as the key here is to match a given problem state with the most suitable low-level heuristic [48]. Since most real-world problems have a large (if not infinite) solution space, it is (often) impractical to consider describing all problem states in terms of complete solutions. The benefit of using partial solutions is that a point in

the problem state space corresponds to several points in the space of partial solutions. The disadvantage for hyperheuristics here is that an initial training phase is required for each problem.

4. Consider a modular description of your hyperheuristic (Divide and Conquer strategy). For example we can view the hyperheuristic system as being made of two modules. The first module contains the learning mechanism as well as information as to whether a single solution or a population of solutions is maintained. The learning mechanism tells the hyperheuristic which heuristic to choose at each decision point. The second module contains the low-level heuristics and the evaluation function, which tells us how solutions (whether partial or complete) are evaluated during the solution process. The first module is generic whereas the second module is problem-specific. This modular description allows us to see that once the learning mechanism is developed (i.e. the first module), one only needs to input low-level heuristics and an evaluation function. This would be all that is needed in order to apply a hyperheuristic to a given problem. The low-level heuristics are simple and easy-to-implement. This paves the way for a cheap hyperheuristic system which is easily re-usable across a wide range of problems.

5. Keep it simple where possible.

Of course these are guidelines, not rules. They can therefore be modified to suit specific needs and thus create different flavours of hyperheuristics (e.g. combining a constructive hyperheuristic with a local search hyperheuristic, incorporating problem-specific considerations in the learning mechanism module, using sophisticated metaheuristics as low-level heuristics, evolving a population of hyperheuristics, etc.). As can be seen the hyperheuristic designer can be as imaginative as they wish.

Solution(s)	complete or partial
	single or population
Heuristics	simple or sophisticated (metaheuristics)
	applied once or several times
	constructive or local search
Learning mechanism	search in heuristic space is single point or population based
	initial training phase or not

Table 5.2: A summary of key issues when designing a hyperheuristic

5.2.5 Summary

To conclude this section, we give in Table 5.2 below a summary of the key issues that need to be considered when designing a hyperheuristic. These issues have already been discussed above.

In the next section we present our own hyperheuristics. We choose a single-point approach as opposed to a population based approach because of the large number of parameters involved when one is dealing with a population of solution. We consider local search hyperheuristics which deal with complete solutions, as our hyperheuristics are designed to be applicable not just to one problem but to different problems. Hyperheuristics developed in this thesis will be applied to several instances of three different problems. The novelty of our hyperheuristics resides in a learning mechanism never developed before.

5.3 Hyperheuristics developed

With reference to issues summarised in Table 5.2, the hyperheuristics developed in this section are local search hyperheuristics which deal with complete solutions. The low-level heuristics are simple local search operators which operate in the solution space. Each low-level heuristic can be applied to the current solution either once or

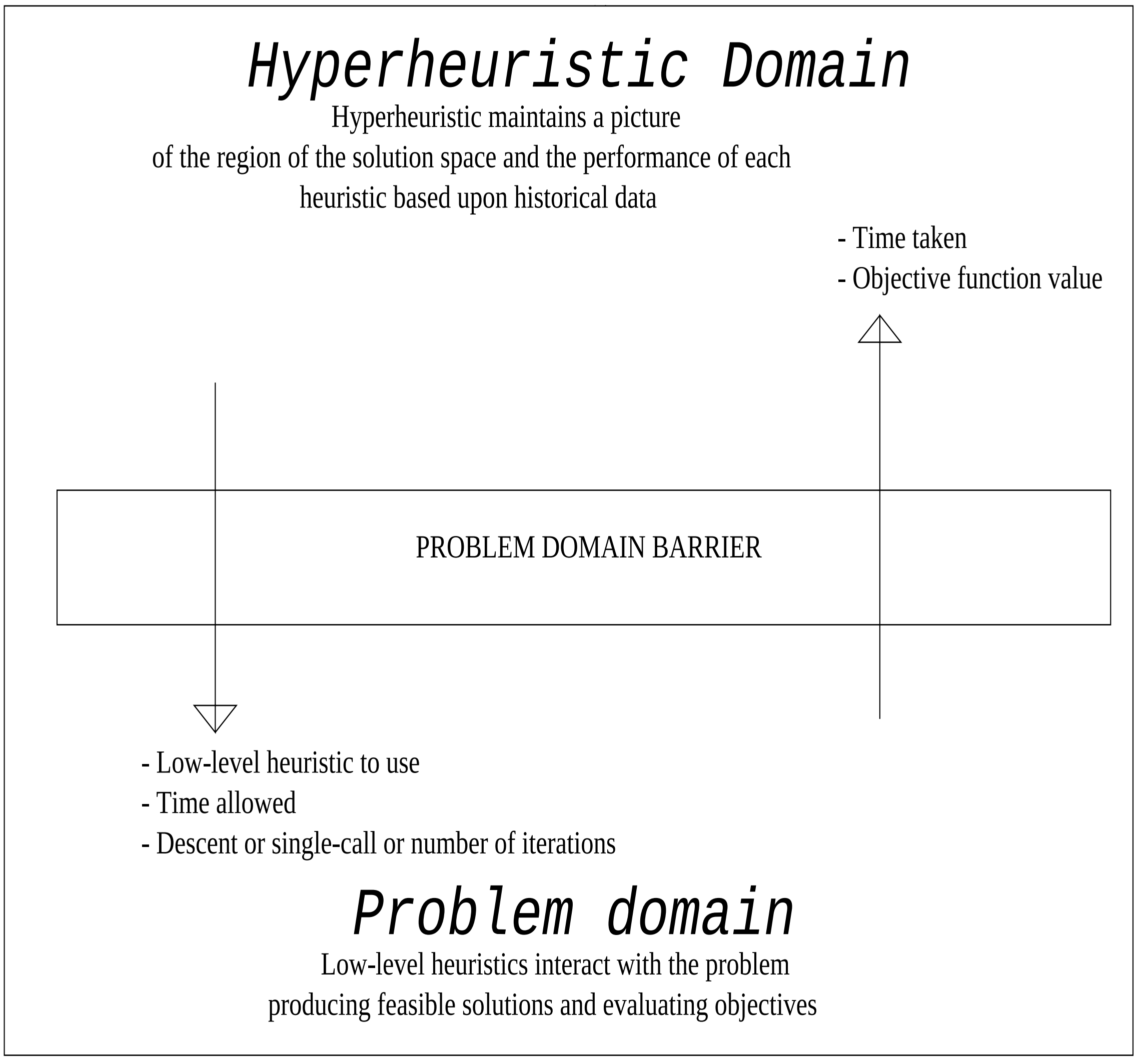

Hyperheuristic Domain
Hyperheuristic maintains a picture of the region of the solution space and the performance of each heuristic based upon historical data
- Time taken
- Objective function value
PROBLEM DOMAIN BARRIER
- Low-level heuristic to use
- Time allowed
- Descent or single-call or number of iterations
Problem domain
Low-level heuristics interact with the problem producing feasible solutions and evaluating objectives

Figure 5.4: The general framework of hyperheuristics developed

several times. Each low-level heuristic can modify the current solution and return a new one, whose objective function value can be better, worse, or of the same value as that of the previous solution. The decision point is taken to be the point immediately after a low-level heuristic has been applied. Our hyperheuristics are single-point methods (as opposed to population-based methods) which maintain one solution at a time both in the solution space and the heuristic space. We depict in Figure 5.4 the main features of the type of hyperheuristics which are developed.

Because the hyperheuristic does not have control over the way the solution is altered by the different low-level heuristics, it operates at a higher level of abstraction and generality than most current metaheuristic approaches. The process of choosing a low-level heuristic takes place dynamically so that at each decision point the hyperheuristic must choose which low-level heuristic to apply next. The hyperheuristic interacts with the low-level heuristics but only non problem-specific information such as CPU time and the change in the evaluation function passes between the two. Problem-specific information is prohibited from passing through the hyperheuristic/low-level heuristic interface as illustrated in Figure 5.4. This again is in view of developing a non-problem-specific hyperheuristic.

This is where the fundamental difference between the terms *hyperheuristic* and *metaheuristic* lies. A metaheuristic can (and usually does) use domain-specific knowledge in order to control the way a low-level heuristic modifies the solution. This limits the range of applicability of that particular metaheuristic but can result in excellent solution quality and possibly low computational times (e.g. [89, 9]). Since a hyperheuristic prohibits almost all problem-specific information from passing through the hyperheuristic/low-level heuristic interface, it is readily re-usable for other problems and domains if new low-level heuristics and objectives are supplied. The hyperheuristic is therefore a generic and easy / fast-to-implement method, which should produce solutions of acceptable or good quality, based on a set of simple low-level heuristics. Because relatively little problem-specific knowledge is used (which is contained in the low-level heuristics), the hyperheuristic may also be used in cases where little domain-knowledge is available (for instance when dealing with a new or poorly understood problem) or when a solution must be developed quickly (for example when prototyping). The hyperheuristic may manage a set of simple, knowledge-poor, low-level heuristics (such as ‘swap’, ‘add’ and ‘drop’ moves). In order for a hyperheuristic to be applicable to a given problem, all that is needed is a set of low-level heuristics and a formal means for evaluating solution quality (one or more objective functions). The hyperheuristic works by iteratively choosing a low-level heuristic to apply until some stopping criterion is met.

As mentioned in the previous section, the learning mechanism can be enabled or disabled. We first describe simple hyperheuristics in which the learning mechanism is disabled.

5.3.1 Simple hyperheuristics

General overview

Because there is no element of learning in this type of hyperheuristic, the only way one can choose a low-level heuristic is either at random or in a certain predefined sequence (e.g. in Variable Neighbourhood Search [137, 194]. VNS considers problem domain information.). Thus the hyperheuristic conducts a random search in the space of heuristics. The other issue discussed here is whether the chosen heuristic is applied once (single call) or repeatedly until it reaches a local optimum (with respect to the neighbourhood of solutions reachable using that heuristic). We consider below four variants of hyperheuristics which carry out a random search in the space of heuristics.

Variants considered

SimpleRandom (SR): This algorithm repeatedly chooses one low-level heuristic uniformly at random and applies it once. This process goes on until a stopping condition has been met. This is described in the following pseudocode.

```
Do
    Select a low-level heuristic uniformly at random and apply it once.
Until Stopping condition is met.
```

RandomDescent (RD): This algorithm repeatedly chooses one low-level heuristic uniformly at random, then continues to apply it until no further improvement is possible. This process continues until a stopping condition has been met. This is described in the following pseudocode.

```
Do
    Select a low-level heuristic uniformly at random and apply it in a steepest
    descent fashion.
Until Stopping condition is met.
```

RandomPerm (RP): This algorithm chooses a random permutation of all the low-level heuristics and applies each low-level heuristic once in the chosen order. It cycles round from the last low-level heuristic in the permutation to the first one. This process goes on until a stopping condition has been fulfilled. This is described in the following pseudocode.

```
Create a random permutation of all low-level heuristics available.
Do
    Select the next low-level heuristic in the sequence and apply it once.
```

```
Until Stopping condition is met.
```

RandomPermDescent (RPD): This algorithm does the same thing as *RP* but each low-level heuristic is applied repeatedly until we reach a local optimum for that low-level heuristic. This is similar to variable neighbourhood descent [137, 194]. *RandomPermDescent* is described in the following pseudocode.

```
Create a random permutation of all low-level heuristics available
Do
    Select the next low-level heuristic in the sequence and apply it in a steepest
    descent fashion.
Until Stopping condition is met.
```

As mentioned in the previous section, these simple hyperheuristics serve as a means of comparison against sophisticated hyperheuristics which are equipped with some form of learning mechanism. In order for a sophisticated hyperheuristic to be effective, it should be able to produce solutions that are at least as good as those obtained using a simple hyperheuristic.

5.3.2 A choice function hyperheuristic

Having discussed and developed simple hyperheuristics, we now consider a hyperheuristic in which the learning mechanism is enabled. Different heuristics have different performances on different solutions and different parts of the solution space. Indeed, since different heuristics have different strengths and weaknesses, it makes sense to see whether they can be combined in some way so that the strengths of one heuristic compensates for the weaknesses of another [48]. The hyperheuristic tries to co-ordinate this combination of heuristic based on the guidance of a *choice function*. The choice function will be used to rank low-level heuristics.

We are looking for an effective way in which to choose an appropriate heuristic at each decision point. There may be difficulties connected with both the nature of the solution space and the individual characteristics of the available low-level heuristics. However, we do not want to employ problem-specific information. The hyperheuristic is not designed with any particular problem in mind. The aim is to raise the level of generality at which current optimisation systems operate. If we somehow knew that a given heuristic will perform well, we should *exploit* this. That heuristic should be selected and applied. Sometimes, however, it might be useful to *explore* the space of heuristics by selecting a heuristic which will not necessarily perform well but will help explore other regions of the search space. In the absence of knowledge about future heuristic performance, we make use of statistical prediction. Of course, if we knew which heuristic(s) performs well, hyperheuristics would no longer be relevant. The sort of information which we can use in order to predict the performance of the low-level heuristics can include historical (statistical) data, regarding the quality of solutions obtained, the amount of CPU time used, the time elapsed since a given heuristic was last called etc, none of these are specific to any problem.

Thus the choice function ranks the low-level heuristics on the basis of forecast of future performance. The choice function attempts to capture the correspondence between the region of the solution space currently being investigated and the historical performance of each low-level heuristic. As in machine learning, the choice function exercises two major roles in the guidance which it provides to the hyperheuristic: *exploitation* (level 2 intensification) and *exploration* (level 2 diversification).

Exploitation (level 2 intensification)

Exploitation is maintained by collating information regarding both individual and collective performance of the low-level heuristics. The intuitive idea here is that at each decision point, the choice of a given low-level heuristic, say N_j, may be motivated by the following observations:

- N_j may yield an improvement when applied alone (individual performance),
- N_j might not yield an improvement, but it may help another low-level heuristic (or a collection of low-level heuristics) to yield an improvement (joint performance),

Individual performance for heuristic N_j is expressed in the following function:

$$f_1(N_j) = \sum_n \alpha^{n-1} \left(\frac{I_n(N_j)}{T_n(N_j)} \right) \tag{5.1}$$

where $I_n(N_j)$ (respectively $T_n(N_j)$) is the change in the evaluation function (respectively the amount of CPU time taken) the n^{th} last time heuristic N_j was called, and α is a parameter between 0 and 1, which reflects the importance attached to recent performance. After calling heuristic N_j, the new value $f_1^{new}(N_j)$ can be quickly calculated from the previous value $f_1^{old}(N_j)$ using the following iterative formula

$$f_1^{new}(N_j) = \frac{I_1(N_j)}{T_1(N_j)} + \alpha f_1^{old}(N_j). \tag{5.2}$$

Collective performance for heuristic N_j is expressed in the following series of functions.

For a pair of heuristics

$$f_2(N_k, N_j) = \sum_n \beta^{n-1} \left(\frac{I_n(N_k, N_j)}{T_n(N_k, N_j)} \right) \tag{5.3}$$

where $I_n(N_k, N_j)$ (respectively $T_n(N_k, N_j)$) is the change in the evaluation function (respectively amount of CPU time taken) the n^{th} last time heuristic N_j was called immediately after heuristic N_k and β is a parameter between 0 and 1, which again reflects the greater importance attached to recent performance. If N_j has just been called after N_k, then the new value $f_2^{new}(N_k, N_j)$ can be quickly calculated from the previous value $f_2^{old}(N_k, N_j)$ using the iterative formula

$$f_2^{new}(N_k, N_j) = \frac{I_1(N_k, N_j)}{T_1(N_k, N_j)} + \beta f_2^{old}(N_k, N_j). \tag{5.4}$$

These ideas may be generalised to larger tuples and may take into account the time elapsed between calls to particular low-level heuristics. Both f_1 and f_2 aim to exploit the search experience gathered so far. The idea behind the expressions of f_1 and f_2 is analogous to the exponential smoothing forecast of their performance [274]. If a given low-level heuristic has been performing well, it might perform well if called again.

Exploration (level 2 diversification)

Exploration can be maintained by monitoring the amount of time elapsed since each low-level heuristic was last called. The intuitive idea here is that if we always select the heuristics which have been performing well, we might get stuck in a poor local optimum. In order to escape from a local optimum we can select a heuristic that has not been called recently. Of course, we do not expect that heuristic to improve on the current solution (if it does, so much the better - indeed there is a way to adjust the learning in order to reflect this - this will be discussed below). The idea of exploration can be expressed in a function, f_3, which provides an element of (level 2) diversification, by favouring those low-level heuristics that have not recently been used. Then we have

$$f_3(N_j) = \tau(N_j) \tag{5.5}$$

where $\tau(N_j)$ is the number of seconds of CPU time which have elapsed since heuristic N_j was last called.

The resulting choice function

The resulting choice function can be obtained by simply putting together the exploitation and the exploration functions. If the low-level heuristic just called was N_k then for any low-level heuristic N_j, the choice function f of N_j can be defined as

$$f(N_j) = \alpha f_1(N_j) + \beta f_2(N_k, N_j) + \delta f_3(N_j). \tag{5.6}$$

Which means:

$CHOICE \quad FUNCTION = EXPLOITATION + EXPLORATION$

We rank low-level heuristics using the choice function.

Catering for multiple objectives In the above expression, the choice function attempts to predict the overall performance of each low-level heuristic, that is, the effect of each low-level heuristic on the objective function. When the problem has several objectives, we may consider a variant of choice function f which separately predicts the performance of each low-level heuristic with respect to each objective. Often the objective function (or any other formal means of evaluating solution quality) of an optimisation problem, particularly one of personnel scheduling, is made of several factors [19, 20, 23, 83] such as individual preferences, labour costs, etc. The choice function can be redefined with respect to each of these criteria (objectives). Thus, for each individual objective l, the choice function with respect to l is

$$\forall l, \quad f_l(N_j) = \alpha_l f_{1l}(N_j) + \beta_l f_{2l}(N_k, N_j) + \frac{\delta}{c} f_3(N_j) \tag{5.7}$$

Which means:

$$CHOICE \quad FUNCTION_l = EXPLOITATION_l + EXPLORATION,$$

with c the number of individual objectives. $f_{1l}(N_j)$ is calculated by replacing $I_n(N_j)$ with $I_{nl}(N_j)$ in the expression of $f_1(N_j)$ in equation (5.1) where $I_{nl}(N_j)$ is the first order improvement with respect to criterion $l \in \mathbf{L}$. Similarly $f_{2l}(N_k, N_j)$ is calculated by replacing $I_n(N_k, N_j)$ with $I_{nl}(N_k, N_j)$ in the expression of $f_2(N_k, N_j)$ in equation (5.3) where $I_{nl}(N_k, N_j)$ is the second order improvement with respect to criterion $l \in \mathbf{L}$.

Of course, for a problem with multiple objectives, the individual choice functions with respect to each individual objective can be regrouped (aggregated) into one single choice function such as that in equation (5.6). The relationship between choice function f and individual choice functions f_l's is illustrated as follows:

$$f(N_j) = \sum_{l \in \mathbf{L}} f_l(N_j) = \sum_{l \in \mathbf{L}} \left[\alpha_l f_{1l}(N_j) + \beta_l f_{2l}(N_k, N_j) + \frac{\delta}{c} f_3(N_j) \right] \tag{5.8}$$

where $\mathbf{L}$ is the set of the evaluation function criteria, c the cardinality of set $\mathbf{L}$.

Consequently, when the problem has several objectives, it is possible to guide the hyperheuristic search to reflect this. The search is no longer conducted as if there was only one objective, but instead with respect to each individual objective. The search for a good solution can then be viewed as a search for a solution that is good with respect to each single objective in $\mathbf{L}$. Thus when searching for a good solution regarding a criterion $l \in \mathbf{L}$, we shall use the corresponding choice function f_l and the low-level heuristic for which f_l is maximum will be selected and applied.

Of course, even when the problem has multiple objectives, it is still possible to apply the choice function hyperheuristic as if we were dealing with a single objective problem. In this case, equation (5.8) is used, which aggregates all individual objectives into one single choice function. This is known as the *a priori* approach in multi-objective optimisation [190], in which the weight of each individual objective is assigned before solving the multi-objective optimisation problem. The other alternative is the use of *a posteriori* approaches which include Pareto optimisation techniques [64, 190]. It should be noted that equation (5.7) of the choice function allows for the use of Pareto optimisation techniques. The main goal of multi-objective optimisation is to find solutions that represent a good compromise between the various criteria or objectives (some of them conflicting) used to evaluate solution quality. A solution x is said to be non-dominated with respect to a set of solutions S if there is no other solution in set S that is as good as x in all objectives and better than x in at least one of the objectives [65]. The Pareto optimal front is the set of non-dominated solutions with respect to the whole solution space. The choice function in equation (5.7) provides a means to search for non-dominated solutions because in (5.7), the search is conducted with respect to individual objectives. For example [53] uses this result to implement a hyperheuristic which employs choice function (5.7) in a Pareto optimisation framework. This would not be possible with the choice function of equation (5.6) or (5.8) in which the hyperheuristic search is conducted as though there is only one objective. Again this thesis is not specifically concerned with the use of hyperheuristics for Pareto optimisation.

Exploitation and exploration: a self-adaptive procedure

The issue of self-adaptive hyperheuristics was discussed in section 5.2.1. It is desirable to have a self-adaptive hyperheuristic. In effect, if the hyperheuristic is self-adaptive, it would be able to adjust itself to the conditions of the environment it is operating in (e.g. heuristic space, solution space). This would enhance the generality and robustness of the hyperheuristic. In addition, the hyperheuristic would no longer require parameter specification from the user. One way to achieve self-adaptiveness in the choice function hyperheuristic is to maintain an adaptive ranking of the heuristics. Rather than having a constant expression of the choice function whose parameters remain constant during the search as was the case in [69] (the choice function parameters had to be manually tuned at the beginning of the search), we developed a procedure in [70] that adaptively adjusts the choice function parameters α_l, β_l and δ for each criterion l.

Let low-level heuristic N_j be the selected heuristic, for which f_l is maximum. Before applying heuristic N_j, we check, for criterion l, which of its choice function factors $\alpha_l f_{1l}(N_j), \beta_l f_{2l}(N_k, N_j)$ and $\frac{\delta}{c} f_3(N_j)$ is maximum. We call that element the biggest contributor, G_l, in choice function f_l. We shall use I_l to refer to both $I_l(N_j)$ and $I_l(N_k, N_j)$ and T to refer to both $T_{1l}(N_j)$ and $T_{1l}(N_k, N_j)$ as appropriate. The procedure works as follows.

1. *If $G_l = \alpha_l f_{1l}$ then apply low-level heuristic N_j, note the resulting change, I_l, and change parameter α_l so that $\alpha_l = \alpha_l(1+\epsilon)$, where ϵ is a small number having the same sign as I_l*
2. *If $G_l = \beta_l f_{2l}$ then apply low-level heuristic N_j, note the resulting change, I_l, and change parameter β_l so that $\beta_l = \beta_l(1+\epsilon)$, where ϵ is a small number having the same sign as I_l*
3. *If $G_l = \frac{\delta}{c} f_3$ then apply the low-level heuristic, say N_i, that maximises $\alpha_l f_{1l} + \beta_l f_{2l}$. If this produced a solution better than the previous then decrease δ by a certain positive quantity q so that $\delta = \delta - q$. Otherwise return to the previous solution (i.e. undo the application of heuristic N_i), keep δ unchanged and apply heuristic N_k;*
4. *In case of tie we apply heuristic N_j.*
5. *If there has been no improvement after a certain number of iterations augment δ by a certain positive quantity p so that $\delta = \delta + p$.*

We may also use the procedure for a single objective function, where $l = 1$.

The procedure described above allows interplay between all factors of the choice function. Although we know that each factor is important for choosing the right low-level heuristic, we have no idea as to how important each factor is relative to the others. The procedure adaptively adjusts the values of the different parameters so that, in light of the observed performance of each low-level heuristic, the weighting assigned to each factor is modified in an appropriate manner.

More precisely, in cases 1 and 2, the idea is to increase the corresponding parameter when the improvement is positive and to decrease it when the improvement is negative or null. These ideas are borrowed from reinforcement learning [154, 254] and have been applied by other OR and AI researchers [211, 210]. By increasing/decreasing α_l (β_l) in case 1 (2) we increase/decrease the degree of confidence that we place in the choice of those low-level heuristics that have shown a first-order (second-order) improvement/non-improvement. When I_l is positive the reward is there to encourage the emergence of 'good' low-level heuristics from the group. When I_l is negative or null the penalty is there to ensure that 'bad' low-level heuristics are not chosen often, thus really reinforcing the predictions. If f_1 (f_2) is the strongest predictor and it is a good choice of low-level heuristics, reinforce f_1 (f_2); else reduce its significance. However when there is a zero improvement we do not want to give a large penalty (as a null improvement is still better than a negative one). We choose $\epsilon = \frac{T}{n_h^2 freq}$ if $I_l = 0$, where $freq$ is the number of times heuristic N_j has been called and n_h is the total number of low-level heuristics, so that the penalty is proportional to the time 'wasted' in calling the heuristic and is smaller if $freq$ is larger - if $freq$ is larger it is presumably because the heuristic concerned has been performing well. if $I_l \neq 0$ we choose $\epsilon = \frac{I_l}{n_h E_0}$, where E_0 = evaluation of initial solution. This allows the α and β parameters to increase as we grow more confident in our forecast, and decrease when we cannot find an improved solution.

In case 3, the choice function suggests that an exploration move be made and proposes a low-level heuristic for this purpose. As this may not be the case, we 'question' the choice function's suggestion by applying a test low-level heuristic, the one for which $\alpha_l f_{1l} + \beta_l f_{2l}$ is maximum. If applying the test heuristic yielded an improvement, we decrease the value of δ which turned out to have been too large (exploration was suggested too soon). Otherwise we apply an exploration move (as suggested by the choice function) and the value of δ was appropriate, and is not changed. In case 4, we simply apply heuristic N_j as the appropriateness of exploitation or exploration is not clear. In order to change parameter δ to a value that provides an appropriate level of exploration, we need an effective expression for p and q. For example we can choose q as follows. Denote by N_1 the (test) heuristic that maximises $\alpha_l f_{1l} + \beta_l f_{2l}$ and N_2 the heuristic N_j that is due (for which both f_l and f_3 are maximum). What we wanted is N_1 to be the due heuristic in place of N_2, so we would wish that $f_l(N_1) > f_l(N_2)$. We are then looking for a q such that $\alpha_l f_{1l}(N_1) + \beta_l f_{2l}(N_1) + \frac{\delta - q}{c} f_3(N_1) > \alpha_l f_{1l}(N_2) + \beta_l f_{2l}(N_2) + \frac{\delta - q}{c} f_3(N_2)$, which gives $q > c\frac{f_l(N_2)-f_l(N_1)}{f_3(N_2)-f_3(N_1)}$. In practice we choose $q = c\frac{f_l(N_2)-f_l(N_1)}{f_3(N_2)-f_3(N_1)} + \nu$, where ν is a small positive number. Similarly we can choose p as follows. Denote by N_1 the heuristic applied without improvement and N_2 the heuristic for which f_3 is maximum at that time. What we wanted is that N_2 should have been called in place of N_1, so we would wish that $f_l(N_2) > f_l(N_1)$. We are then looking for a p such that $\alpha_l f_{1l}(N_2) + \beta_l f_{2l}(N_2) + \frac{\delta + p}{c} f_3(N_2) > \alpha_l f_{1l}(N_1) + \beta_l f_{2l}(N_1) + \frac{\delta + p}{c} f_3(N_1)$, which gives $p > c\frac{f_l(N_1)-f_l(N_2)}{f_3(N_2)-f_3(N_1)}$. In practice we choose $p = c\frac{f_l(N_1)-f_l(N_2)}{f_3(N_2)-f_3(N_1)} + \nu$, where ν is a small positive number.

Remarks

1. There are various ways in which run statistics could be used when evaluating the performance of the low-level heuristics. For example instead of cumulatively calculating the improvement on the objective function value in an exponential smoothing fashion [274], one could simply calculate the sum of all improvements, the average improvement, the maximum improvement, the best improvement, etc. over the past v trials for each low-level heuristic. Even with this, one needs to define parameter v. For example v can be set to 1 if

we are only interested in the most recently obtained statistics. At the other extreme, v can also be set to be equal to $freq$ if we are interested in collecting statistics from the beginning of the run to date. v can also be chosen to be heuristic-specific, so as to collect, for example, recent statistics for well-performing heuristics and less recent statistics for badly-performing ones; or vice versa. In addition it might be possible to use other statistical regression methods such as the moving average technique (here we regard the choice function values produced by each low-level heuristic over time as a time series)[274].

2. The choice function, which evaluates the performance of each low-level heuristics, takes into account the change I in the objective function value from the previous solution to the new solution. The magnitude of I may vary greatly during the execution of the hyperheuristic on one given problem. The magnitude of I may vary even more greatly across different application domains. The choice function, which is expressed as a function of I, should be able to cope with changing magnitudes if it is to be effective. This is taken care of by the way in which parameters α, β and δ of the choice function are calculated. Both α and β are normalised and are only allowed to vary within interval [0, 1] (e.g. if the values of α or β becomes negative, it is set to 0.001). The initial values for each of α, β and δ is also randomly chosen within $]0, 1[$. Then, during the solution process, δ is changed so as to reflect how urgent the desire to explore the search is. In effect, from the expression $f = \alpha f_1 + \beta f_2 + \delta f_3$ of the choice function f it can be deducted that $\delta = \frac{f - \alpha f_1 - \beta f_2}{f_3}$; which highlights the fact that the value of δ depends on those of α and β. This implies that the magnitude of α and β is reflected in the way in which δ is computed. No matter what the magnitude of α and β is, δ is increased and decreased by a certain quantity p and q respectively whose values incorporate that magnitude

of α and β, thus keeping parameter values in proportion. It can thus be seen in the next three chapters that our choice function is indeed able to cope well with different magnitudes, both during the solution process within each application problem and across different application domains. Of course, the initial values of parameters α, β and δ - which are randomly chosen - are not necessarily appropriate. As will be shown in chapter 6, the adaptive procedure takes care of adjusting the values of α, β and δ in an appropriate manner throughout the hyperheuristic search.

3. In light of early discussions in section 5.2.1, it would seem likely that constructive heuristics would not be suitable for our choice function hyperheuristic framework, which is a local search hyperheuristic. In effect, our hyperheuristic deals with complete solutions, whereas constructive heuristics deal with partial solutions. If the low-level heuristics are local search based heuristics, they would be suitable for the current hyperheuristic framework. For example, in chapters 6 and 8, different sets of local search low-level heuristics are used in the hyperheuristic framework.

Schematic view of the choice function hyperheuristic

In Figure 5.5, we give a schematic flow chart of the choice function hyperheuristic. The hyperheuristic is described for any objective l, for a minimisation problem. We therefore adopt the following notations which drops index l. We also use C++ conventions.

- I is the *relative* improvement, that is, the change in the evaluation function value from the previous solution to the new solution obtained by applying the selected heuristic (whether in a single-call or a steepest descent fashion).

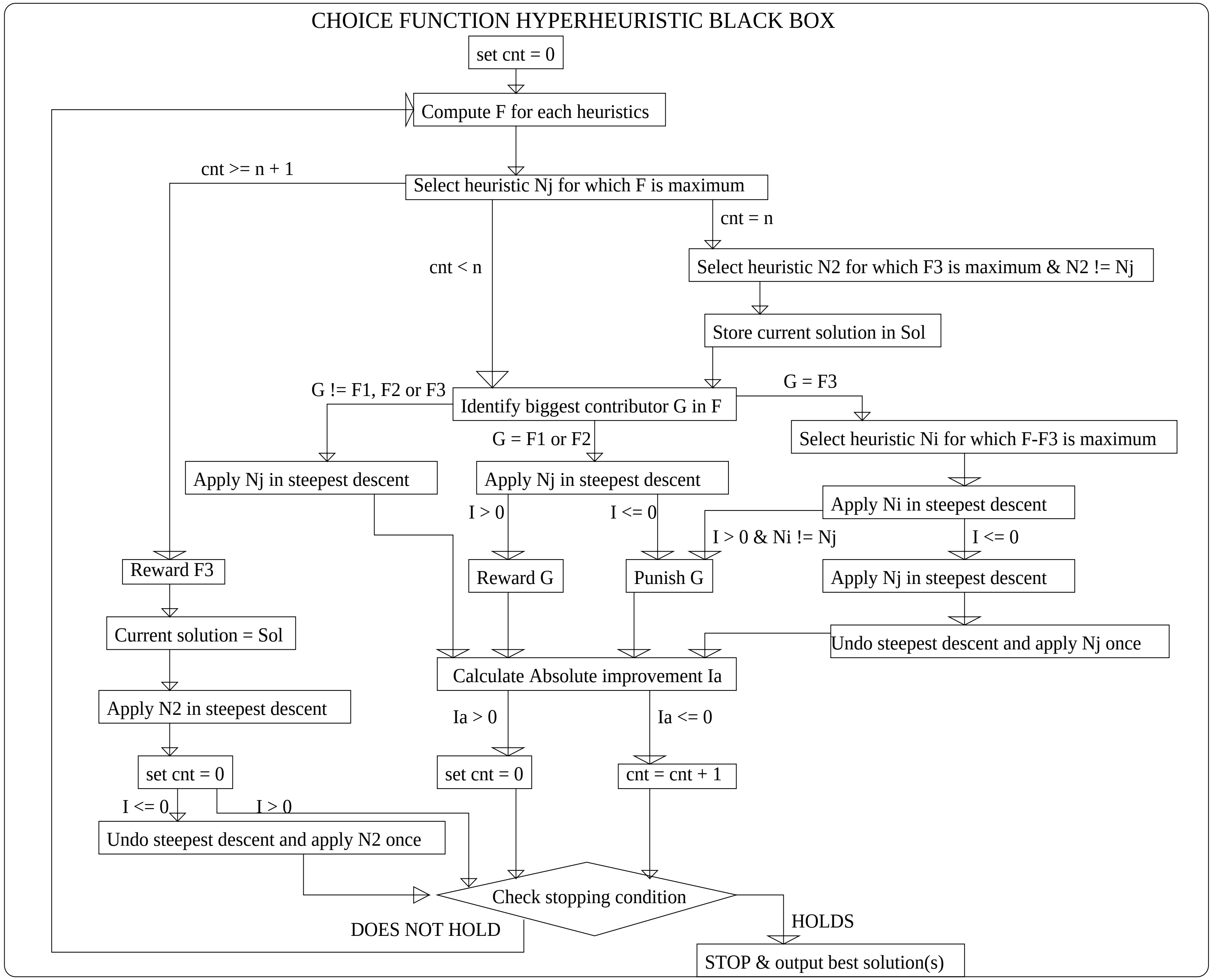

CHOICE FUNCTION HYPERHEURISTIC BLACK BOX
set cnt = 0
Compute F for each heuristics
cnt >= n + 1
Select heuristic Nj for which F is maximum
cnt = n
cnt < n
Select heuristic N2 for which F3 is maximum & N2 != Nj
Store current solution in Sol
G != F1, F2 or F3
Identify biggest contributor G in F
G = F3
G = F1 or F2
Select heuristic Ni for which F-F3 is maximum
Apply Nj in steepest descent
Apply Nj in steepest descent
Apply Ni in steepest descent
I > 0
I <= 0
I > 0 & Ni != Nj
I <= 0
Reward F3
Reward G
Punish G
Apply Nj in steepest descent
Current solution = Sol
Undo steepest descent and apply Nj once
Calculate Absolute improvement Ia
Apply N2 in steepest descent
Ia > 0
Ia <= 0
set cnt = 0
set cnt = 0
cnt = cnt + 1
I <= 0
I > 0
Undo steepest descent and apply N2 once
Check stopping condition
HOLDS
DOES NOT HOLD
STOP & output best solution(s)

Figure 5.5: Choice function hyperheuristic framework

Therefore a positive I means that the new solution is better than the previous (minimisation problem).

- Ia is the *absolute* improvement, that is, the change in the evaluation function value from the best solution so far to the new solution obtained by applying the selected heuristic (whether in a single-call or a steepest descent fashion). Therefore a positive Ia means that the new solution is better than the best solution found so far (minimisation problem).
- G is the biggest contributor in the choice function.
- We denote $F1 = \alpha f_1$, $F2 = \beta f_2$, $F2 = \delta f_3$ and $f = F = F1 + F2 + F3$.
- *cnt* is an integer variable which counts the number of consecutive *absolute* non-improvements.
- n is the number of low-level heuristics that are made available to the hyper-heuristic.

At the beginning of the search, *cnt* is initialised to 0. We then compute the choice function value for each heuristic, so as to select the one with the highest F value. In order to decide whether we are at an exploitation or an exploration stage, we determine G. This dictates the way we apply the chosen heuristic. More precisely the chosen heuristic is applied in a steepest descent fashion if we are in an exploitation phase (i.e. $G = F1$ or $G = F2$). We apply the chosen heuristic once (single call) if $G = F3$. If there are ties we apply the chosen heuristic in a steepest descent fashion. During an exploitation phase we reward G if the resulting solution is better than the previous solution (i.e. $I > 0$) and punish G otherwise. We know that the chosen heuristic should be applied once if $G = F3$. However, before doing so, we apply it in a steepest descent fashion. If this yields a positive improvement, then we punish G. Otherwise, we return to the previous solution and apply the chosen heuristic once. Rewarding $F3$ takes places only if there has been more than n consecutive absolute non-improvements. We therefore need to remember the solution (Sol) obtained at the end of the n^{th} consecutive absolute non-improvement.

Of course, there are several ways in which 'Reward' and 'Punishment' for $F1$, $F2$ and $F3$ in Figure 5.5 can be implemented. The way in which this is done above is just a possibility. The user can be as imaginative as they please. For example rather than having a reward system which is linearly proportional to I, we could consider non-linear schemes (e.g. instead of $\alpha_l = \alpha_l(1+\epsilon)$, we could have $\alpha_l = \alpha_l^{(1+\epsilon)}$ and instead of $\beta_l = \beta_l(1+\epsilon)$, we could have $\beta_l = \beta_l^{(1+\epsilon)}$; where ϵ can be a negative or positive constant, a function of I, etc.) [210, 211]. We could also consider a simple scheme in which the reward / punishment is increased by a constant value, regardless of the magnitude of I.

It should be noted that the sort of information utilised by the choice function hyperheuristic is not specific to any particular problem. The hyperheuristic has no knowledge of the application domain. It has no knowledge as to the purpose or function of each low-level heuristic. It has no knowledge of the application problem. The motivation behind this is that once the hyperheuristic has been developed, then new problem domains can be tackled by only having to replace the set of low-level heuristics and the evaluation function. This is exactly what is being done in chapters 6, 7 and 8, where the choice function hyperheuristic is being applied to three very different problems.

Using its internal state, the hyperheuristic has to decide which low-level heuristic should be applied next. Should it call the heuristic which produced the largest relative improvement? Should it call the heuristic which produced the largest absolute improvement? Should it call the heuristic that runs fastest? Should it call the heuristic that has not been called for the longest amount of time? The choice function expression proposed above attempts to maintain a balance between all these factors.

To make this process of changing the set of heuristics easy, we have developed an interface between the hyperheuristic and the low-level heuristics. There are further benefits in having this interface.

1. The interface allows the hyperheuristic to communicate with all low-level heuristics in the same way. Otherwise it would need a separate interface for each low-level heuristic.
2. The interface prevents domain-specific information from reaching the hyperheuristic. For example in the above implementation of the choice function (see Figure 5.4) only non problem-specific information such as CPU seconds, objective function value are allowed to pass through the interface. For example we have added a component in the interface which allows the user to 'tell' a low-level heuristic how long it can run. Thus the hyperheuristic can call each heuristic in turn giving it a specified amount of CPU time and the heuristic that produced the best result within the allowed CPU time is the one that is applied to the current solution. The hyperheuristic can perform a series of tests, by applying each heuristic in order to find out how well they would each perform. Then the hyperheuristic can decide to select a subset of those heuristics which performed well for example. So there is scope for further variations of the current hyperheuristic implementation.
3. The interface allows for rapid prototyping for other domains [73]. When solving a new problem, the user has to supply (plug into the black box) a set of low-

level heuristics and a suitable evaluation function to assess solution quality. If the low-level heuristics follow a standard interface the hyperheuristic need not be altered whatsoever. The hyperheuristic is readily applicable to the new problem. Again the aim is to raise the hyperheuristic to a higher level of abstraction than current metaheuristic approaches.

4. If the user is questioning whether a heuristic is any good or not, they can simply throw it into the mix (the set of already existing low-level heuristics) and let the black box do its job. The hyperheuristic will choose an appropriate heuristic at each decision point.

5. The hyperheuristic framework offers a great deal of flexibility and there is a wide range of possible modifications.

5.3.3 A simulated annealing hyperheuristic

In addition to the simple and choice function hyperheuristics, we have also implemented a simulated annealing hyperheuristic which will be used in the next chapter for further means of comparison [71].

The idea behind simulated annealing was explained in chapter 3. Our simulated annealing hyperheuristic chooses the low-level heuristics uniformly at random and applies them once. The simulated-annealing hyperheuristic differs from the simple hyperheuristics in the decision as to whether to accept a new candidate solution or not. The new candidate solution is accepted if it is better than the current solution. If not, then it may be accepted (with a certain probability). The acceptance probability is high in the beginning of the search to allow for a wider exploration of the search space and gradually decreases as the search progresses to allow for intensification. The acceptance probability is controlled using a temperature parameter (cooling schedule) [2]. We use a geometric cooling schedule for our simulated annealing hyperheuristic. This cooling schedule is used quite often in practice [2]. The initial value of the *temperature* is often set to a value that represents the maximum difference (in the evaluation function) between two consecutive solutions (the previous and the one obtained from the previous by making a heuristic move) [2]. In our case an initial *temperature* of 50% of the evaluation function value of the starting solution (that produced by our greedy algorithm of Section 2) produced consistently good results (this was obtained after experimenting with different initial temperatures). In the geometric cooling schedule, the temperature is typically decreased by a factor k where $0.8 \leq k < 1$ [2]. For our simulated-annealing hyperheuristic, the value of the temperature was decreased by a factor of 0.85 at each iteration (after experimental trials). The resulting hyperheuristic is described in the following pseudocode.

```
Do
    Choose a low-level heuristic uniformly at random and apply it to obtain NewSolution
    Calculate $I = E_{CurrentSolution} - E_{NewSolution}$
    Accept NewSolution with probability $p = Min\{1, e^{\frac{I}{Temperature}}\}$
    $Temperature \leftarrow Temperature \times TemperatureMultiplier$
Until stopping condition is met
```

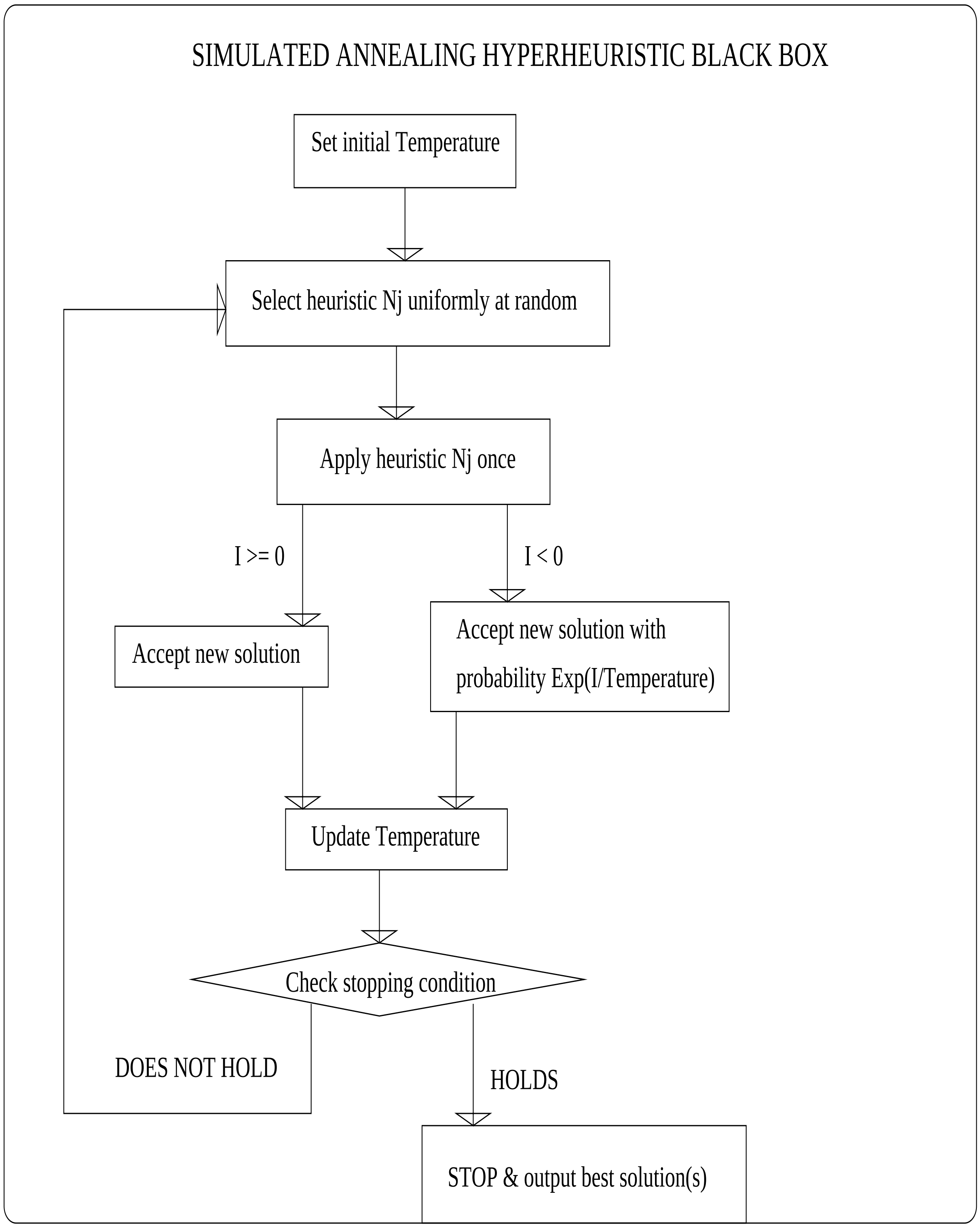

SIMULATED ANNEALING HYPERHEURISTIC BLACK BOX
Set initial Temperature
Select heuristic Nj uniformly at random
Apply heuristic Nj once
I >= 0
I < 0
Accept new solution
Accept new solution with probability Exp(I/Temperature)
Update Temperature
Check stopping condition
DOES NOT HOLD
HOLDS
STOP & output best solution(s)

Figure 5.6: Simulated annealing hyperheuristic framework

Schematic view of the simulated annealing hyperheuristic

We also present the flow chart of the simulated annealing hyperheuristic in Figure 5.6. Again, I is the *relative* improvement, that is, the change in the evaluation function value from the previous solution to the new solution obtained by applying the selected heuristic (whether in a single-call or a steepest descent fashion). Therefore a positive I means that the new solution is better than the previous (minimisation problem). The 'Update Temperature' phase in Figure 5.6 is the cooling schedule. It can be carried out in various ways [2]. In the implemented version of our simulated annealing above, we use a geometric cooling schedule [2].

5.4 Summary

We have presented simple random hyperheuristics, a choice function hyperheuristic, and a simulated-annealing hyperheuristic. The choice function hyperheuristic exists in two variants: a single-objective version, which uses the choice function expression in equation (5.6) or (5.8), and a multiple-objective version, which uses individual choice functions with respect to each individual objective l in equation (5.7). While the former is used when dealing with single-objective optimisation problems, the latter may be used when there are multiple objectives (in this case, each individual objective is considered separately). Of course, if several objectives are aggregated into one objective function, both variants of the choice function can be used. Our simple hyperheuristics will serve primarily as a basis for comparison against the choice function hyperheuristic. Overall, the choice function hyperheuristic when considering individual objectives separately (i.e. equation 5.7) works as follows.

Do

```
        Choose a search criterion $l$.
        Select the low-level heuristic that maximises $f_l$ and apply it.
        Update the choice function $f_l$'s parameters using the adaptive procedure above.
    Until Stopping condition is met.
```

where a number of different ways are possible for choosing criterion l at each step. To avoid cycling problems, we may choose l stochastically rather than deterministically. More precisely we choose a given criterion l with a certain probability p_l proportional to the relative importance of that particular criterion $l \in \mathbf{L}$ in the objective function. Also we apply low-level heuristics in a descent fashion when exploiting the search experience, as guided by f_1 and f_2 and in a single-call fashion when exploring the search space, as guided by f_3.

Overall, the choice function hyperheuristic for a single objective problem or when (in the case of a multi-objective problem) all individual objectives are aggregated into a single objective, works as follows.

```
    Do
        Select the low-level heuristic that maximises $f$ and apply it.
        Update the choice function $f$'s parameters using the adaptive procedure.
    Until Stopping condition is met.
```

Note that if $|\mathbf{L}| = 1$, the choice function hyperheuristic (5.6) or (5.7) and (5.8) are the same.

Remarks / Notation

- The next three chapters will each be devoted to the application of our hyperheuristic methods to three different problems. In all our experimental results we shall refer to our simple hyperheuristics as SR, RD, RP and RPD, meaning $SimpleRandom$, $RandomDescent$, $RandomPermutation$ and $RandomPermutationDescent$ respectively (see section 5.2). The choice function hyperheuristic of equation (5.6) will be denoted as CFa, and the choice function hyperheuristic of equation (5.7) as CFb. The simulated annealing hyperheuristic will be denoted as $SAHH$.
- For all three types of hyperheuristics (simple, choice function and simulated annealing), we consider both acceptance criteria for the solutions produced by the low-level heuristics, that is, All Moves (AM) and Only Improving (OI).

Chapter 6

Application to sales summit scheduling

6.1 Introduction

Our first application problem is that of scheduling a sales summit. This is a real-world problem encountered by a UK commercial company. The aim of this chapter is to show that our choice function hyperheuristic is an effective method which uses little domain knowledge (contained in the low-level heuristics). Thus, in addition to the hyperheuristic methods described in chapter 5, two more methods which were developed by us are also presented in this chapter. They serve as an additional means of comparison against the choice function hyperheuristic. The structure of this chapter is as follows. In section 6.2 we give a description and mathematical formulation of the problem. Section 6.3 is devoted to experiments including the description of additional solution methods and section 6.4 concludes the chapter.

6.2 The sales summit scheduling problem

6.2.1 Problem description

The sales summit scheduling problem is that of a commercial company organising sales summits which involve two groups of company representatives: on the one hand *suppliers*, who want to sell products or services, and on the other hand *delegates* who are representatives of companies that are potentially interested in purchasing those products and services. Suppliers pay a registration fee to have a stand at the sales summit and provide a list of the delegates that they would like to meet. A meeting (between one delegate and one supplier) is classified as *Priority* or *Non-Priority* depending on how strongly the supplier would like to meet the corresponding delegate. Delegates pay no fee but instead are a cost to the commercial company who pay for their travel and hotel expenses. In addition to meetings, seminars are organised where delegates may meet other delegates. Each delegate provides a list of the seminars that he/she would like to attend and (if he/she is invited to the summit) is guaranteed attendance at all requested seminars. There are t meeting timeslots available for both seminars and meetings, and each seminar lasts for a whole number (3 or 4) of consecutive supplier/delegate meeting timeslots. There are s suppliers, d potential delegates and sem seminars. First, delegates are assigned to all seminars that they have requested. Then the aim is to schedule meetings, that is, determine (supplier, delegate, timeslot) triples under the constraints: (a) Each delegate may be scheduled for at most $Maxmeet$ meetings; (b) Each delegate must be scheduled for at most one activity (meeting or seminar) within a given timeslot; (c) Each supplier must be scheduled for at most one meeting within a given timeslot; (d) Each supplier should ideally have at least $MinPrmeet$ priority meetings; (e) Each supplier should ideally have at least $Minmeet$ (priority and non-priority) meetings.

These two latter constraints are soft constraints that can be modelled as penalties in the objective function. Hence the overall objective is to minimise the number of delegates who will actually attend the sales summit out of the d possible delegate attendees and thus minimise cost as well as ensuring that suppliers have sufficient delegate meetings (Priority and Non-Priority).

6.2.2 Problem formulation

We denote by $\mathbf{S}$ (respectively $\mathbf{D}, \mathbf{T}$) the set of suppliers (respectively delegates, timeslots). Let P_{ij} be 1 if (supplier i, delegate j) is a Priority meeting and 0 otherwise ($i \in \mathbf{S}, j \in \mathbf{D}$). Let T_j be the set of timeslots when delegate j is available for supplier meetings (and is not attending a seminar). Our decision variables are denoted by matrix $x = (x_{ijk})$ of $S \times D \times T$ dimension, ($i \in \mathbf{S}, j \in \mathbf{D}, k \in \mathbf{T}$), where x_{ijk} is 1 if supplier i is to meet delegate j in timeslot k, otherwise x_{ijk} is 0. The formulation is given as follows:

Minimise $E(x) = B(x) + 0.05C(x) + 8V(x)$

Subject to

$$\sum_{i \in \mathbf{S}} \sum_{k \in \mathbf{T_j}} x_{ijk} \leq Maxmeet, \quad (j \in \mathbf{D}) \tag{6.1}$$

$$\sum_{i \in \mathbf{S}} x_{ijk} \leq 1, \quad (j \in \mathbf{D}, \quad k \in \mathbf{T_j}) \tag{6.2}$$

$$\sum_{\{j:k\in T_j\}} x_{ijk} \leq 1, \quad (i \in \mathbf{S}, \quad k \in \mathbf{T}) \tag{6.3}$$

$$x_{ijk} \in \{0,1\}, \quad (i \in \mathbf{S}, \quad j \in \mathbf{D}, \quad k \in \mathbf{T}) \tag{6.4}$$

where $B(x) = \sum_{i\in\mathbf{S}} \left[max\left(0, MinPrmeet - \sum_{j\in\mathbf{D}} \sum_{k\in\mathbf{T_j}} P_{ij}x_{ijk}\right)\right]^2$,

$C(x) = \sum_{i\in\mathbf{S}} \left[max\left(0, Minmeet - \sum_{j\in\mathbf{D}} \sum_{k\in\mathbf{T_j}} x_{ijk}\right)\right]^2$,

and $V(x) = \sum_{j\in\mathbf{D}} min\left[1, \sum_{i\in\mathbf{S}} \sum_{k\in\mathbf{T_j}} x_{ijk}\right]$.

Constraint (1) expresses the fact that no delegate must be scheduled for more than *Maxmeet* meetings. Constraints (2) (and (3)) express the fact that no delegate (supplier) must be scheduled for more than one activity within the same timeslot. Both $B(x)$ and $C(x)$ of the objective function are the relaxation of the constraints on suppliers' meeting-satisfaction. More precisely $B(x)$ represents the penalty associated with suppliers who have less than $MinPrmeet$ priority meetings, where constraint (d) is not satisfied, $C(x)$ represents the penalty associated with suppliers with less than $Minmeet$ meetings in total, where constraints (e) is not satisfied. $V(x)$ is the number of delegates who attend the sales summit in the meeting schedule. The different coefficients in $E(x)$ represent the subjective assessment of the relative importance of each criterion. They were obtained after discussion with the user. To simplify the notation we shall assume x and refer to the above quantities as E, B, C and V. Note that this problem has 3 objectives aggregated in one single objective function E.

The company uses a computerised greedy algorithm that produces a solution which (it was felt) used too many delegates. We developed another greedy algorithm, described in [69] and presented in the pseudocode below, which yields a better solution. This latter solution is used as starting solution in all our hyperheuristic algorithms in the next section.

```
Do
    1- Let $\mathbf{S_0}$ be a list of suppliers ordered by increasing number of scheduled priority meetings (and increasing number of total meetings where two suppliers have the same number of priority meetings in the current schedule).
    2- Let $\mathbf{D_O}$ be a list of delegates who currently have less than $Maxmeet$ meetings scheduled, ordered by decreasing number of meetings scheduled.
    3- Find the first supplier $s \in \mathbf{S_O}$ such that there is a delegate $d \in \mathbf{D_O}$ where both $s$ and $d$ have a common free timeslot $t$, and $(s, d, t)$ is a priority meeting.
    4- If no meeting triple was found in 3, then find the first supplier $s \in \mathbf{S_O}$ such that there is a delegate $d \in \mathbf{D_O}$ where both $s$ and $d$ have a common free timeslot $t$, and $(s, d, t)$ is a non-priority meeting.
Until no meeting is found in either step 3 or step 4.
```

By considering the most priority-meeting dissatisfied supplier first at each iteration, we attempt to treat suppliers equitably. By attempting to choose the busiest possible delegate at each iteration, we try to minimise the number of delegates in the solution.

6.3 Experimental study

6.3.1 Problem instances

We used twelve instances of the sales summit scheduling problem. We obtained two real-world data sets and considered several different sets of values for parameters t, $MinPrmeet$ and $MinMeet$ for each data set. Hence instances DR, DR1, DR2 and DR3 are based on one data set, and instances DR4 and DR5 are based on the other. The other six instances were created randomly based on the characteristics of the real-world data. To analyse the relative complexity of each instance we considered the following two criteria:

Distribution of the suppliers' demand (Γ^p): This is the most important criterion. If there is a large number of 'popular' delegates, who have been requested by a large number of suppliers, then it is difficult to schedule meetings satisfactorily. To estimate the degree of popularity of the delegate, we can ask the question as to what proportion of delegates have been requested for a priority meeting by at least a given percentage x of the suppliers. We denote by Γ^p_x the value representing the answer to that question. The larger Γ^p_x is the more difficult it is to find a free (available) delegate and arrange meetings involving him/her. In particular, the larger Γ^p_{50} is the more difficult it is to find a free (available) delegate and arrange meetings involving him/her. Γ_y represents the proportion of delegates who have been requested for a meeting (priority or not) by at least $y\%$ of the suppliers. The suffix 'p' is used to indicate that only priority meetings are considered.

Average suppliers' demand (D^p): This is given by the average number D^p, of priority meeting request per supplier. It reflects the degree of flexibility of the suppliers. The larger D^p the more flexible the suppliers are and therefore the easier it is to satisfy them. D represents the average number of meeting (priority or not) requests per supplier. Again, the suffix 'p' is used to indicate that only priority meetings are considered.

We use the above criteria to evaluate the relative difficulty of any instance of the sales summit scheduling problem, especially with respect to scheduling priority meetings. More precisely, for any instance of the problem, we define $Dif^p = \frac{\Gamma^p_{50}}{D^p}$ as the relative difficulty of solving that instance. Large values of Dif^p correspond to large values of Γ^p_{50} and small values of D^p which makes the problem relatively difficult. Conversely small values of Dif^p correspond to small values of Γ^p_{50} and large values of D^p which makes it a relatively easy problem.

Table 6.1 presents the different problem instances considered. The random instances (DS1-DS6) are generated according to the following pseudo-code:

```
Choose a value for each of the parameters ($|S|, |D|, |T|, sem$)
Seminar generation: For each delegate Do:
    Choose a number $p \leq sem$ of seminars uniformly at random
    Repeat:
        Choose a seminar uniformly at random and assign it to the delegate
    sem times
Priority-Meeting generation: For each supplier $i$ select a number $d^p_i$ of delegates
```

that supplier i wants to meet with priority.

Non-Priority-Meeting generation: For each supplier i select a number d_i of delegates that supplier i wants to meet without priority.

The random selection of the d_i^p and d_i delegates out of $|D|$ may or may not be based on a uniform distribution. It should be noted that if the distribution is uniform there will be fewer popular delegates (first criterion) as all delegates have an equal chance of being picked up by the suppliers. Consequently this creates the easiest problem possible regarding the first criterion.

In Table 6.1 we give, for each problem instance, the problem parameters, the average demand/priority-demand per supplier (D/D^p), the number of delegates that have been requested for a meeting/priority-meeting by at least 50% of the suppliers ($\Gamma_{50}/\Gamma_{50}^p$) and the resulting relative difficulty (Dif/Dif^p). DS1, DS2 and DS4 are among the easiest instances as their meetings were generated uniformly at random. Although they have many popular delegates, their suppliers have requested to meet many delegates and are thus relatively flexible. The fact that many delegates are so popular is a corollary of the fact that many suppliers have requested many meetings. DR4 is difficult because of its large proportion of popular delegates. Its seminars are run in parallel sessions over two days and all seminars are repeated on the second day to allow delegates to attend those seminars that they could not attend on the first day due to parallel sessions. In DR4 delegates are assigned to either session (Day 1 or Day 2 session) of the seminars with the same probability whereas in DR5 delegates are assigned to the first day session of seminars whenever possible. This further complicates the problem of scheduling meetings of this instance as most of the

Instance	Parameters	D/D^p	$\Gamma_{50}/\Gamma_{50}^p$	Dif/Dif^p
DR	(43, 99, 24, 17, 20, 12)	40.30/26.32	26/8	0.65/0.30
DR1	(43, 99, 23, 17, 20, 12)	40.30/26.32	26/8	0.65/0.30
DR2	(43, 99, 25, 17, 20, 12)	40.30/26.32	26/8	0.65/0.30
DR3	(43, 99, 23, 19, 21, 12)	40.30/26.32	26/8	0.65/0.30
DS1	(43, 99, 24, 15, 19, 12)	59.62/39.65	94/6 (Uniform)	1.57/0.15
DS2	(50, 100, 20, 16, 19, 10)	55.72/38.80	78/4 (Uniform)	1.39/0.17
DS3	(43, 99, 24, 15, 17, 12)	38.88/23.41	4/0	0.17/0.00
DS4	(50, 100, 24, 17, 20, 12)	45.42/27.44	27/0 (Uniform)	0.98/0.00
DS5	(43, 99, 24, 15, 17, 12)	38.27/18.86	8/0	0.42/0.00
DS6	(43, 99, 24, 15, 17, 12)	39.02/19.46	6/0	0.24/0.00
DR4	(21, 62, 23, 16, 20, 11)	29.85/24.14	28/17	0.93/0.70
DR5	(21, 62, 23, 16, 20, 11)	29.85/24.14	28/17	0.93/0.70

Table 6.1: The different problem instances. Problem parameters are given in the format $(|S|, |D|, |T|, MinPrMeet, MinMeet, MaxMeet)$ representing respectively the numbers of suppliers, delegates, timeslots, the minimum number per supplier of priority-meetings, the minimum number per supplier of meetings and the maximum number of meetings per delegate. D is the average number of requested delegates per supplier. Γ_{50} is the number of delegates being requested by at least 50% of the suppliers. $Dif = \frac{\Gamma_{50}}{D}$ is the relative difficulty of the problem. The larger the value of Dif the more difficult the problem. The suffix 'p' in each case is the corresponding measure when only priority meetings are considered.

delegates will be unavailable for meetings on the first day, thus considerably reducing the possibility of scheduling sufficient meetings for each suppliers. Thus DR5 is more difficult than DR4 and is in fact the most difficult of all instances considered. Using Γ_{50} for example, we see that DS5 and DS6 are instances of comparable difficulty. However, if we use the demand criterion we are able to say that DS5 is more difficult than DS6. DR through to DR3 are of comparable difficulty as well (based on Dif) but the larger value of t in DR2 for example makes it an easier problem than DR and DR1. Ordering the different instances by increasing relative difficulty gives the following, approximate order: DS1, DS2, DS4, DS3, DS6, DS5, DR2, DR, DR1, DR3, DR4, DR5.

6.3.2 The low-level heuristics

We used $n_h = 10$ low-level heuristics.

1. Remove one delegate(N_1): This heuristic removes one delegate who has at least one meeting. It chooses the delegate with the least number of priority meetings, and the least number of meetings in total where there is a tie (further ties are broken randomly).
2. Move one meeting for a delegate(N_2): This heuristic takes one delegate (the next one in the list of delegates), removes one of his meetings (next meeting on the data list, in increasing order of priority) and adds one different meeting to him (next meeting on the list, in decreasing order of priority).
3. Add one delegate(N_3): This heuristic chooses one unscheduled delegate (the next one on the list) with the largest number of potential priority meetings and greedily adds as many meetings as possible to him (next meeting on the list, in decreasing order of priority).
4. Move one meeting for a saturated delegate(N_4): Same as heuristic N_2 but considers saturated delegates only (i.e. a delegate who has $Maxmeet$ scheduled meetings already).
5. Add one meeting to dissatisfied supplier(N_5): This heuristic adds one meeting to a supplier who has insufficient meetings in total in decreasing order of priority.
6. Add one meeting to priority-dissatisfied supplier(N_6): This heuristic adds one meeting to a supplier who has insufficient priority meetings in total in decreasing order of priority.
7. Cut surplus supplier meetings(N_7): This heuristic takes each supplier who has

more than *MinMeet* meetings scheduled and removes all the extra meetings (in increasing order of priority).

8. Move one meeting for a supplier(N_8): This heuristic takes one supplier (the next one on the list), removes one of his meetings in increasing order of priority, and adds one different meeting to him, in decreasing order of priority.

9. Move one meeting for a priority-dissatisfied supplier(N_9): Same as heuristic 8 but considers suppliers who have insufficient scheduled priority meetings only.

10. Move one meeting for a priority-dissatisfied supplier Version 2(N_{10}): Same as heuristic 9 but here we allow the addition of a new delegate (delegate who currently has no meeting).

These heuristic moves reflect the methods used to manually improve solutions. All low-level heuristics were easy to implement as they are all based on adding/removing objects (a delegate or a meeting). Each low-level heuristic exists in a single call and a descent form. It was quite easy to code these heuristics (a couple of days at the early stage of my PhD research).

6.3.3 Simple, simulated-annealing and choice function hyperheuristics

All algorithms were coded in Microsoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running under Microsoft Windows 2000 version 5. In all experiments the stopping condition was 600 seconds of CPU time. Unless otherwise specified all experimental results were averaged over 10 runs (results were not significantly different) [71]. For each algorithm we distinguished the case where all moves (AM) are accepted and the case where only improving moves (OI) are accepted.

In Table 6.2 (page 152) we present results for the real-world instances and in Table 6.3 (page 152) those for the random instances. In both tables we report results produced by all three types of hyperheuristics (simple, simulated annealing, and choice function). The choice function results are those produced by CFb. Preliminary experiments using CFa were very unsuccessful with results often worse than those of the simple hyperheuristics. Previously, the company used a greedy heuristic to solve instance DR. The company's greedy heuristic produced a solution of evaluation $E = 1020.43$ ($B = 226, C = 48.65, V = 99$). Our greedy algorithm produced a better solution of 784.40 ($B = 0.00, C = 8.00, V = 98$). All of our hyperheuristics produced results dramatically better than those used by the problem owner and are worthy of further investigation. Moreover, the problem owner's subjective assessment was that these solutions represented a significant improvement and an implementable solution.

We can see that for the real-world instances in Table 6.2 the best results produced by *CFb* and *SAHH* are better than those produced by the simple hyperheuristics. This is also true of most of random instances in Table 6.3. We note that this superiority slightly decreases when solving easy instances such as our easy random instances. This suggests that for easy problems our simple hyperheuristics may be able to produce acceptable solutions quite quickly. Apart from the difficult instance DR5, *SAHH* gave better results than *CFb* for the real-world instances. Similarly for the random instances, $SAHH$ produces better results than CFb except for the difficult instance DS5 of this category. This may suggest that CFb is better equipped for difficult instances than $SAHH$. We shall investigate the behaviour of CFb in the next 3 sections from different perspectives. For the easier random instances both hyperheuristics have comparable performance. The fact that $SAHH$ finds better solutions than CFb appears to be explained by the fact that the latter does not diversify by accepting non-improving solutions as often as the former, especially in the early stage of the search. Thus $SAHH$ escapes from local optima more effectively than CFb.

Simulated annealing metaheuristic: To provide a further comparison, we also solved the sales summit scheduling using a 3-phase simulated annealing metaheuristic. Our simulated annealing algorithm was designed by hand. Each phase used a simulated annealing associated with different neighbourhood structures. More precisely, in phase 1 the simulated annealing metaheuristic is associated with three types of neighbourhood structures (remove one delegate, add one delegate, cut surplus meetings for suppliers), which are heuristics 1, 3 and 7 above. The second phase uses a simulated annealing associated with a different set of three neighbourhood structures. The first neighbourhood structure changes or moves the delegate meeting supplier s in timeslot t. It chooses one scheduled meeting uniformly at random, removes it, and schedules the first meeting encountered which involves the same supplier as that of the removed meeting. The other two neighbourhood moves are 'swap two meetings in two different timeslots' and 'add one meeting to a priority-dissatisfied supplier'. The former of them chooses one scheduled meeting uniformly at random and searches for the first scheduled meeting involving a different pair of supplier-delegate in order to make the swap. An other variant of this neighbourhood move was also considered where the swap does not decrease the number of priority meetings of the suppliers involved in the swap. The latter type of neighbourhood move is heuristic 6 above. Finally a third phase uses a simulated annealing associated with a more complex type of neighbourhood move often referred to as *compound move* or *chain move* [89, 264]. For example, given three meetings $(s_1, d_1, t_1), (s_2, d_2, t_2)$ and (s_3, d_3, t_3) where (s, d, t) means that a meeting between supplier s and delegate d is scheduled in timeslot t, the chain of 3 moves will move supplier s_1 to meet delegate d_2 in timeslot t_2 and move supplier s_2 to meet delegate d_3 in timeslot t_3 and move supplier s_3 to meet delegate d_1 in timeslot t_1. So that the resulting solution now includes newly arranged meetings $(s_3, d_1, t_1), (s_1, d_2, t_2)$ and (s_2, d_3, t_3). Chains of moves are robust neighbourhood moves that allow for further improvements in cases where simple neighbourhood moves cannot [89, 264]. In all chain moves, the choice of the first (s, d, t) in the chain is made uniformly at random while the subsequent (s, d, t) of the chain are chosen on a first-encountered basis. As in the second phase, an advanced variant of these chain moves was considered where no decrease of the number of priority meetings of the suppliers involved is allowed. In addition to the three phases, we allowed for re-heating of the initial temperature of the simulated annealing by cycling round the three phases in the order phase1, phase2, phase3, phase1 and so forth. The transition from the current phase to the next one takes place when, within the current phase, there have been $NonImprov$ consecutive iterations without improvement. The next phase is then invoked starting from the best solution found so far.

We used a geometric cooling schedule and, in addition to fine-tuning the temperature parameter, $TemperatureMultiplier$, we had to fine-tune $TemperatureLength$ parameter which represents the number of iterations before we decrease the temperature by $TemperatureMultiplier$, and $NonImprov$. After experimentation we retained $TemperatureMultiplier = 0.85$ as for the simulated annealing hyperheuristic, $NonImprov = 10$ and $TemperatureLength = 2$. The initial temperature was set to 50% of the evaluation function value of the starting solution (that produced by our greedy algorithm of Section 2) [2]. It should be noted that the implementation of the final version of the simulated annealing metaheuristic was arrived at after a good deal of fine tuning and experimentation. In both Table 6.2 and Table 6.3 we give the results obtained by the simulated annealing metaheuristic ($SAMH$).

We first note that the metaheuristic produced results which were better than those obtained from the greedy algorithm used by the company. To our surprise, hyperheuristics produced better results than those produced by the simulated annealing metaheuristic in most instances. The only instances where the simulated annealing metaheuristic beat the hyperheuristics are DS2 and DS4, which are among the easiest ones. This suggests that overall, $SAHH$ and CFb are quite robust when compared to the simulated annealing metaheuristic and as such, can cope with difficult instances better than $SAMH$, which used much richer heuristic moves. It seems that combining simple heuristics in an intelligent manner (as in CFb and $SAHH$) may not only yield a better overall improvement than using these low-level heuristics individually, but also this improvement is greater than that made by a special-purpose metaheuristic such as $SAMH$ which makes use of higher-quality neighbourhood moves. It should be noted that while $SAHH$ was designed for the problem, it was not extremely tuned so that it performed well on the instances considered.

In the next 3 sections we investigate the behaviour of the choice function hyperheuristic from different perspectives. We aim to understand its effectiveness.

Algorithm	DR2	DR	DR1	DR3	DR4	DR5
Greedy Heuristic	784.10	784.40	786.35	830.45	496.45	706.75
RP-AM	768.62	775.46	775.19	825.76	444.60	640.08
RP-OI	593.68	635.81	661.26	795.72	352.20	691.00
RPD-AM	595.91	639.58	659.07	794.56	358.65	691.00
RPD-OI	595.63	633.05	658.81	794.74	363.84	691.00
SR-AM	685.44	712.71	729.20	816.78	360.62	622.20
SR-OI	609.44	640.53	661.44	799.70	362.84	691.00
RD-AM	596.54	639.03	658.33	795.99	350.26	691.00
RD-OI	594.46	638.04	659.33	798.09	359.40	691.00
CFb-AM	600.28	619.46	665.17	798.35	324.06	**610.31**
CFb-OI	590.21	641.12	658.52	794.80	356.23	691.00
SAHH-AM	**566.86**	**582.44**	**635.03**	**784.79**	**319.19**	627.79
SAHH-OI	598.05	635.53	661.30	798.24	349.39	691.00
SAMH	603.57	642.86	674.22	802.77	358.30	696.07

Table 6.2: Experiments with 10 low-level heuristics, real-world data- instances are ordered in increasing difficulty (from left to right)

Algorithm	DS1	DS2	DS4	DS3	DS6	DS5
Greedy Heuristic	792.00	800.40	800.00	792.00	792.00	792.00
RP-AM	721.86	791.89	771.30	701.79	778.41	764.83
RP-OI	**541.90**	711.08	**637.68**	569.18	615.25	628.05
RPD-AM	551.43	716.57	654.40	577.28	613.50	639.43
RPD-OI	550.26	717.29	648.20	580.89	612.70	639.43
SR-AM	669.49	740.29	699.20	674.93	666.85	695.16
SR-OI	551.51	707.30	645.33	569.77	614.06	632.62
RD-AM	549.56	715.15	649.56	577.49	614.48	636.43
RD-OI	551.26	715.32	647.76	576.90	613.09	639.92
CFb-AM	546.30	699.81	641.43	566.11	610.43	**627.86**
CFb-OI	555.15	720.35	639.06	572.05	614.26	632.35
SAHH-AM	552.28	710.38	638.91	**561.68**	**610.22**	631.70
SAHH-OI	547.85	707.84	646.11	569.77	614.14	637.51
SAMH	570.34	**694.15**	637.78	569.20	613.41	634.53

Table 6.3: Experiments with 10 low-level heuristics, random data- instances are ordered in increasing difficulty (from left to right)

6.3.4 Effectiveness and learning ability of the choice function hyperheuristic

In this section we report experiments conducted to determine how effective each hyperheuristic approach is. We first focused on the question as to how often the application of the low-level heuristic chosen by the hyperheuristic leads to a better global solution. To do this we defined function $\kappa(N_j, \rho) = 10 \times \frac{freqB(N_j,\rho)}{freq(N_j)}$ for each low-level heuristic N_j, where $freqB(N_j, \rho)$ is the number of times that the application of N_j has produced an absolute improvement (a solution better than the best solution so far) within the next ρ low-level heuristic calls. For a given low-level heuristic N_j, we can evaluate how effectively different hyperheuristics have been able to use it by comparing the value of $\kappa(N_j, \rho)$ with respect to each of these hyperheuristics at the end of a given run. The hyperheuristic in which the value of $\kappa(N_j, \rho)$ is high would indicate that that hyperheuristic 'makes the most' of heuristic N_j. Here we chose $\rho = n_h = 10$, the number of low-level heuristics used. The idea here is that the ρ heuristics which prepare the solution for an absolute improvement should all be equally rewarded. In Table 6.4 and Table 6.5 (page 157) respectively we present values of κ for each of the 10 low-level heuristics when *RP-OI* and *CFb-AM* respectively are applied to the real-world problem instance DR (in a single run of 600 seconds of CPU time). A comparison of κ for both algorithms shows that the choice function hyperheuristic makes a more effective choice of the low-level heuristics than does a simple hyperheuristic for 9 low-level heuristics out of 10. This means that for most heuristics (9 out of 10), the choice function hyperheuristics was capable of learning to combine and use them more effectively than a simple hyperheuristic in which the learning is disabled. During a run, the value of κ typically increases in the early part of the search (as it is easier then to produce better absolute solutions) and then gradually decreases as it becomes more and more difficult to produce better absolute solutions. Although the general trend is downward, there are sudden peaks when an absolute improvement has just been achieved.

We now focus on the adaptive procedure used in our more sophisticated parameter-free hyperheuristic to adjust the choice function parameters and reflect the interplay of the different low-level heuristics. Our intention is to see if there really is an interplay between the low-level heuristics, the region of the search space being explored and the different factors f_{1l}, f_{2l} and f_3 of the choice function. We considered a number of 'extreme' initial values assigned to the different parameters ($\alpha_b, \alpha_c, \alpha_v, \beta_b, \beta_c, \beta_v$ and δ). We noticed that the value of parameters α and β does not vary much throughout the hyperheuristic search (applied to average instance DR) whereas that of parameter δ varies considerably during the entire search. Note that there are only 2 degrees of freedom in the formula $\alpha f_1 + \beta f_2 + \delta f_3$, so we would expect at most 2 of parameters α, β, δ to change usefully, i.e. we could fix $\delta = 1$ and vary α, β. In a preliminary experiment we assigned values close to 0 and values close to 1 to each parameter in order to see if such values would converge to a specific range of values during the search. This did not happen. More precisely we noticed that very often both parameters α and β keep their initial values (whether extreme or not) within a very narrow range while parameter δ varies across a large range of values during the search without any clear evidence of convergence towards a narrow range of values.

To measure the relation between the choice function's parameters, we define $\lambda = \frac{\alpha_b+\alpha_c+\alpha_v+\beta_b+\beta_c+\beta_v}{6\delta}$. Bearing in mind the fact that parameters α and β are associated with the exploitation of the search and that parameter δ is associated with the exploration of the search space, λ reflects how the choice function balances the desire to exploit the search experience and the desire to explore other heuristics, dependent on the performance of the low-level heuristics and of the region of the search space in which the search finds itself. Figure 6.1 presents the evolution of λ and E for *CFb-AM* over the number of low-level heuristic calls when exploitation parameters α and β and exploration parameter δ are given initial extreme values of 0.01 and 0.99.

λ is kept almost constant in cases illustrated by both Figure 6.1 (a) and (b), thus showing that a certain narrow range of values of the proportion of exploitation/exploration has been arrived at and has been retained for the entire duration of the search. More precisely, in Figure 6.1 (a), λ started off with a value of 99 and suddenly dropped from 98.99 to 4.7×10^{-4} at heuristic call 42. λ then varies within the narrow range of 4.7×10^{-4} to 9.8×10^{-6} (with a peak of 1.3×10^{-3} from heuristic call 60 to 75). In Figure 6.1 (b), λ started off with a value of 1 then at heuristic call 45 it suddenly increased from 0.99 to 4.66. It then drops down to 6.8×10^{-3} at heuristic call 56 and varies in the narrow range of 6.8×10^{-3} and 4.6×10^{-5}. In the case of Figure 6.1 (a) the search led to a solution of evaluation 609.10. Figure 6.1 (b) yielded a solution of 628.05. In both Figure 6.1 (c) and Figure 6.1 (d) our adaptive parameter tuning allows for wide variation in λ, and thus in the relative importance of exploitation and exploration. Figure 6.1 (c) led to a solution of evaluation 621.75 and Figure 6.1 (d) to a solution of evaluation 624.15. So it would appear that this variability in λ is less effective than a small value of λ as in Figure 6.1 (a) and Figure 6.1 (b) Also, it seems that small (α, β) gives lots of 'noise' although λ still remains small. Overall we can say that the procedure for varying parameters α, β, δ might produce solutions better than fixing those parameters or assigning random values to them. In Figure 6.1 (e) we present the individual evolution of parameter δ corresponding to the case of Figure 6.1 (b). The values of δ are changed in a consistent

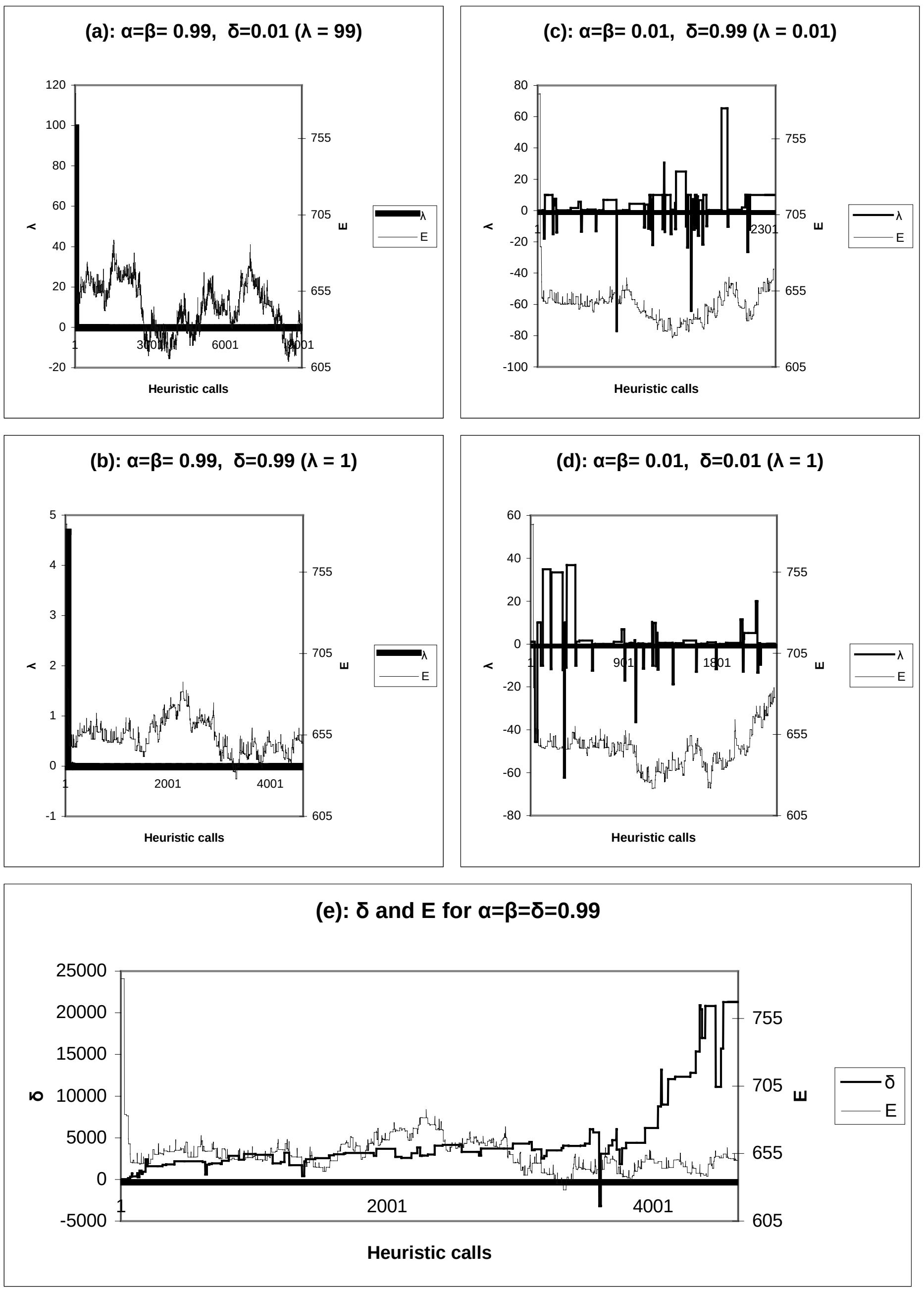


Figure 6.1: Interplay between the choice function parameters, for DR

Heuristic	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}
$freq/freqB$	329/15	359/7	336/1	366/7	297/10	353/11	349/12	352/7	344/11	369/9
κ	0.45	0.19	0.02	0.19	0.33	0.31	0.34	0.19	0.31	0.24

Table 6.4: $freq(N_j)/freqB(N_j,\rho)$ and $\kappa(N_j,\rho)$ for RP-OI, $\rho = 10$

Heuristic	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}
$freq/freqB$	713/40	137/6	65/3	136/3	87/4	132/5	144/6	144/6	205/3	75/3
κ	0.56	0.43	0.46	0.22	0.45	0.37	0.41	0.41	0.14	0.40

Table 6.5: $freq(N_j)/freqB(N_j,\rho)$ and $\kappa(N_j,\rho)$ for CFb-AM, $\rho = 10$

manner with those of E. Thus its value drops as we get a better solution and its value increases (suddenly) in order to escape from local optima. This again results from the way our procedure for parameter settings works. There seems to be an interplay between the performance of the low-level heuristics and the region of the search space currently under exploration. By modifying parameters α, β, δ, hence by modifying the proportion of exploitation/exploration appropriately, the adaptive procedure attempts to reflect this interplay.

We then introduced two other heuristics called *Idle* and *Nasty*. The Idle heuristic is a heuristic that does nothing whatsoever. It simply consumes CPU time. The Nasty heuristic removes all meetings. We wanted to see how the choice function hyperheuristic would react to the presence of such poor heuristics. To do this we compare solution quality. We also give the total amount of time used by $CFb-AM$ on each of these two heuristics. Table 6.6 presents results (time spent on *Idle* and/or *Nasty*, value of the objective function) when *Idle* alone is added to the ten other heuristics, when *Nasty* alone is added to the ten other heuristics and when both *Idle* and *Nasty* are added to the ten heuristics. Results are averaged over 10 runs of 600 seconds CPU.

We see that $CFb-AM$ performs poorly in presence of *Idle* or *Nasty* used alone

	Idle (sec)	E_{Idle}	$(B,C,V)_{Idle}$	E	B, C, V
Idle	0.013	636.68	27.5/103.60/75.50	619.46	37.70/163.20/71.70
	Nasty (sec)	E_{Nasty}	$(B,C,V)_{Nasty}$	E	B, C, V
Nasty	28.66	647.47	15.00/89.40/78.50	619.46	37.70/163.20/71.70
	Idle/Nasty (sec)	$E_{Idle/Nasty}$	$(B,C,V)_{Idle/Nasty}$	E	B, C, V
$Idle + Nasty$	0.006/39.21	647.35	24.30/109.00/77.20	619.46	37.70/163.20/71.70

Table 6.6: Proportion of time used on *Idle* and *Nasty* low-level heuristics. E_{Idle}, E_{Nasty} and $E_{Idle/Nasty}$ denote E when *Idle* and *Nasty* are introduced alone and when both heuristics are introduced simultaneously. In each case $CFb - AM$ is applied to instance DR, and results are averaged over 10 runs of 600 second CPU.

as well as when both heuristics are simultaneously available. Solution quality is better when using *Idle* than when using *Nasty* which causes more damage to the solution. It is interesting to note that when both heuristics are available, it seems that $CFb - AM$ has a stronger preference for *Nasty* than for *Idle*. This reflects the idea that the hyperheuristic 'hopes' to find out more about the dynamics of the search using *Nasty* than using *Idle*[1]. Using *Idle* will not tell us anything simply because *Idle* does not do anything. Overall it would appear that the choice function hyperheuristic may deliver poorer results when in presence of extremely negative heuristics. We observed that the choice function hyperheuristic was still able to produce results better than those obtained by SR and RD when in presence of *Nasty* and *Idle*. Solutions obtained by SR were even much worse than the starting solution produced by our greedy heuristic. It should be noted, however, that this is an extreme situation that is not likely to occur in practice as the negativity of such heuristics as *Idle* (which consumes time uselessly) and *Nasty* (which actually counters every effort of building a solution) is evident. Therefore no one would actually want to use them as a way of constructing solutions.

In order to analyse which parts of our expression for the choice function were

[1]at least *Nasty* does something.

Choice Function	DR2	DR	DR1	DR3	DR4	DR5
$f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ -AM	606.14	643.90	680.72	803.30	354.31	692.01
$f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ -OI	610.88	646.46	677.22	800.43	367.11	687.71
$f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$ -AM	609.10	648.72	679.27	807.17	349.33	689.84
$f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$ -OI	612.05	642.54	679.68	802.20	367.87	689.94
$f_l = \alpha_l f_{1l} + \beta_l f_{2l}$ -AM	624.60	654.32	688.99	814.34	383.76	690.72
$f_l = \alpha_l f_{1l} + \beta_l f_{2l}$ -OI	610.39	656.39	694.03	808.37	371.72	689.86
$f_l = \alpha_l f_{1l} + \beta_l f_{2l} + \frac{\delta}{c} f_3$ -AM	600.28	**619.46**	665.17	798.35	**324.06**	**610.31**
$f_l = \alpha_l f_{1l} + \beta_l f_{2l} + \frac{\delta}{c} f_3$ -OI	**590.21**	641.12	**658.52**	**794.80**	356.23	691.00

Table 6.7: Different Choice Function expressions, real-world data

Choice Function	DS1	DS2	DS4	DS3	DS6	DS5
$f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ -AM	574.21	708.69	652.11	593.26	614.46	633.92
$f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ -OI	569.47	712.05	652.41	605.92	625.23	642.01
$f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$ -AM	570.29	708.15	653.62	580.27	613.05	635.72
$f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$ -OI	570.49	710.69	650.89	602.31	619.94	647.42
$f_l = \alpha_l f_{1l} + \beta_l f_{2l}$ -AM	580.54	719.02	656.96	623.11	626.92	654.42
$f_l = \alpha_l f_{1l} + \beta_l f_{2l}$ -OI	593.90	715.04	655.79	607.62	621.86	652.91
$f_l = \alpha_l f_{1l} + \beta_l f_{2l} + \frac{\delta}{c} f_3$ -AM	**546.30**	**699.81**	641.43	**566.11**	**610.43**	**627.86**
$f_l = \alpha_l f_{1l} + \beta_l f_{2l} + \frac{\delta}{c} f_3$ -OI	555.15	720.35	**639.06**	572.05	614.26	632.35

Table 6.8: Different Choice Function expressions, random data

the most important, we analysed a range of alternative choice functions, which use the same basic elements. For each criterion $l \in \mathbf{L}$, the choice function f_l consists of three factors: f_{1l} representing the first order performance of the low-level heuristic, f_{2l} the joint performance of a pair of heuristics and f_3 the time since a heuristic was last called. To see which of these three factors is of most importance we considered the following choice function expressions: $f_l = \alpha_l f_{1l}$, $f_l = \beta_l f_{2l}$, $f_l = \alpha_l f_{1l} + \beta_l f_{2l}$, $f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ and $f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$. In both Table 6.7 and Table 6.8, we present the results of the three latter ones.

In 50% of the problems $f_l = \alpha_l f_{1l} + \frac{\delta}{c} f_3$ gave better results than $f_l = \beta_l f_{2l} + \frac{\delta}{c} f_3$ showing that both factors f_{1l} and f_{2l} are about equally important for the search while f_3's presence is more than vital as results without it are significantly poorer. Further, if we compare with the results of Table 6.2 and Table 6.3 (page 152) we see that CFb, which uses all three factors gives the best results. This suggests that it may be worth considering another factor representing the joint performance of a triple or m-tuple of heuristics.

6.3.5 Experiments with a smaller set of low-level heuristics

In the previous sub-section we saw the superiority of both the choice function and the simulated-annealing based hyperheuristic over simple multiple neighbourhood search techniques for difficult problem instances. In this section we show that hyperheuristics can still perform well even in the presence of not-so-rich sets of low-level heuristics. We experimented with a few subsets of four low-level heuristics among the 10 considered above and retained low-level heuristics 1, 2, 3 and 5. Results are given in Table 6.9 (real-world instances) and Table 6.10 (random instances).

We compared results obtained from all three types of our hyperheuristics with those of the simulated annealing metaheuristic described earlier. In all real-world instances, both hyperheuristics CFb and $SAHH$ produced results better than those produced by the simulated annealing metaheuristic. We also noticed that for all real-world instances there was at least one simple hyperheuristic that outperformed the metaheuristic. CFb and *SAMH* showed comparable performance. Similarly, there was at least one simple hyperheuristic that produced results better than those of *SAMH.* Even in presence of a not-so-rich set of low-level heuristics, it seems that hyperheuristics can still produce good quality solutions.

In half of the instances the best result is from a simple hyperheuristic (DR1, DR3, DR, DS1-2,6), but for the difficult instances of DR4 and DR5 both the choice function and the simulated annealing hyperheuristics gave better results than the

Algorithm	DR2	DR	DR1	DR3	DR4	DR5
Greedy Heuristic	784.10	784.40	786.35	830.45	496.45	706.75
RP-AM	776.10	776.30	778.35	826.49	478.63	692.55
RP-OI	608.05	**631.58**	**655.35**	788.35	334.48	691.00
RPD-AM	603.60	645.55	661.70	**788.10**	352.20	691.00
RPD-OI	602.64	645.55	661.70	788.10	352.01	691.00
SR-AM	648.03	690.37	709.01	820.37	355.48	693.79
SR-OI	602.47	635.79	658.58	792.22	352.57	691.00
RD-AM	604.38	644.97	664.83	789.45	349.21	691.00
RD-OI	601.26	646.05	664.70	790.19	352.01	691.00
CFb-AM	602.76	633.62	670.29	811.53	**330.81**	684.34
CFb-OI	**601.12**	642.23	661.07	794.17	349.54	691.00
SAHH-AM	601.69	633.80	660.26	798.88	340.62	**681.42**
SAHH-OI	601.31	640.33	658.61	791.53	348.62	691.00
SAMH	603.57	642.86	674.22	802.77	358.30	696.07

Table 6.9: Experiments with 4 low-level heuristics (1, 2, 3 and 5), real-world data-instances are ordered in increasing difficulty (from left to right)

simple ones. A similar situation happens with the random instances where $SAHH$ and CFb produce better results for the difficult instances DS3 and DS5 of this category. For most instances, the performance of the choice function hyperheuristic decreases with a small number of low-level heuristics (when compared to experiments with 10 low-level heuristics in Table 6.2 and Table 6.3 of page 152), whilst that of the simple hyperheuristic does not change significantly overall, thus reducing the gap between the two categories of algorithms. This can be explained by the fact that the chance of choosing the 'right' low-level heuristic increases when the number of those low-level heuristics is small. Thus for example, in the extreme case of choosing between two heuristics, it would be pointless to use a sophisticated hyperheuristic. A simple greedy (try both heuristic and select the best) or random (choose either heuristic at random) approach might probably yield a result at least as good as that of a choice function hyperheuristic in this case.

Algorithm	DS1	DS2	DS4	DS3	DS6	DS5
Greedy Heuristic	792.00	800.40	800.00	792.00	792.00	792.00
RP-AM	777.60	786.94	792.00	784.00	783.10	784.00
RP-OI	**539.96**	702.61	645.36	614.00	614.05	646.00
RPD-AM	550.40	692.20	642.32	584.10	611.10	646.00
RPD-OI	550.40	692.60	640.40	584.10	611.10	646.00
SR-AM	619.97	736.20	686.58	622.93	642.39	665.18
SR-OI	543.36	697.25	643.70	595.78	614.24	646.00
RD-AM	552.45	**692.01**	641.32	584.08	**611.10**	646.00
RD-OI	550.39	692.15	643.99	584.89	611.10	646.00
CFb-AM	550.84	704.53	655.93	580.27	613.37	638.54
CFb-OI	550.35	693.40	639.88	584.05	612.58	646.00
SAHH-AM	550.34	698.56	640.94	597.64	617.41	634.61
SAHH-OI	545.95	695.04	642.08	594.75	614.33	646.00
SAMH	570.34	694.15	**637.78**	**569.20**	613.41	**634.53**

Table 6.10: Experiments with 4 low-level heuristics (1, 2, 3 and 5), random data-instances are ordered in increasing difficulty (from left to right)

6.3.6 Experiments with a larger set of low-level heuristics

In the two previous sub-sections we saw that hyperheuristics can still produce good solutions even in the presence of a not-so-rich, small set of low-level heuristics. We now consider a larger set of low-level heuristics by including six low-level heuristics which were used in [69]. The low-level heuristics considered in [69] are also based on the same intuitive ideas of manually repairing/improving on a solution (i.e. remove or add events). However, instead of adding one single meeting, they would add as many meetings as possible. The disadvantage of these heuristics is that they perform 'macro' moves and this tended to create plateaux in the search landscape. For example, when adding, say, 6 meetings at a time to all dissatisfied suppliers, it is difficult to reach a solution where one supplier has one or two meetings less. The search jumps over intermediate solutions (e.g a solution with one or two meetings less) which could lead to a globally better solution. In all, the hyperheuristic now manages a set of 16 low-level heuristics including six extra low-level heuristics presented below. However, these 'macro' moves might be expected to perform well in conjunction with ‘micro’ moves.

1. *h11: Add meetings to dissatisfied supplier - version 1:* This heuristic adds as many meetings as possible to one dissatisfied supplier until the supplier is satisfied (if possible), without adding new delegates. This may only involve the deletion and rearrangement of meetings already arranged between delegates and other suppliers, but only for ‘saturated’ delegates who already have $Maxmeet$ meetings.
2. *h12: Add meetings to dissatisfied or priority-dissatisfied supplier:* Same as the previous heuristic except that here the heuristic considers priority-dissatisfied suppliers (who may already have enough meetings, but not of sufficient priority) as well as dissatisfied ones.
3. *h13: Add meetings to dissatisfied supplier - version 2:* Same as in heuristic h_{11}, except that here the heuristic may move meetings of nonsaturated delegates who have less than $Maxmeet$ meetings as well as saturated ones.
4. *h14: Add meetings to priority-dissatisfied supplier:* This heuristic takes a supplier who has too few priority meetings and adds as many priority-meetings as possible to him, without adding delegates or violating the limitation on the maximum number of meetings per delegate .
5. *h15: Add meetings to dissatisfied supplier:* Same as h_{14} but considers only suppliers who have enough priority meetings but too few meetings in total, and adds non-priority meetings, without adding delegates or violating the limitation on the maximum number of meetings per delegate.
6. *h16: Add delegates and meetings to priority-dissatisfied supplier:* Same as h_{14} except we allow the addition of new delegates (those who do not currently have any meetings).

Results are given in Table 6.11 (real-world instances) and Table 6.12 (random instances).

In more than half of the instances (4 real-world instances and 3 random instances) *CFb* beat *SAMH*. *SAMH* was also outperformed by *SAHH* in 8 instances out of 12 (5 real-world instances and 3 random instances). This suggests that both hyperheuristics can produce results competitive with those of our sophisticated simulated annealing metaheuristic. In fact, the simulated annealing metaheuristic was only able to produce results better than the best results produced by the simple hyperheuristics in only a third of the instances considered (2 real-world instances and 2 random instances). It should be mentioned however that this is an unfair comparison as *SAMH* is being compared against the best result amongst 8 simple hyperheuristics. At least it can be said that the sophisticated hyperheuristics (*CF* and *SAHH*) can each compete with *SAMH*. We also note that in 6 instances (3 real-world and 3 random) *CF* produced results better than or equal to the best result amongst the 8 simple hyperheuristics. *SAHH* did slightly worse. It produced results better than or equal to the best simple hyperheuristic result in 5 instances (4 real-world instances and 1 random instance).

It can be seen that overall, results produced by all three types of hyperheuristics (simple, choice function and simulated annealing) are better with both a set of 10 low-level heuristics and a set of 4 low-level heuristics than with a larger set of 16 low-level heuristics. For all three sets of low-level heuristics, the best results produced by the simple hyperheuristics were generally produced by *RP-OI* and *RPD*.

In half of the instances there was at least one simple hyperheuristic that produced results better than those produced by both *CFb* and *SAHH*. We ran all experiments for double the CPU time (that is another 600 seconds of CPU time) on those instances where the simple hyperheuristic produced better results. The sophisticated hyperheuristics ($SAHH$ and CFb) then produced better results on two more problems (results obtained by the choice function hyperheuristic) and the gap between the results of the simple hyperheuristic and the sophisticated ones was reduced in two other instances (one by the choice function hyperheuristic and one by the simulated-annealing one). We have evidence here that the choice function hyperheuristic is more robust than the simulated annealing one which in turn is more robust than the simple hyperheuristics for difficult instances but that robustness decreases when the number of low-level heuristic increases (when compared to experiments with 10 low-level heuristics). Also, the simulated annealing hyperheuristic produced better results than the choice function one in 8 instances. This is likely to be due to the fact that with a large number of low-level heuristics to manage, it becomes more and more difficult to find the 'right' heuristic and the amount of knowledge (hence the amount of time) needed to make a 'good' choice increases. With a very large number of low-level heuristics, it may even be best to choose a low-level heuristic at random rather than spending lots of time trying to learn which low-level heuristic to apply next. It might also be interesting to 'pre-screen' and select a subset of around ten low-level heuristics. The power of the statistical methods will fall as the number of data points decreases.

6.4 Conclusions

We have presented several hyperheuristic approaches applied to a special kind of personnel scheduling problem. Hyperheuristics are approaches that can be developed quickly using limited domain knowledge and expertise by implementing only knowledge-poor low-level heuristics. We have presented three types of hyperheuris-

Algorithm	DR2	DR	DR1	DR3	DR4	DR5
Greedy Heuristic	784.10	784.40	786.35	830.45	496.45	706.75
RP-AM	757.53	760.96	773.06	828.96	465.93	692.55
RP-OI	619.50	645.43	**668.20**	**797.09**	355.76	691.00
RPD-AM	630.68	646.60	676.09	797.32	379.53	691.00
RPD-OI	630.32	646.50	675.68	797.76	374.19	691.00
SR-AM	710.52	723.52	738.80	827.27	412.80	690.66
SR-OI	616.59	645.77	669.53	801.25	355.47	691.00
RD-AM	630.45	645.96	673.54	797.28	373.63	691.21
RD-OI	630.78	645.27	674.08	799.17	376.54	691.21
CFb-AM	**601.46**	643.50	686.95	819.66	360.53	690.09
CFb-OI	618.06	643.97	673.13	798.81	383.80	692.05
SAHH-AM	607.93	**642.52**	676.58	803.13	**338.17**	**688.97**
SAHH-OI	615.85	645.21	670.37	797.97	358.91	691.21
SAMH	603.57	642.86	674.22	802.77	358.30	696.07

Table 6.11: Experiments with 16 low-level heuristics, real-world data- instances are ordered in increasing difficulty (from left to right)

Algorithm	DS1	DS2	DS4	DS3	DS6	DS5
Greedy Heuristic	792.00	800.40	800.00	792.00	792.00	792.00
RP-AM	728.81	792.65	772.02	700.02	772.01	763.20
RP-OI	574.43	**692.11**	639.57	572.05	610.54	637.20
RPD-AM	552.48	693.94	657.53	586.77	609.61	639.43
RPD-OI	550.79	694.29	657.55	588.85	**608.59**	639.44
SR-AM	683.73	732.37	724.52	677.38	691.18	709.88
SR-OI	571.62	693.07	651.45	572.95	613.14	**633.40**
RD-AM	555.16	695.78	654.50	581.14	609.94	639.13
RD-OI	553.63	695.13	649.34	580.79	609.60	643.81
CFb-AM	**547.16**	708.24	649.64	571.63	611.33	638.42
CFb-OI	551.83	694.65	645.18	572.05	611.69	634.17
SAHH-AM	565.11	695.72	641.09	572.49	613.83	637.66
SAHH-OI	568.02	693.73	646.31	571.12	611.58	635.30
SAMH	570.34	694.15	**637.78**	**569.20**	613.41	634.53

Table 6.12: Experiments with 16 low-level heuristics, random data- instances are ordered in increasing difficulty (from left to right)

tics: simple hyperheuristics, a simulated annealing hyperheuristic and a choice function hyperheuristic. All three types of hyperheuristic were applied to a real-world case study problem of scheduling a sales summit and produced results dramatically better than those produced by a greedy heuristic used by the problem owner. In general we have observed, over a range of problems, that multiple neighbourhood approaches greatly outperform single neighbourhood ones. All three types of hyperheuristic considered in this paper are worthy of further investigation. We experimented with different sets of low-level heuristics and it can be said that hyperheuristics can still produce results as good as those produced by a metaheuristic even if the sets of low-level heuristics managed by the hyperheuristic are poorly chosen so long as there are enough heuristics (i.e. at least 'add', 'delete', 'move'). It would appear that the set of low-level heuristics should be complete, ideally rich, but not necessarily too rich. Findings in chapter 8 will further confirm this point. Indeed even when we have pathologically bad low-level heuristics (such as *Idle* and *Nasty*), the hyperheuristic approaches are still capable of producing reasonable solutions, much better than those obtained using the simple hyperheuristics (SR and RD). In fact, in several instances of the sales summit problem considered in this chapter, the hyperheuristic outperformed a simulated annealing metaheuristic equipped with sophisticated neighbourhood structures. Of all types of hyperheuristics considered here, the choice function hyperheuristic was the most robust and outperformed all other hyperheuristics for the difficult instances of the sales summit scheduling problem. The simulated annealing hyperheuristic also outperformed the simple hyperheuristics in difficult instances. The choice function hyperheuristic is equipped with an adaptive procedure which modifies the different parameters of the choice function in an effective manner. The choice function thus attempts to capture the interplay between the different low-level heuristics and the region of the search space currently under exploration, and makes adjustments of the value of its parameters in order to exploit or explore the search. We experimented with different expressions of the choice function and it turned out that each factor - first and second order improvement and exploration function - of the choice function that we considered, contributed to an effective search. Thus, in particular, the second order improvement factor, which represents the joint performance of a pair of heuristics, proved to be useful and if sufficient iterations are possible we might consider triples or m-tuples of heuristics as well. This point will be further discussed in chapter 9.

Chapter 7

Application to Presentation scheduling

7.1 Introduction

In the previous chapter we showed that hyperheuristics, in general, and our choice function hyperheuristics in particular, can be effective for solving a variety of instances of a real-world problem. In this chapter we shall further investigate the power of the choice function hyperheuristic. First we show that it is possible to develop good-quality solutions in a very short amount of time using our hyperheuristic framework. Then we carry out a detailed investigation of the choice function hyperheuristic when compared to a purely random hyperheuristic approach. To do this we apply our hyperheuristics to a different problem, that of scheduling final-year project presentations. This is a real-world problem encountered at the University of Nottingham. This chapter is structured as follows. Section 7.2 describes the problem and gives a mathematical formulation. This is followed in section 7.3 by computational experiments. Section 7.4 concludes the chapter.

7.2 Scheduling of project presentations

7.2.1 Problem description

Every academic year the School of Computer Science and Information Technology of the University of Nottingham is faced with the problem of scheduling final year BSc students' project presentations during a period of up to 4 weeks. As part of their course requirements, final year BSc students have to give a 15-minute presentation of their project. Each student works on a chosen project topic and is assigned a member of academic staff to supervise the project. Project presentations are then organised and each student must present his/her project before a panel of three members of academic staff who will mark the student's presentation: The chair or first marker, the second marker and the observer. Ideally, the project's supervisor should be involved in the presentation (either as chair or observer) but this is often not the case in practice. Once every student has been assigned a supervisor for his/her project, the problem is to schedule all individual presentations, that is, determine a first marker, a second marker and an observer for each individual presentation, and allocate both a room and a timeslot to the resulting quadruple (student, 1st marker, 2nd marker, observer). The presentations are organised in sessions, each containing up to six presentations. Typically the same markers and observers will see all of the presentations in a particular session. So the problem can be seen as that of determining (student, 1st marker, 2nd marker, observer, room, timeslot) tuples, that respect the following constraints:

(1) Each presentation must be scheduled exactly once;

(2) No more than six presentations for each room and for each session;
(3) No member of staff (whether as 1st marker or as 2nd marker or as observer) can be scheduled to 2 different rooms within the same session. In addition presentations can only be scheduled in a given session when both the academic members of staff and the room assigned to those presentations are available during that session. There are four objectives to be achieved:
(A) Fair distribution of the total number of presentations per staff member;
(B) Fair distribution of the total number of sessions per staff member;
(C) Fair distribution of the number of 'inconvenient' sessions per staff member, i.e. sessions at bad times (before 10:00 am, after 4:00 pm);
(D) Optimise the match between staff research interest and project themes, and try to ensure that a supervisor attends presentations for projects which they supervise.

Previously, the problem was solved manually, and objective (D) was largely ignored in this solution process, resulting in a lowered level of marker interest and satisfaction, and student presentations where the level of questions asked felt short of the ideal. All problem requirements were obtained through the School's timetable officer. Also an early model was obtained, which was refined in order to take into account further points including objective (D). It should be mentioned that the notion of 'fairness' here is subjective but has been quantified so that every member of staff should have the same number of presentations/sessions/'inconvenient' sessions. The actual enterprise of developing an automated system for scheduling final year project presentations was for us an exercise of consultancy. This required a number of consultative meetings involving the School's timetabling officer, members of academic staff, and ourselves. From a practical point of view, analysis, development and testing of solutions were carried out within a fairly short amount of time (two weeks) as this project was indeed the rapid prototyping of a real-world solution for the School's final year project presentation scheduling problem.

7.2.2 Problem formulation

To formulate the problem, we denote by **I** the set of students, **S** the set of academic staff members, **Q** the set of sessions and **R** the set of seminar rooms. Our decision variables are denoted by matrix $x = (x_{ijklqr})$ of $I \times S \times S \times S \times Q \times R$ dimension, $(i \in \mathbf{I}, j, k, l \in \mathbf{S}, j \neq k, j \neq l, k \neq l, q \in \mathbf{Q}, r \in \mathbf{R})$, where x_{ijklqr} is 1 if presentation of student i is assigned to 1st marker j, 2nd marker k, observer l and allocated to session q in seminar room r, and where all four persons are available for a meeting, otherwise x_{ijklqr} is 0; and y_{jqr} $(j \in \mathbf{S}, q \in \mathbf{Q}, r \in \mathbf{R})$ where y_{jqr} is 1 if staff j is in room r during session q, otherwise y_{jqr} is 0. We may then formulate the problem as follows:

Minimise $E(x) = 0.5A + B + 0.3C - D$

s.t.

$$\sum_{j,k,l \in \mathbf{S}} \sum_{q \in \mathbf{Q}} \sum_{r \in \mathbf{R}} x_{ijklqr} = 1, \quad (i \in \mathbf{I}) \tag{7.1}$$

$$\sum_{i \in \mathbf{I}} \sum_{j,k,l \in \mathbf{S}} x_{ijklqr} \leq 6, \quad (q \in \mathbf{Q}, r \in \mathbf{R}) \tag{7.2}$$

$$\sum_{r \in \mathbf{R}} y_{jqr} \leq 1, \quad (j \in \mathbf{S}, q \in \mathbf{Q}) \tag{7.3}$$

$$\sum_{i \in \mathbf{I}} \sum_{k,l \in \mathbf{S}} (x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) \leq M y_{jqr}, \quad (j \in \mathbf{S}, q \in \mathbf{Q}, r \in \mathbf{R}) \tag{7.4}$$

$$x_{ijklqr}, y_{jqr} \in \{0,1\}, \quad i \in \mathbf{I}, j,k,l \in \mathbf{S}, j \neq k \neq l, q \in \mathbf{Q}, r \in \mathbf{R} \tag{7.5}$$

where $A = \sum_{j\in\mathbf{S}} \left(\sum_{q\in\mathbf{Q}} \sum_{r\in\mathbf{R}} \sum_{i\in\mathbf{I}} \sum_{k,l\in\mathbf{S}} (x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) - K\right)^2$,

$B = \sum_{j\in\mathbf{S}} \left(\sum_{q\in\mathbf{Q}} \sum_{r\in\mathbf{R}} y_{jqr} - K_1\right)^2$,

$C = \sum_{j\in\mathbf{S}} \left(\sum_{q\in\mathbf{Q_{bad}}} \sum_{r\in\mathbf{R}} y_{jqr} - K_2\right)^2$, and

$D = \sum_{j\in\mathbf{S}} \left(\sum_{q\in\mathbf{Q}} \sum_{r\in\mathbf{R}} \sum_{i\in\mathbf{I}} \sum_{k,l\in\mathbf{S}} (p_{ij} + 10Sup_{ij})(x_{ijklqr} + x_{ikjlqr} + x_{ikljqr})\right)$.

Equations (1), (2), (3) express constraints (1), (2), (3) respectively. Equation (4) links variables x_{ijklqr} with y_{jqr}, where M is a large number. $K = \frac{3|I|}{|S|}$, $K_1 = \frac{6P_1}{|S|}$ and $K_2 = \frac{6P_2}{|S|}$ where K, (K_1/K_2) is the average number of presentations (sessions/'bad' sessions) per member of staff, with P_1 (P_2) the total number of (bad) sessions used in the solution and Q_{bad} a subset of Q containing early sessions (before 10:00 am) and late sessions (after 4:00pm). In objective D, p_{ij} is an integer value associated with the level of matching between the topic of presentation i and the research interest of staff member j if he/she is involved in presentation i. The higher the value of p_{ij}, the better the matching. Sup_{ij} is an indicator of whether staff member j is the supervisor of the project for presentation i. If this is the case, then $Sup_{ij} = 1$. Otherwise $Sup_{ij} = 0$. The different coefficients in the objective function were set so as to reflect the relative but subjective importance of each objective from the point of view of the problem owner. The problem admits a feasible solution if there is enough room-time to allocate each presentation to, hence if $6|R||Q| > |I|$. We used three instances for this problem. The first instance is *csit*0 taken from [73] and has the following problem characteristics $|I| = 151$, $|Q| = 80$, $|R| = 2$, and $|S| = 26$. The last two instances are *csit*1 and *csit*2 taken from [157] with $|I| = 240$, $|Q| = 36$,

$|R| = 2$, and $|S| = 24$ for $csit1$. $csit2$ is the same as $csit1$ except in $|S| = 22$. Thus $csit2$ is more difficult (tighter constraints) than $csit1$. Note that $csit0$ is much easier (slack constraints) than both $csit1$ and $csit2$ as from the former to the latter instances there is a 58% increase in $|I|$ and a 45% decrease in $|Q|$, hence many more projects to schedule in fewer timeslots. In increasing order of difficulty we have $csit0, csit1, csit2$. Note that all instances have thousands of constraints and several millions of variables in our integer programming model.

We developed a constructive heuristic in [73] that produces an initial solution which is better than the manually constructed one. The constructive heuristic iteratively chooses a triple of staff members and assigns them to as many as 6 presentations in the first available session and room. Priority is given to non-bad sessions (to optimise objective C), to presentations whose supervisor is among the three staff and whose project topic is most related to the concerned staff research interest (to optimise objective D) , and the staff members are chosen on a cyclic basis (to optimise objectives A and B). The solution of the constructive heuristic, presented below, is used as starting solution for all our algorithms presented in this chapter. In the next section we report experiments carried out on all hyperheuristics when applied to the CSIT third year problem of scheduling project presentations.

Repeat:

Step1: *-Choose one staff, say j (the next staff on the list - the first one is chosen at random)*

-Find a free PP session for staff j and a room available during that PP session.

-Choose another staff, say k (chosen at random) who is available

```
            during the PP session above.
  Step2:  -Store all projects whose supervisor is either staff $j$ or $k$
          -While number of selected Projects $< 6$ OR all such projects have been
            considered
            – Select the project with the highest research interest (increment
            number of selected projects)
          -For each selected project, schedule it in the room-PPsession, keeping
            the supervisor as the first marker.
  Until all room-PPsessions are utilised OR all PP's are scheduled.
```

By choosing the next staff in the round (in Step 1) and by assigning up to 6 PP's to him/her (in step 2), we aim to minimise objective B (even number of PPsessions per staff). Also, priority is given to those non-early PPsessions, which minimises objective C (minimise the number of early PP sessions). In step2, the aim is to maximise objective D.

7.3 Experiments

The first set of experiments[1] aimed at making a direct comparison between the purely random hyperheuristic approach (RD), and the choice function hyperheuristic (CFa) (though we also give results for other simple hyperheuristics). The choice function results are those produced by CFa. Unlike our experience with the sales summit in the previous chapter, here initial experiments using CFb were very un-

[1]Unless otherwise stated, all algorithms reported in this chapter were coded in Microsoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1500MHz with 228MB RAM running under Microsoft Windows 2000 version 5.

	csit0	csit1	csit2
ch	-908.5	-2557.6	-946.6
RP-am	-1063.97*	-2633.61	-1116.94
RP-oi	-1197.59*	-2878.41	-1625.74
RPD-am	-1287.25*	-2888.58	-1714.37
RPD-oi	-1284.75*	-2880.09	-1688.56
SR-am	-1121.11*	-2620.18	-1100.09
SR-oi	-1193.19*	-2880.93	-1614.11
RD-AM	-1274.55*	-2884.49	-1668.76
RD-OI	-1303.38*	-2892.81	-1675.48
CFa-AM	**-1444.99****	-2960.3	-1650.67
CFa-OI	-1316.56*	**-2963.37**	**-1720.23**
RD1-AM	-1406.64*	-2892.02	**-1724.15**
RD1-OI	-1398.74*	-2887.90	-1720.29
ml	-90.1	-	-
CFa-AM(ml)	-644.43	-	-
s_1	71.1	4304.5	6051.5
CFa-AM(s_1)	-1326.37	-323.23	717.96
s_2	516.8	98.9	986.4
CFa-AM(s_2)	-991.96	-1342.3	-114.5

Table 7.1: Initial solution is *ch*. *ml* is a manual solution produced by the problem owner. All algorithms in the upper part start with ch as initial solution. $HH(x)$ denotes algorithm HH starting with initial solution x. $RD1$ is a hand-made RD which was tailored for this problem. csit0 results marked with * are taken from [73] which used a 1Ghz PC with 128Mb RAM.

successful with results often worse than those of the simple hyperheuristics. For both algorithms we distinguished the case where all moves (AM) are accepted and the case where only improving moves (OI) are accepted. Results (averaged over 10 runs) are given in the upper part of Table 7.1 for the three instances *csit*0, *csit*1 and *csit*2 (results were not significantly different).

We used three types of low-level heuristics based on 'Replacing' one staff member in a session with a different one, 'Moving' a presentation from one session to another, and 'Swapping' two staff members, one from each presentation. The 'Replace' and 'Move' type have three variants, and the 'Swap' type two variants. Overall we used the following $n_h = 8$ low-level heuristics.

1. *Replace one staff member in a session (N_1)*: This heuristic chooses a random staff member, say j_1, chooses a random session, say q during which staff j_1 is scheduled for presentations and replaces j_1 with another random staff member, say j_2, in all presentations involving staff j_1 during session q. Staff j_2 must not be involved in any presentations during session q prior to the substitution.
2. *Replace one staff member in a session (N_2) Version 2*: Same as previous heuristic but staff j_1 has the largest number of scheduled sessions.
3. *Replace one staff member in a session (N_3) Version 3*: Same as N_2 but session q is the one where staff j_1 has the smallest number of presentations. Also staff j_2 may be involved in presentations during session q prior to the substitution.
4. *Move a presentation from one session to another (N_4)*: This heuristic chooses a random presentation, removes it from its current session and reschedules it in another random session and a random room.
5. *Move a presentation from one session to another (N_5)Version 2*: Same as previous heuristic but the chosen presentation is that for which the sum of pre-

sentations involving all three staff (i.e. 1st marker, 2nd marker, observer) is smallest of all sessions.

6. *Move a presentation from one session to another (N_6)*: Same as N_5 but the new session is one where at least one of the staff members (i.e. 1st marker, 2nd marker, observer) is already scheduled for presentations.
7. *Swap 2nd marker of one presentation with observer of another (N_7)*: This heuristic chooses two random presentations and swaps the 2nd marker of the first presentation with the observer of the second presentation. The swap cannot involve the removal of a supervisor.
8. *Swap 1st marker of one presentation with 2nd marker of another (N_8)*: This heuristic chooses two random presentations and swaps the 1st marker of the first presentation with the 2nd marker of the second presentation. The swap cannot involved the removal of a supervisor.

For each of 'Replace' and 'Move' types of low-level heuristic the third version generally yields solutions of better quality than the two others. We shall see, later on, that the choice function hyperheuristic is capable of detecting this behaviour.

Both RD and CFa start with a solution produced by the constructive heuristic solution, *ch*, used in [73] and described earlier. The stopping condition was 600 seconds CPU. The results (averaged over 10 runs) correspond to the best value of E found during the search of each algorithm. We see that both algorithms produced results much better than *ch*. Also CFa gave better results than RD. We note that the gap in terms of objective E between CFa and RD is greatest with $csit0$ and smallest with $csit2$. It seems that CFa outperforms RD though the difference appears to decrease as the difficulty of the instances to solve increases. Furthermore, it was observed in [73] that finding a better solution becomes increasingly difficult as the search goes on. This suggests that there is an advantage in using CFa over RD. In [73] CFa also achieved results superior to those of RD when both methods started from a very poor-quality solution constructed manually for $csit0$ (called ml in Table 7.1). The hyperheuristic results were however inferior to those of *ch*. We decided to run CFa from two initial solutions of very poor quality. Both initial solutions (s_1 and s_2) were obtained randomly. In the lower part of Table 7.1 the objective is given for CFa after 2 hours of CPU time in order to see if CFa will get any closer to *ch*. The results (averaged over 3 runs) suggest that the area of the search space the initial solutions are at are so poor that it is difficult to quickly move to a good area. It should be noted however that CFa made a huge improvement on the initial solutions and is able to catch up and even overtake *ch* on instance csit0. Therefore it is still possible for CFa to reach good areas of the solution space that were quite remote.

The second set of experiments aimed at investigating the low-level behaviour of the choice function hyperheuristic. In Table 7.2, we give the proportion of call, by $CFa-AM$, of each low-level heuristic during the first 100 heuristic calls and during the last 100 heuristic calls to the best solution. We also rank the low-level heuristics according to their overall proportion of call so that the top (bottom) heuristic is the one that has been called most (least) often. Results are obtained after 30 minutes CPU of run in order to allow for a realistic sampling. From the proportion of calls during the 1st 100 calls it is clear that in the early stage of the search calls are spread fairly evenly over the low-level heuristics, as the hyperheuristic has not 'learned' which ones are best. Because $CFa-AM$ seems to be continually improving on the search, the last 100 heuristic calls to the best solution correspond to the last 100 heuristic calls of the search. It is interesting to note that not all low-level heuristics need be called at this later stage. Thus only low-level heuristics h_2, h_3,

h_5 and h_6 are needed for instances *csit*1 and *csit*2 whereas h_3 alone, which works towards improving on objective B[2], suffices for problem instance *csit*0. It seems that the choice function hyperheuristic shows different behaviours for different problem instances. This provides some evidence that the hyperheuristic is capable of learning about the interplay existing between the low-level heuristics dependent on both the problem being solved and the part of the search space currently being explored. From the overall proportion of calls we see that overall (across the 3 instances), heuristics h_2, h_3 and h_6 figure among the top 3 heuristics whereas heuristic h_1 is at the bottom. This can be regarded as a feature common to the 3 problem instances. As noted in [73] it seems that h_3 and h_6 which are the most sophisticated version of their category ('replace' type h_1, h_2, h_3 and 'move' type h_4, h_5, h_6) are likely to be called more often than the others. There was no plateau landscape during the hyperheuristic search on instances csit1 and csit2. For csit0 however a plateau of solutions evaluated at -1390.6 was identified. The 100 heuristic calls covering the plateau landscape were distributed as 0, 28, 35, 0, 5, 18, 2, 12 for heuristics h_1, h_2, .., h_7 and h_8 respectively. Comparing this to csit0 results in Table 7.2 we see a totally different low-level behaviour, which helped the hyperheuristic escape from the plateau by first accepting worse solutions (up to -1292.6) in order to reach out for good ones, ending up at -1414.6 (at the 100^{th} heuristic call).

As mentioned earlier, a comparison of the overall proportion of calls of the low-level heuristics in Table 7.2 within each of 'Replace' and 'Move' types shows that the third version for each type is called more often than each of the other two versions of that type, for all three instances. Thus N_3 is called by the choice function hyperheuristic more often than N_1 and N_2. Similarly, N_6 is called by the choice function hyperheuristic more often than N_4 and N_5. Overall the choice function

[2]Note that objective B has the largest coefficient in the objective function E.

	csit0	csit1	csit2
h_1	2/0, 7/0.006	4/0, 8/0.009	2/0, 8/0.005
h_2	25/0, 2/0.134	16/4, 3/0.118	31/6, 2/0.129
h_3	43/100, 1/0.691	16/76, 1/0.552	32/88, 1/0.672
h_4	5/0, 6/0.001	5/0, 7/0.013	10/0, 5/0.016
h_5	8/0, 5/0.041	7/2, 4/0.078	6/1, 4/0.038
h_6	10/0, 3/0.077	9/18, 2/0.126	10/5, 3/0.121
h_7	3/0, 6/0.001	28/0, 6/0.046	3/0, 7/0.009
h_8	4/0, 4/0.049	15/0, 5/0.058	6/0, 6/0.010
E	-1462.6	-2946.6	-1730.0

Table 7.2: heuristic calls by $CFa - AM$. Format: # calls during 1st 100 calls/last 100 calls to best solution, overall rank/overall proportion of call

hyperheuristic appears capable of detecting good low-level heuristics [70]. Findings from experiments in Table 7.2 suggest that there is a certain probability distribution of the low-level heuristics which a hyperheuristic should be able to find out if it is to be effective as in the Bayesian heuristic approach developed by Mockus et al. [198, 203] (see section 2.3.2). Experiments in the next paragraph investigate whether the choice function hyperheuristic is able to work out appropriate heuristic call frequencies.

Using the results in Table 7.2 we implemented an 'intelligent' random hyperheuristic, $RD1$, based on RD. Instead of selecting each low-level heuristic uniformly at random (i.e. equal probability of choice) $RD1$ chooses each low-level heuristic with a certain probability which corresponds to its overall proportion of call by the choice function hyperheuristic CFa[3]. The aim of the experiment was to see if the choice function hyperheuristic is able to choose appropriate heuristic call frequencies, and if so whether the choice function gives additional power by providing a better-than-random ordering of the low-level heuristics. $RD1$ 10-run average results

[3]For example when applying $RD1$ to instance *csit*1, heuristics h_1, h_2, h_3, h_4, h_5, h_6, h_7 and h_8 are chosen with probability 0.009, 0.118, 0.552, 0.013, 0.078, 0.126, 0.046 and 0.058 respectively.

can be found in the upper part of Table 7.1 (page 176). Both RD and $RD1$ give similar results on instance $csit1$. On instances $csit2$ and $csit0$ however, $RD1$, which uses heuristic call frequencies obtained from the choice function hyperheuristic outperforms RD, which simply chooses the low-level heuristics with equal probability distribution. It would appear that the choice function hyperheuristic is able to work out appropriate heuristic call frequencies. This is in line with the rationale behind the Bayesian heuristic approach [198, 203] which, too, aimed at finding out a good probability distribution of the low-level heuristics. Furthermore, while CFa and $RD1$ gave comparable results on instance $csit2$, CFa outperformed $RD1$ on $csit1$ and $csit0$. It seems that, in some cases, an approach which maintains an adaptive combination of the low-level heuristics (CFa), in which heuristic call frequencies vary, may appear to be more robust than one which keeps the same combination of the low-level heuristics ($RD1$), due to the ability of the former to adapt to changes in the search landscape (valleys, plateaux, ...). Therefore we think that the similarity of results between CFa and $RD1$ on instance $csit2$ is due to the fact that the area of the landscape we are at is somewhat smooth and so the current heuristic combination used in $RD1$ is good enough to cope with that. To further confirm this, and to see if CFa is nothing more than just a good random hyperheuristic (like $RD1$) we ran both $RD1$ and CFa using initial solutions s_1 and s_2. This has the effect of starting the search from a rather different area (different to that of ch). $RD1$ still uses the same probabilities, which were obtained by CFa with ch as initial solution. Results (averaged over 10 runs) are given in Table 7.3. When starting from s_1, CFa beats $RD1$ by 150 (in difference) on csit1 and by 934.12 on csit2. Both algorithms have comparable results on csit0 (small difference of only 18.51). When starting from s_2, $RD1$ and CFa have similar results (small difference of only 12.86) on csit1. $RD1$ beats CFa by 66.94 on csit2 and by 116.82 on csit0. Although we have mixed results, it should be recalled that $RD1$ was manually tuned using the weights obtained from CFa. At least it can be said that adaptively changing the probability of choice of the low-level heuristic during the search allows us to deal robustly with different problem instances and starting solutions in some cases. In other words in some cases, just having a 'magic' combination of the low-level heuristic is not enough ($RD1$). We must maintain an adaptive control on the way we combine the low-level heuristics in order to carry out an effective search. The choice function hyperheuristic appears capable of achieving this intelligently. This also means that the way the hyperheuristic works is quite different from a random search, however effective that random search is. It is interesting to note that the superiority of CFa over $RD1$ is greater on initial solution s_1 which is worse than s_2.

The sort of solutions produced by CFa appeared to be practical. As a result, the choice function hyperheuristic solution has been implemented by the school for this academic year 2001-2002. The school's timetabling officer described the results as 'excellent'. The school would now like to extend our project to pen-ultimate year students who also have to give presentations at the end of every academic year. The number of pen-ultimate year students is twice as large as that of final year students). Solving the current problem saved the timetable officer the equivalent of one-week of man hours, during the school's busiest period of the academic year.

7.4 Conclusions

We have investigated the low-level behaviour of a choice function hyperheuristic using an 'intelligent' tailor-made random hyperheuristic. It appears that the choice function hyperheuristic not only makes an effective and realistic combination of the

	csit0	csit1	csit2
RD1-AM(s_1)	-500.93	1665.77	4164.21
RD1-OI(s_1)	-481.84	1545.55	4273.03
CFa-AM(s_1)	-394.77	1450.55	3230.09
CFa-OI(s_1)	-482.42	1395.91	3350.64
RD1-AM(s_2)	-463.01	-1059.73	27.70
RD1-OI(s_2)	-427.38	-1028.04	-3.27
CFa-AM(s_2)	-346.19	-976.95	63.67
CFa-OI(s_2)	-345.60	-1040.09	76.06

Table 7.3: Comparison between CFa and $RD1$ with different initial solutions

low-level heuristics at hand but also shows some evidence of an ability to adapt this heuristic combination to both the problem being solved and the region of the search space currently under exploration. While much of the power of the hyperheuristic appears to come from selecting appropriate probabilities for calling low-level heuristics as in the Bayesian heuristic approach developed by Mockus et al. [198, 203], additional benefits are sometimes obtained by adaptively varying those probabilities so that they are tailored to the solution space and low-level heuristics.

Whilst the aggregated choice function hyperheuristic, CFa, produced better results than the decomposed one, CFb, for the project presentation scheduling problem, the opposite happened with the sales summit scheduling problem of the previous chapter. Indeed, CFa's results for the sales summit problem were even worse than those of the simple hyperheuristics. The reason why CFb did not outperform CFa here is probably due to the fact that CFb for the project scheduling problem would have to deal with more individual objectives than for the sales summit. The more individual objectives there are (i.e. the larger $|L|$ is), the more parameters α_l, β_l ($l \in \mathbf{L}$) would need to be managed. Thus convergence to a good solution for the choice function hyperheuristic search could slow down when in presence of a substantial number of individual objectives in E. It would appear that when there are too many single objectives (in our experience more than 3), CFb becomes rapidly intractable. In which case we would recommend CFa. This point will be confirmed in the next chapter where, in presence of only 2 objectives, CFb will greatly outperform CFa which often produced solutions even worse than those of the simple hyperheuristics.

We would like to emphasize the fact that the implementation of the hyperheuristic techniques for this problem was quite fast. In effect, all hyperheuristics presented here are 'standard' approaches which worked well for the sales summit scheduling problem in the previous chapter [69, 70, 72]. Indeed all that was needed was a set H of low-level heuristics to be input to the hyperheuristic black box. The way the hyperheuristic works is independent of both the nature of the low-level heuristics and the problem to be solved except for the objective function's value and CPU time which are passed from the low-level heuristics to the hyperheuristic. Whilst producing the hyperheuristic framework has taken over 18 months, using this framework took us only the equivalent of 101 hours of work (or two and a half weeks at 40 hours work per week) from understanding the problem to obtaining good hyperheuristic solutions.

Our model appeared to be quite realistic and was able to represent all practical requirements made known to us. The design of the low-level heuristics was quite natural and reflected some of the repair moves which the problem owner used before ('swap', 'replace', 'move'). The implementation of the solution was quite fast and results were presented in Microsoft Excel tables which were then posted on the school's internal website, so that both students and members of academic staff could easily make note of their own timetable. Having seen the project presentations happening we can confirm that the schedule operated smoothly throughout.

Chapter 8

Application to nurse scheduling

8.1 Introduction

In the two previous chapters we showed that our choice function hyperheuristics performed well when applied to two different real-world problems. In this chapter we compare our choice function hyperheuristic with two other methods independently developed by other authors. The application problem is that of scheduling nurses. We consider real-world instances of the nurse scheduling problem encountered at a major UK hospital. Nurse scheduling has been previously tackled using tabu search [89] and genetic algorithms [9]. We aim to demonstrate that hyperheuristics are not only readily applicable to a wide range of scheduling and other combinatorial optimisation problems, but that they can also provide very good quality solutions which are comparable to those of knowledge-rich sophisticated metaheuristics, while using less development time and simple, easy-to-implement low-level heuristics. We shall also investigate the sensitivity of our choice function hyperheuristic, using a variety of different sets of low-level heuristics of various quality levels. The remainder of this chapter is structured as follows. In section 8.2 we describe the application problem, that of scheduling nurses. This is followed in section 8.3 by computational experiments and section 8.4 presents our conclusions.

8.2 The nurse scheduling problem

8.2.1 Problem description

The problem is that of creating weekly schedules for wards containing up to 30 nurses at a major UK hospital. These schedules must respect working contracts and meet the demand in terms of number of nurses of different grades required for each day-shift and night-shift of the week, whilst being perceived to be fair by the nurses themselves. In any given week, nurses work either days or nights. The day is partitioned into two types of shift: 'earlies' and 'lates'. A full week's work typically includes more days than nights. For example, a full-time nurse works 5 days or 4 nights, whereas a part-time nurse works 4 days or 3 nights, 3 days or 3 nights, or 3 days or 2 nights. The problem can be decomposed into 3 independent stages [90]. Stage 1 uses a knapsack model to check if there are enough nurses to meet demand. Additional nurses are needed for stage 2 otherwise (thus stage 2 will always admit a feasible solution). Stage 2 is the most difficult and is concerned with the actual allocation of the weekly shift-patterns to each nurse. Then stage 3 uses a network flow model to assign those on day-shifts to 'earlies' and 'lates' (see [90] for further details). As in [89] and [9] we limit ourselves to the most difficult sub-problem in stage 2. The stage 2 problem is described as follows. Each possible weekly shift-pattern for a given nurse can be represented as a 0-1 vector of 14 elements, where the first 7 elements represent the seven days of the week (7 day-shifts) and the last

7 elements the corresponding 7 nights of the week (7 night-shifts). A '1'/'0' in the vector represents a day or night 'on'/'off'. For each nurse there is a limited number of shift-patterns corresponding to the number of combinations of the number of days s/he is contracted to work in a week. For example a full-time nurse contracted to work either 5 days or 4 nights has a total of $C_7^5 = 21$ feasible day shift-patterns and $C_7^4 = 35$ feasible night shift-patterns. There are typically between 20 and 30 nurses per ward, 3 grade-bands, and 411 different (full-time and part-time) shift-patterns. Based upon the nurses' preferences for the various shift-patterns, the recent history of shift-patterns worked, and the overall attractiveness of the shift-pattern, a penalty cost is associated to each assignment nurse-shift pattern, values of which were set after discussions with the hospital, ranging from 0 (ideal) to 100 (undesirable) -see [89] for further details.

8.2.2 Problem formulation

The stage 2 problem can then be formulated as follows. Our decision variables are denoted by x_{ij} which assume 1 if nurse i works shift-pattern j and 0 otherwise. Let parameters g, n, s be the number of grades, nurses and possible shift-patterns respectively. Parameter a_{jk} is 1 if shift-pattern j covers shift k, 0 otherwise. b_{ir} is 1 if nurse i is of grade r or higher, 0 otherwise. p_{ij} is the penalty cost of nurse i working shift-pattern j. S_{kr} is the demand of nurses of grade r or above on day/night (i.e shift) k. Finally $F(i)$ is the set of feasible shift-patterns for nurse i. We may then use the following mathematical formulation from [89, 9]:

$$Min \quad PC = \sum_{i=1}^{n} \sum_{j=1}^{s} p_{ij} x_{ij} \tag{8.1}$$

s.t.

$$\sum_{j \in \mathbf{F(i)}} x_{ij} = 1, \quad \forall i \tag{8.2}$$

$$\sum_{j=1}^{s} \sum_{i=1}^{n} b_{ir} a_{jk} x_{ij} \geq S_{kr}, \forall k, r \tag{8.3}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{8.4}$$

The objective is expressed in Equation (1) as that of minimising the overall penalty cost associated with the nurses' desirability for the shift-patterns. Constraints (2) express the idea that each nurse must work exactly one shift-pattern. Constraints (3) reflects the demand for nurses. Note that b_{ir} is defined in such a way that higher-grade nurses can substitute for those at lower grades if needed. The problem is NP-hard [9] and instances typically involve between 1000 and 2000 variables and up to 70 constraints. It was noted in [9] that the difficulty of a given instance depends upon the shape of the solution space, which in turn depends on the distribution of the penalty costs (p_{ij}) and their relationship with the set of feasible solutions. In this chapter, we consider 52 instances of the problem, based on three wards and corresponding to each week of the year. These 52 instances, as a whole, feature a wide variety of solution space landscapes ranging from easy problems with many low-cost global optima scattered all over the space, to very hard ones with few global optima and in some cases with relatively sparse feasible solutions [9]. Optimal solutions are known for each instance as the problem was solved in [8] using a standard IP package. However some instances required more than 15 hours of (Pentium II 200 Mhz PC) run-time. Further experiments with a number of descent methods using different neighbourhoods, and a standard simulated annealing metaheuristic were conducted unsuccessfully, failing to obtain feasibility [9]. The most successful method to date which works within the low CPU time available in practice is a tabu search metaheuristic [89] which uses chain-moves whose design and implementation were highly problem and instance specific. These moves relied on the way the different factors affecting the quality of a schedule were combined in the p_{ij} values [9]. In [9] a genetic algorithm which did not use the chain-moves was also used to solve the problem. Failure to obtain good solutions led to the use of a co-evolutionary strategy which decomposed the main population into several co-operative sub-populations. Knowledge on the problem structure was incorporated in both the way the sub-populations were built, and the way partial solutions were recombined to form complete ones. The co-evolutionary strategy is highly problem-specific, and, both tabu search of [89] and genetic algorithm of [9] are only applicable to problems with a similar structure.

The evaluation function (also known as the fitness function in the GA literature) used by our hyperheuristic distinguishes between 'balanced' and 'unbalanced' solutions [89, 9]. In effect, since nurses work either days or nights it appears that in order for a given solution to be feasible, (i.e enough nurses covering all 14 shifts at each grade-brand), the solution must have sufficient nurses in both days and nights independently of one another[1]. Formally, a solution is balanced in days (or nights) at a given grade-band r if there are both under-covered and over-covered shifts in the set of days (or nights) at grade r such that the nurse surplus in the over-covered day (or night) shifts suffices to compensate for the nurse shortage of the under-covered day (or night) shifts. Clearly, a solution cannot be made feasible until it is balanced [89, 9]. We define [72]

$Infeas = \sum_{r=1}^{g}(\rho \times Bal_r + 1) \sum_{k=1}^{14} max\left(\left[S_{kr} - \sum_{i=1}^{n}\sum_{j=1}^{s} b_{ir}a_{jk}x_{ij}\right], 0\right),$

[1]Recall that nurses work either days or nights, but not both, in a given week.

where Bal_r is 2 if both day and night are unbalanced at grade r, 1 if either day or night is unbalanced at grade r, and 0 otherwise; ρ is a severity parameter for unbalanced solutions, whose value is chosen so that a balanced solution with more nurse-shortages is preferred to an unbalanced one with fewer nurse-shortages, as the latter is more difficult to make feasible than the former. We chose $\rho = 5$ as given in [72]. Based on this, we define the evaluation function
$E = PC + C_{demand} InFeas$
with C_{demand} a weight associated to $InFeas$ as in [9]. The definition of C_{demand} is based on the number, q, of nurse-shortages in the best least-infeasible solution so far, i.e.
$q = \sum_{k=1}^{14} \sum_{r=1}^{g} max\left(\left[S_{kr} - \sum_{i=1}^{n} \sum_{j=1}^{s} b_{ir} a_{jk} x_{ij}\right], 0\right)$.
Coefficient C_{demand} of $InFeas$ in E is then given by $C_{demand} = \gamma \times q$ if $q > 0$, and $C_{demand} = v$ otherwise; where γ is a preset severity parameter, and v is a suitably small value. The idea is that the weight C_{demand} depends on the degree of infeasibility in the best least-infeasible solution encountered so far, after which it remains at v. We use $\gamma = 8$ and $v = 5$ as given in [9]. [9] contains a further interesting discussion on the choice of evaluation functions. Note that while in [9] unbalanced solutions are avoided during the search through the use of incentives/disincentives to reward/penalise balanced/unbalanced individuals in the population, they are instead repaired in [89]. Here we opt for the latter approach and use the same 'balance-restoring' low-level heuristic used in tabu search of [89]. While this low-level heuristic is specific to this problem, it only uses a 'change' and a 'swap' type of move as described in section 4. We next describe our experimental findings.

Instances	CFb	Direct GA [9]	Indirect GA [9]	Tabu search [89]	Optimal cost[8]
Week 1	1/8	1/0	1/0	1/0	1/0
Week 2	1/52.8	1/12	1/12	1/11	1/11
Week 3	1/50	1/18	1/18	1/18	1/18
Week 4	1/17	1/0	1/0	1/0	1/0
Week 5	1/11	1/0	1/0	1/0	1/0
Week 6	1/ 2	1/1	1/1	1/1	1/1
Week 7	1/13.55	0.5/13	1/11	1/11	1/11
Week 8	1/14.95	1/11	1/11	1/11	1/11
Week 9	1/3.6	0.95/3	1/3	1/3	1/3
Week 10	1/5.05	1/1	1/2	1/1	1/1
Week 11	1/2	1/1	1/1	1/1	1/1
Week 12	1/2	1/0	1/0	1/0	1/0
Week 13	1/2	1/1	1/1	1/1	1/1
Week 14	1/3.15	1/3	1/3	1/3	1/3
Week 15	1/3.05	1/0	1/0	1/0	1/0
Week 16	1/40.1	0.95/25	1/25	1/24	1/24
Week 17	1/17.6	1/4	1/4	1/4	1/4
Week 18	1/20.85	1/7	1/6	1/7	1/6
Week 19	1/1.6	1/1	1/1	1/1	1/1
Week 20	1/15.45	0.95/5	1/4	1/4	1/4
Week 21	1/0	1/0	1/0	1/0	1/0
Week 22	1/25.5	1/1	1/1	1/1	1/1
Week 23	1/0	0.95/0	1/0	1/0	1/0
Week 24	1/1	0.75/1	1/1	1/1	1/1
Week 25	1/0.4	1/0	1/0	1/0	1/0
Week 26	1/48	0.1/0	1/0	1/0	1/0

Table 8.1: Hyperheuristic and metaheuristic performance on the nurse scheduling problem. For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost for feasible solutions.

Instances	CFb	Direct GA [9]	Indirect GA [9]	Tabu search [89]	Optimal cost[8]
Week 27	1/3.65	1/2	1/3	1/2	1/2
Week 28	1/65.8	1/1	0.95/1	1/1	1/1
Week 29	1/15	0.35/3	1/1	1/2	1/1
Week 30	1/39.4	1/33	1/33	1/33	1/33
Week 31	1/66.9	0.8/66	1/36	1/33	1/33
Week 32	1/41.6	1/21	1/21	1/20	1/20
Week 33	1/10.6	1/12	1/10	1/10	1/10
Week 34	1/42.9	1/17	1/16	1/15	1/15
Week 35	1/38.8	1/9	1/11	1/9	1/9
Week 36	1/34.85	1/7	1/6	1/6	1/6
Week 37	1/8.05	1/3	1/3	1/3	1/3
Week 38	1/13.3	1/3	1/0	1/0	1/0
Week 39	1/5.1	1/1	1/1	1/1	1/1
Week 40	1/9.35	1/5	1/4	1/4	1/4
Week 41	1/61.3	0.95/27	1/27	1/27	1/27
Week 42	1/47.55	1/5	1/8	1/5	1/5
Week 43	1/27.35	0.9/8	1/6	1/6	1/6
Week 44	1/31.75	0.9/45	1/17	1/16	1/16
Week 45	1/5.35	1/0	1/0	1/0	1/0
Week 46	1/9.4	0.7/6	1/4	1/3	1/3
Week 47	1/3.3	1/3	1/3	1/3	1/3
Week 48	1/6.05	1/4	1/4	1/4	1/4
Week 49	1/30.4	1/26	0.7/25	1/24	1/24
Week 50	1/109.25	0.35/38	0.8/36	1/35	1/35
Week 51	1/74.3	0.45/46	1/45	1/45	1/45
Week 52	1/62.2	0.75/63	1/46	1/46	1/46
Average	1/23.5	0.91/10.8	0.99/9.0	1/8.8	1/8.7
Run time	< 60 sec	15 sec	10 sec	30 sec	up to hours

Table 8.2: Hyperheuristic and metaheuristic performance on the nurse scheduling problem (continued). For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost for feasible solutions. Note that the average is given over the 52 problem instances.

8.3 Experiments

Both our hyperheuristics and low-level heuristics were coded in Microsoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running under Microsoft Windows 2000 version 5. In order to compare our results with those of TS and GA, all our hyperheuristics start with a solution generated randomly by assigning a random feasible shift-pattern to each nurse as in [89] and [9]. This is given by the following pseudocode.

```
Do
    Choose a random nurse, $i$, who has not been assigned a shift-pattern
    Choose a random feasible shift-pattern, $j$, and assign to nurse $i$
Until all nurses have been assigned a shift-pattern
```

This generates a very bad solution which is rarely feasible.

All results were averaged over 20 runs (results were not significantly different, often results were identical in more than 15 runs). We first give the 11 low-level heuristics used in the TS algorithm of [89].

h1 : Change the shift-pattern of a random nurse.

h2 : Same as [h1] but 1st improving $InFeas$.

h3 : Same as [h1] but 1st improving $InFeas$ and no worsening of PC.

h4 : Same as [h1] but 1st improving PC.

h5 : Same as [h1] but 1st improving PC and no worsening of $InFeas$.

h6 : Change the shift-pattern type (i.e from day to night or vice versa) of a random nurse, if solution unbalanced.

h7 : Same as [h6] but the aim is to restore balance. That is from day to night if night is unbalanced and vice-versa. If both days and nights are unbalanced a swap of shift-pattern type for a pair of nurses, one working days and the other working night is considered. The nurse working day is assigned a night shift-pattern and the nurse working night is assigned a day shift-pattern.

h8 (shift-chain1): This heuristic considers chains of moves aiming at decreasing both the nurse-shortage in one (under-covered) shift and the nurse-surplus in one (over-covered shift), and leaving the remaining shift unchanged. The chain-moves are defined as paths in a graph. The move is only attempted if the solution is already balanced but not yet feasible.

h9 (nurse-chain1): This heuristic considers chains of moves which move the first nurse in the chain to cover an under-covered shift and move the subsequent nurses to the shift-pattern just vacated by their predecessor in the chain. [h9] chain-moves are also defined as paths in a graph. The move is only attempted if the solution is already balanced but not yet feasible.

h10 (shift-chain2): This heuristic considers a shift-chain of moves aiming at decreasing the penalty cost when the solution is already feasible. Chains are represented as cycles in a graph.

h11 (nurse-chain2): This heuristic considers nurse-chains of moves aiming at decreasing the penalty cost when the solution is already feasible. Here, too, chains are represented as cycles in a graph.

Instead, our choice function hyperheuristic used 9 low-level heuristics including the first 7 low-level heuristics above, all of which are relatively simple, and the following:

H8 (Change-and-keep1): This heuristic finds the first move which improves PC by changing the shift-pattern of a nurse and assigning the removed shift-pattern to another nurse.

H9 (Change-and-keep2): Same as [H8], but only considers moves which do not worsen $InFeas$ [2]

Heuristics [h8], [h9], [h10] and [h11] from [89] are highly effective problem-specific moves which were responsible for both feasibility (using shift-chain1 and nurse-chain1) and optimality (using shift-chain2 and nurse-chain2) of the solution in most cases in [89]. TS can only yield good solutions when equipped with such moves [8, 9]. Indeed TS was able to produce optimal solutions for many instances of the problem. However, as noted in both [8] and [9] these moves are highly problem-specific and, in fact, instance-type specific. Unlike in TS, the low-level heuristics used by the choice function hyperheuristic are fewer and much simpler than the chain-moves. They are all based around changing, or swapping one or two shift-patterns, thus reflecting what users usually do when manually solving the problem [69]. In Table 8.1 and Table 8.2, we give results of our choice function hyperheuristic, along with those of both the direct and indirect GA [8, 9] as well as TS [89] and the IP optimal solution [8] for each of the 52 weeks (data sets) of the year. The choice function results are those produced by CFb. Preliminary experiments using CFa were very unsuccessful with results often worse than those of the simple hyperheuristics (both in terms of feasibility and cost). All our hyperheuristic results used the AM version (though feasibility was achieved in most cases, the OI version produced a cost between 10 and 40 times worse than the AM one - OI gets stuck quickly in local optima). The stopping condition of CFb is 6000 iterations, which corresponds to a CPU time

[2]At this point, the reader might wonder what results are obtained without [H8] and [H9]. This is discussed in the last paragraph of this section.

between 44 and 60 seconds on a Pentium II 1000Mhz. Stopping condition for TS was 1000 moves without overall improvement. The stopping condition for the GA was 30 generations without improvement (in each generation, the population size was 100). Also the GA was coded in Turbo Pascal under MS-DOS, which resulted in low CPU times (Some instances were solved within 5 seconds CPU using the GA [8]). We see that for all instances CFb is able to find feasible solutions in each of 20 runs. It appears that the choice function hyperheuristic is more reliable than both the direct and the indirect GA in terms of producing practical solutions for the hospital (although it could be argued that *CF* is given more CPU time than the GA, the amount of CPU time used by *CF* - 60 seconds CPU - is not considerably large). To confirm this, we ran CFb on instance 50 (which is a difficult instance for both GA's and appeared to be the most difficult for CFb) 100 times and feasibility was again achieved for every single run, within 6000 iterations (less than a minute of CPU time). From this point of view, the hyperheuristic appears to be as robust as TS which, too, always produced feasible solutions. CFb however has the highest average cost of 23.5, though more than half of the instances (27 instances) were solved to within 10% of the optimal solution, including 3 instances where optimality is achieved on each of 20 runs. These hyperheuristic solutions are of acceptable quality according to the standard of [89, 9]. Moreover, in 9 instances the optimal solution is hit up to 19 times out of 20 runs, corresponding to a probability (frequency) of optimality of 0.95. This shows that optimal solutions are indeed, within the reach of CFb despite its simplicity and that of the low-level heuristics that CFb used, when compared with the problem and instance-specific information used by both TS (chain-moves) and GA (population decomposition and recombination using problem structure) implementations. In terms of cost, we noted that the hyperheuristic performed well for instances with slack demand-constraints and poorly for those

Instances	SR	RD	RP	RPD	CFb-Low	CFb	CFb-High
Week 1	1/9.45	1/20.55	1/14.3	1/20.7	**1/8**	**1/8**	**1/8**
Week 2	1/63.35	0.95/68.57	1/69.6	0.95/67.42	1/53.85	**1/52.8**	1/54.3
Week 3	1/58.45	0.95/72	1/66.85	1/66.85	1/50.1	**1/50**	**1/50**
Week 4	**1/17**	1/24.25	1/17.4	1/23	**1/17**	**1/17**	**1/17**
Week 5	1/19.85	0.95/24.21	1/28.35	1/23	**1/11**	**1/11**	**1/11**
Week 6	1/3	1/10.5	1/4	1/12.15	1/2.05	**1/2**	1/2.55
Week 7	1/38.3	0.95/32.10	1/50.8	0.85/36.7	1/18	**1/13.55**	1/16.95
Week 8	1/27	1/32.65	1/31.7	1/33.45	1/15.8	**1/14.95**	1/16.65
Week 9	1/26.75	1/13.55	1/37.65	0.95/17.57	1/7.95	**1/3.6**	1/6.8
Week 10	1/12.75	0.65/30.92	1/16.2	0.65/25.23	0.95/5.78	**1/5.05**	1/5.75
Week 11	1/5.6	1/25.65	1/11.25	1/23.8	1/2.8	**1/2**	1/2.7
Week 12	1/3.55	1/17.6	1/5.4	1/15.2	1/2.5	**1/2**	1/2.65
Week 13	1/ 2.3	0.95/19.21	1/3.2	1/14.55	1/2.2	**1/2**	1/2.4
Week 14	1/10.05	0.8/31.68	1/17.7	0.65/32.69	1/5.35	**1/3.15**	1/7.1
Week 15	1/8.85	0.9/43.72	1/9.55	0.75/36	1/4.05	**1/3.05**	1/5.6
Week 16	1/61.9	0.95/146.31	1/70.5	0.9/155.83	1/51.1	**1/40.1**	1/50.7
Week 17	1/54.35	0.3/80.33	0.95/72.52	0.7/68.21	1/29	**1/17.6**	1/29.75
Week 18	1/41	0.05/40	1/49.2	0.15/7	1/33.2	**1/20.85**	1/46.6
Week 19	1/14.25	1/57.75	1/20.15	0.95/75.84	1/3.6	**1/1.6**	1/5.15
Week 20	1/33.8	0.85/42.94	1/44.15	1/41.8	1/16.5	**1/15.45**	1/15.7
Week 21	1/1.35	0.75/12.66	1/4	0.8/16.56	1/0.55	**1/0**	1/0.95
Week 22	1/29.9	0.8/51.18	1/35.35	0.75/45.4	1/29.9	**1/25.5**	1/29.15
Week 23	1/1.4	0.35/19.71	0.85/2.52	0.35/12.28	1/1.85	**1/0**	1/1.9
Week 24	1/4.85	0.4/44.25	0.9/9.16	0.45/38.11	1/5.2	**1/1**	1/8.65
Week 25	1/0.6	0.95/22.36	1/1.5	0.75/15.26	**1/0.25**	1/0.4	1/1.1
Week 26	1/60.1	0.45/171	0.85/66.41	0.15/174.33	1/55.3	**1/48**	1/53.2

Table 8.3: Choice function vs simple hyperheuristics applied to the nurse scheduling problem. For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost.

with tight constraints. It should be noted that from the hospital's point of view, feasibility is more important than cost as demand must be satisfied. The issue of cost is only considered *after* we have ensured that there is a sufficient number of nurses for each of the 14 shifts and for each grade-band.

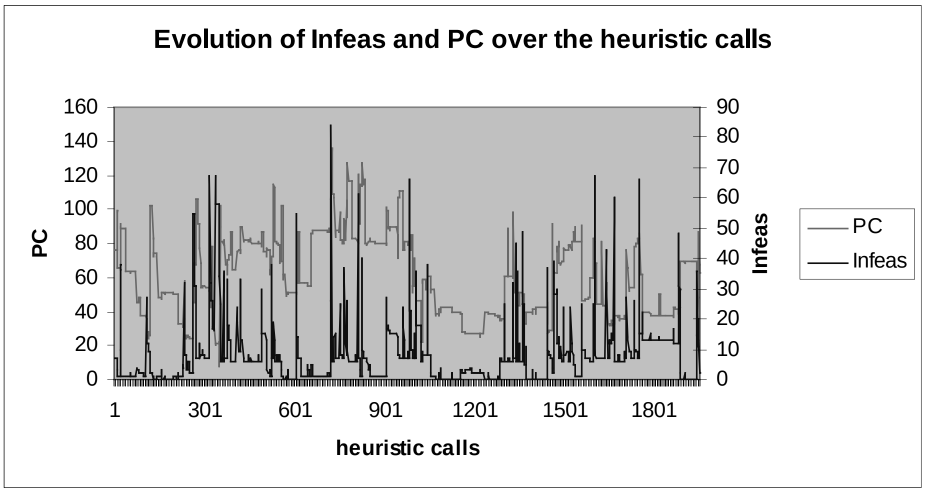
Analysing the frequency of call of the low-level heuristics showed that [h2] is called most often (e.g 37% on average for Week 49), followed by [h6] (e.g 10% on Week 49) and all other heuristics are called between 5% and 9%. It appears that each low-level heuristic has a part to play in the search [89]. Observations of the variation

Instances	SR	RD	RP	RPD	CFb-Low	CFb	CFb-High
Week 27	1/26	0.65/38	1/36.35	0.6/45.16	1/23.45	**1/3.65**	1/10.25
Week 28	1/76.3	0.45/84.11	1/82.25	0.3/82	1/66.9	**1/65.8**	1/69.05
Week 29	1/23.7	0.3/35.83	0.85/24.41	0.4/36.5	1/17.1	**1/15**	1/16.95
Week 30	1/56	0.95/71.73	1/61.1	0.95/76.21	1/42	**1/39.4**	1/42.3
Week 31	1/91.8	0.95/82.84	1/102.3	1/87.05	1/74.5	**1/66.9**	1/72.4
Week 32	1/52.5	1/60.3	1/56	1/56.85	1/44.2	**1/41.6**	1/44.15
Week 33	1/13.15	0.95/25.84	1/13.7	1/23.55	1/13.3	**1/10.6**	1/13.2
Week 34	1/77.45	1/77.6	1/93.05	1/63.8	1/47.75	**1/42.9**	1/45.85
Week 35	1/58.35	1/63	1/60.95	1/67	1/39.15	**1/38.8**	1/42.15
Week 36	1/60.7	1/45.05	1/69.3	1/46.05	1/42.15	**1/34.85**	1/39
Week 37	1/17.1	1/34.4	1/22.05	0.95/30.10	1/10.2	**1/8.05**	1/8.9
Week 38	1/21.9	1/34.95	1/25.7	1/40.1	1/20.8	**1/13.3**	1/16.8
Week 39	1/8.7	1/18.05	1/13.35	1/21.8	1/6.55	**1/5.1**	1/7.25
Week 40	1/22.5	0.95/48.21	1/30.8	0.95/34.73	1/12.55	**1/9.35**	1/17.95
Week 41	1/91.2	0.8/160.56	1/99.9	0.95/168.94	1/70.8	**1/61.3**	1/73.9
Week 42	1/97.2	0.6/109.58	1/106.9	0.55/127.72	1/60.1	**1/47.55**	1/65
Week 43	1/53.4	0.15/101.33	1/63.85	0.1/42	1/37.95	**1/27.35**	1/36.05
Week 44	1/59.25	1/53.7	1/63.85	1/64.15	1/35.85	**1/31.75**	1/36.4
Week 45	1/9.15	0.9/27.61	1/11.7	0.9/48.77	1/9.8	**1/5.35**	1/6.95
Week 46	1/32.25	0.35/33.42	0.75/32.53	0.3/36.5	1/17.6	**1/9.4**	1/21.6
Week 47	1/15.35	0.25/38.4	0.7/26.35	0.55/79.72	1/8.3	**1/3.3**	1/10.5
Week 48	1/14.9	0.9/18.88	1/19.15	0.85/23.58	1/10.3	**1/6.05**	1/10.45
Week 49	1/48.75	0.45/64.33	1/54.45	0.55/71.45	1/36.5	**1/30.4**	1/39.55
Week 50	1/120.55	0.25/126	1/120.4	0.2/159.75	1/113.35	**1/109.25**	1/115.25
Week 51	1/84.65	0.4/94.62	0.9/90.22	0.45/85	1/76.65	**1/74.3**	1/75.35
Week 52	1/77.35	0.5/88.5	1/83.75	0.45/92.22	1/72.55	**1/62.2**	1/72.1
Average	1/36.92	0.76/53.70	0.97/42.76	0.76/54.03	0.99/27.75	**1/23.54**	1/28.10

Table 8.4: Choice function vs simple hyperheuristics applied to the nurse scheduling problem (continued). For each problem instance the format is: proportion of feasible solutions in 20 runs/ average cost. Note that the average is given over the 52 problem instances.

of $Infeas$ and PC during the hyperheuristic search showed (Upper chart of Figure 8.1) that, immediately upon finding a feasible solution (i.e when $InFeas = 0$) there was a sudden increase in PC. Similar remarks mere made in [89]. This behaviour reflects the tight relation between Cost and Feasibility for this highly-constrained problem. In terms of choice function parameters, the hyperheuristic search used a very high δ and a low α and β, thus confirming the need to diversify the search quite frequently, due to the sparse spread of good solutions in the landscape [9]. This was in total agreement with the graph of the variation of $InFeas$ over time (Lower chart of Figure 8.1) which featured sudden low peaks of $Infeas = 0$, similar to the 'comb' shape graph of the same function in [89]. Typically values of $InFeas = 0$ never lasted more than 41 heuristic calls (compared to a total of 10000 heuristic calls overall) after they were obtained. Values for α_{InFeas} and β_{InFeas} were relatively higher than those of α_{PC} and β_{PC} clearly showing the greater importance attached to feasibility over lowering PC.

Having demonstrated the robustness of the choice function hyperheuristic in terms of solution quality, we now investigate its robustness when compared to the simple hyperheuristics described in section 5.2. The aim is to see if CFb does anything better than a simple random choice of the low-level heuristics. To do this we ran all 4 simple hyperheuristics on the same 52 problem instances (all moves are accepted). To allow for a fair comparison the stopping condition was 7000 iterations for RD and RPD and 11000 iterations for SR and RP, which again correspond to an average run time of 60 seconds CPU. Results, averaged over 20 runs, are given in Table 8.3 and Table 8.4. We see that RD and RPD perform poorly in terms of both feasibility and cost when compared to SR and RP. While in both SR and RP the low-level heuristics are applied once, in both RD and RPD they are applied in a steepest descent fashion. Because of the complexity of the problem, it seems



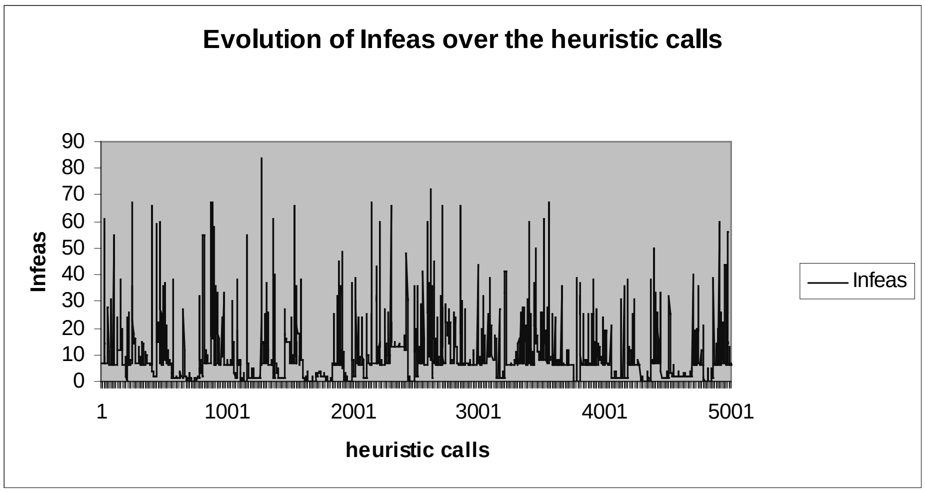


Figure 8.1: Evolution of Feasibility and Cost (Upper chart) and Feasibility only (Lower chart) over the number of heuristic calls during the choice function hyper-heuristic search, when applied to instance 49. The Upper chart goes from heuristic call 2550 to 4500.

that attempting moves in a descent fashion is a ‘waste’ of time (unless if done in an appropriate manner as in CFb). The level of improvement of the solution when applying a heuristic in a descent fashion is not worth the amount of time needed as we get stuck in poor local optima. Of all four simple hyperheuristics, SR gave the best results. Note that SR always found a feasible solution, which is thus even better than both the direct and the indirect GA. However, SR produced solutions of higher cost than those of CFb. We conclude that the choice function hyperheuristic conducts a more effective search than the simple hyperheuristics. Our experience with this problem suggests that high diversity is useful in order to get good solutions. This was built in SR and is well maintained in CFb.

We also carried out further experiments with CFb when in presence of various levels of quality of the sets of low-level heuristics. The aim of these experiments is to see how sensitive the choice function hyperheuristic, CFb, is with regards to the quality of the low-level heuristics under its management. The low-level heuristics given above can be classified into three groups of quality:

1. *Low*: $h1, h2, h3, h4, h5, h6, h7$.
2. *Medium:* $h1, h2, h3, h4, h5, h6, h7, H8, H9$
3. *High*: $h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11$

We applied our choice function hyperheuristic to each of these 3 sets of low-level heuristics. Note that this results in CFb when using set *Medium.* Also note that set *High* corresponds to the TS low-level heuristics in [89]. Results (60 seconds CPU time, average over 20 runs) are given in Table 8.3 and Table 8.4. We note that CFb*-Low*, which uses the poorest set of low-level heuristics does not perform too badly when compared to CFb, which uses *Medium.* To our great surprise, CFb*-Low* performed better than the simple hyperheuristics, which did employ the same low-level heuristics as *CFb-Medium*. Also perhaps surprising is the fact that *CFb-High*, which used the highest quality set, did not perform better than *CFb*. This might be due to the fact that *CFb-High* uses a larger number of low-level heuristics as experiments with large sets of low-level heuristics suggest in chapter 6. In addition, it seems that the use of *Medium*, combined with the power of the choice function hyperheuristic, is good enough to reach solutions of quality similar to those of the sophisticated problem-specific chain-moves used in TS [89]. This suggests a certain robustness of the choice function hyperheuristic as far as solution quality is concerned. This also suggests that the choice function hyperheuristic is very robust in terms of quality of the low-level heuristics used. Hence, from a paradigmatic point of view, it is not necessary to have very sophisticated low-level heuristics in order to produce good solutions using a hyperheuristic. It also should be noted that *CFb* cannot use sophisticated low-level heuristics, which is bad, but we would not generally use them in a hyperheuristic framework.

8.4 Conclusions

We have applied a choice function hyperheuristic to an NP-hard problem of scheduling nurses at a major UK hospital, for which even finding a feasible solution is difficult. The problem was previously solved using tabu search [89] and two genetic algorithms [8, 9]. In terms of solution feasibility, our choice function hyperheuristic proved to be more reliable than both the direct and indirect genetic algorithms and appeared to be as robust as tabu search. In terms of cost, more than 50% of the problem instances were solved within 10% of optimality, despite the simplicity of the hyperheuristic and that of the low-level heuristics employed, when compared to the highly problem-specific information utilised in both tabu search and the genetic algorithms. Also the choice function hyperheuristic appeared much stronger than simple hyperheuristic methods which choose the low-level heuristics at random. Unfortunately our hyperheuristic was not able to use the sophisticated low-level heuristics of the tabu search approach of [89], which were included in a large set of low-level heuristics.

Because of their problem-specific considerations, both tabu search and the genetic algorithm implementations for this problem can only be re-used for problems with a similar structure. However, our hyperheuristics are generic methods that are easily re-usable for other problems and other domains as demonstrated in the two previous chapters. A sensitivity analysis of the choice function hyperheuristic was carried out and revealed a high level of robustness of the method over various levels of quality of the low-level heuristics used. This means that from a pragmatic point of view, it is not necessary to have very sophisticated low-level heuristics in order to produce good solutions using a hyperheuristic.

Moreover, the hyperheuristic needs no parameter tuning [70]. Hyperheuristics are easy-to-implement and require less domain knowledge than most other heuristic approaches, yet still are capable of producing good-quality solutions even for very difficult problems within a reasonable amount of CPU and implementation time (approximately 6 weeks for problem analysis and evaluation function, coding of low-level heuristics, especially the sophisticated ones, and the handling of large data sets), much lower than that needed for tabu search and genetic algorithms (approximately 11 weeks according to the author of [89]).

Chapter 9

Conclusions

We have developed a number of hyperheuristic methods and applied them to various instances of three very different personnel scheduling problems taken from the real world. Hyperheuristic approaches were applied as early as the 1960's (see chapter 2). However, relatively little work has been carried out in this area in the intervening years. The last few years have seen an increasing interest in this re-emerging field of heuristic / search development (e.g. [48, 52, 69, 73, 72, 103, 139, 198, 195, 213, 211, 210, 234, 245, 259]). In this thesis, we have evaluated the performance of our hyperheuristics over the range of problems and problem instances considered.

As mentioned in chapters 1 and 2, in order to apply a hyperheuristic to a given problem we only need a set of low-level heuristics and a formal means for evaluating solution quality. The basic idea of hyperheuristics is easy to grasp. Such approaches are easy to implement because of the higher level of abstraction than current meta-heuristic applications and implementations. Thus we were able to apply the same standard hyperheuristics to three different problems. All we needed for each of these applications was to code some low-level heuristics and an evaluation function [158].

All hyperheuristics presented in this thesis produced excellent results. This was the case particularly for both the sales summit scheduling problem (chapter 6) and the project presentation scheduling problem (chapter 7). One class of hyperheuristic of particular interest is that which makes use of a choice function. The choice function provides an adaptive ranking of the low-level heuristics based upon which the hyperheuristic makes an appropriate heuristic selection. For all three problems the choice function hyperheuristic produced results better than those obtained from simple and random hyperheuristics such as VND [194, 137] which used multiple-neighbourhood search techniques.

An extensive investigation of the choice function hyperheuristic in chapter 6 (sales summit) suggests that the method is effective in terms of solution quality. Particularly the choice function compared favourably with both a simulated annealing hyperheuristic and a simulated annealing metaheuristic which used complex neighbourhood structures. The choice function proved to be robust for different sizes of the set of low-level heuristics managed. Since this was the first problem studied, development time was high and it is hard to separate the design/development time for the hyperheuristic from that which was problem specific (modelling, low-level heuristics for the sales summit scheduling problem).

Using the project presentation scheduling problem, we showed that solution development time using a hyperheuristic can be dramatically reduced while producing solutions of acceptable quality. In effect, our choice function hyperheuristics are ‘standard’ approaches which were successfully applied to all three problems. A further investigation of the choice function hyperheuristic showed that this type of hyperheuristic does a search different to a pure random choice of the low-level heuristics. We observed that the choice function hyperheuristic showed a behaviour superior to that of a specially-tailored random hyperheuristic. Some of the power of the choice function hyperheuristic seems to come from its adaptive nature which takes into consideration both the part of the search space currently under exploration and the performance of the low-level heuristics under its management. Our choice function hyperheuristic is also capable of finding the right frequency of heuristic calls.

Tackling a difficult real-world problem of scheduling nurses we showed that the hyperheuristic approach is capable of producing good solutions in a shorter development time (despite using simple low-level heuristics) when compared to two sophisticated metaheuristic methods developed by other researchers. In terms of feasibility of the solution, the choice function hyperheuristic appeared to be superior to two genetic algorithms and as robust as tabu search. In terms of cost of the solution, the hyperheuristic performed poorly when compared to both methods though over half of the instances were solved by the hyperheuristic to within 10% of optimality. A further investigation of the choice function hyperheuristic using various levels of quality of the low-level heuristics showed that our method is also qualitatively robust. This is a finding of significant importance as it follows that it is not necessary to use high-quality low-level heuristics in order to obtain good results using a hyperheuristics. Simple low-level heuristics, which is quicker to implement should suffice. Our choice function hyperheuristic could not use sophisticated low-level heuristics, but can generate good results anyway without them. It would usually be preferable to not have to design/develop these complex heuristics.

All three problems considered in this thesis are quite different from one another. The project presentation scheduling problem in chapter 7 features a large search space. This is due to the combinatorial number of possible combinations of students with three members of academic staff for each project presentation, to which a large number of room-timeslot allocations is possible. This results in a smooth solution landscape which makes the search for a good solution easy but time-consuming. On the other hand, the nurse scheduling problem in chapter 8 is of a fairly small size. There are fewer solutions to search, but the solution landscape is very *ragged* [89, 9]. Even finding a feasible solution is difficult and required the use of sophisticated chain-moves in a tabu search framework [89]. In between these two problems, we have the sales summit scheduling problem of chapter 6. For this problem, the search space is smaller than that of the project scheduling but larger than that of the nurse scheduling problem. This means that there are fewer solutions to search than for the project scheduling problem but more than for the nurse scheduling problem. Having achieved good results across the range of these three problems we have gathered evidence that hyperheuristics are generally applicable to a wide range of problems (at least problems of the same domain).

We noted in chapter 8 that our choice function hyperheuristic performed poorly for instances of the nurse scheduling problem with tight constraints. This can be fixed using better-than-basic low-level heuristics which cover wider parts of the search space and thus can reach places which are beyond reach otherwise.

We suspect that hyperheuristics (at least in the way that they are developed and presented in this thesis) might not be suitable for dynamic problems. This is due to the fact that in our choice function of chapter 5, we assumed that if a low-level heuristic has been performing well 'recently', it might perform well if applied again. If the problem is not static, there is a good chance that this assumption may no longer hold. Based on our experience, we would recommend the use of a relatively low number of low-level heuristics (between 5 and 10). We would also recommend the use of individual choice functions CFb (i.e. equation 5.7) for problems with less than 4 individual objectives and the use of the aggregated choice function CFa (i.e equation 5.6 or 5.8) in other circumstances.

Overall, our choice function hyperheuristic produced good results for all three real-world problems. The way the hyperheuristic chooses the low-level heuristic is problem-independent. This results in a method which is easily re-usable for other problems and domains. Thus, by using hyperheuristics we are able to raise the level of generality at which most current metaheuristic studies operate. This has significant benefits especially for industrial applications. In effect, most industrial optimisation problems need to be solved quickly enough. Problem instances come in different varieties and bespoke solution methods are often difficult to re-use for other problems which they (the solution methods) were not developed for [48]. This thesis provides evidence that hyperheuristics can produce solutions to a variety of problems *soon-enough, good-enough, cheap-enough.* The potential for scientific progress in the development of hyperheuristics, for a wide variety of application areas, is significant [48].

We recommend three directions for future work:

1. We experimented with different expressions of the choice function and it turned out that each factor - first and second order improvement and exploration function - of the choice function that we considered contributed to an effective search. Thus, in particular, the second order improvement factor, which represents the joint performance of a pair of heuristics, proved to be useful and, if sufficient iterations are possible, we might consider triples or m-tuples of heuristics as well. The issue here is how could enough data points be gathered as we would need more heuristic calls? This would be resolved with increasing processor

power and memory capacity (in order to store the resulting m-matrix).

2. In this thesis we have investigated a choice function hyperheuristic. It would be useful to experiment with other types of choice functions and other types of hyperheuristics using other learning mechanisms. Design issues for hyperheuristics were extensively discussed in chapter 5. There is a great deal of design choices possible and therefore great scope for research. Recent investigations by other scholars produced encouraging results. This is the case in [234] using learning classifier systems, in [68, 135] using genetic algorithms, in [211, 210] using reinforcement learning techniques, and in [51, 220, 52] using case-based reasoning techniques.

3. In order to develop hyperheuristic research, we need to apply hyperheuristics to other problems and other domains. This will help identify classes of problems for which hyperheuristics seem to work well (e.g. scheduling problems) and classes for which they do not.

Will this emerging hyperheuristic direction change the way in which decision support technology is developed and applied? And will it enable a wider uptake of decision support technology?

Appendix: C++ code for a simple hyperheuristic

To illustrate how simple a hyperheuristic can be, we give the C++ code for *SR* described in chapter 5 (page 112).

```
CHyperheuristic::SimpleRandomAM()
{
    clock_t    goal, clockinit, clockfin;
    goal = timeClockAllowed + clock();
    clockinit = clock();
    int    i = 0, j = 0, NbIteration = 0, LLHeuristicIndex = 0;
    double    EV = 0;
    srand( (unsigned)time( NULL ) );
    while((goal > clock()) && (CurrentSolution−>EvaluationFunction() > 0))
    {
        NbIteration += 1;
        LLHeuristicIndex = RandomInteger(NbAvailableLLHeuristics);
        EV = CurrentSolution−>getEvaluationFunction();
        ApplyHeuristic(LLHeuristicIndex, PreviousLLHeuristicIndex, EV);
        $PreviousLLHeuristicIndex = LLHeuristicIndex$;
        if(CurrentSolution−>getEvaluationFunction() <
                    BestSolution−>getEvaluationFunction())
        {
            (*UpdateBestSolutionPtr) ();
        }
    }
    $clockfin$ = clock();
}
```

Bibliography

[1] K. Aardal and A. Ari. Decomposition principles applied to the dynamic production and workforce scheduling problem. *Engineering Costs and Production Economics*, 12:39–49, 1987.

[2] E. Aarts and J. K. Lenstra, editors. *Local search in combinatorial optimisation.* John Wiley & Sons Ltd, 1997.

[3] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3), 1988.

[4] S. C. Aggarwal, F. P. Wyman, and B. A. McCarl. An investigation of a cost-based rule for job shop scheduling. *International Journal of Production Research*, 11(3):247–261, 1973.

[5] I. Ahmad, M. K. Dhodhi, K. Saleh, and R. H. Storer. High-level synthesis of self-recoverable asics using micro rollback. *International Journal of Electronics*, 75(5):919–932, 1993.

[6] S. Ahmadi, P. Cowling, P. Cheng, and R. Barone. Constructive heuristics and a heuristic space search hyperheuristics for the examination timetabling problem. Technical report, School of Computer Science & IT, The University of Nottingham, 2002.

[7] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighbourhood search techniques. *Discrete Applied Mathematics*,

123:75–102, 2002.

[8] U. Aickelin. *Genetic algorithms for multiple-choice optimisation problems.* PhD thesis, European Business Management School, University of Wales Swansea, September 1999.

[9] U. Aickelin and K. A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3:139–153, 2000.

[10] H. K. Alfares. An efficient two-phase algorithm for cyclic days-off scheduling. *Computers and Operations Research*, 25(11):913–923, 1998.

[11] H. K. Alfares, J. E. Bailey, and W. Y. Lin. Integrated project operations and personnel scheduling with multiple labour classes. *Production Planning and Control*, 10(6):570–578, 1999.

[12] J. Allen and S. Minton. Selecting the right heuristic algorithm: runtime performance predictors. In *Canadian Artificial Intelligence Conference*, 1996.

[13] R. Anbil, E. Gelman, B. Patty, and R. Tanga. Recent advances in crew pairing optimisation at american airlines. *Interfaces*, 21(1):62–74, 1991.

[14] D. W. Ashley. A spreadsheet optimisation system for library staff scheduling. *Computers and Operations Research*, 22(6):615–624, 1995.

[15] T. Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42:591–603, 1996.

[16] T. Aykin. A comparative evaluation of modelling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125:381–397, 2000.

[17] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of evolutionary computation.* Oxford University Press, 1997.

[18] T. Bäck, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth Conference on Genetic Algorithms*, pages 2–9. Morgan Kauffman, 1991.

[19] J. Bailey, H. Alfares, and W. Y. Lin. Optimisation and heuristic models to integrate project task and manpower scheduling. *Computers and Industrial Engineering*, 29(1-4):473–476, 1995.

[20] R. N. Bailey, K. M. Garner, and M. F. Hobbs. Using simulated annealing and genetic algorithms to solve staff scheduling problems. *Asia-Pacific Journal of Operational Research*, 119:27–43, 1997.

[21] K. Baker. Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, 27(1):155–167, 1976.

[22] J. J. III. Bartholdi, J. B. Orlin, and H. D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28:1074–1085, 1980.

[23] N. Beaumont. Scheduling staff using mixed integer programming. *European Journal of Operational Research*, 98:473–484, 1997.

[24] S.E. Bechtold, M.J. Brusco, and M.J. Showalter. A comparative evaluation of labor tour scheduling methods. *Decision Sciences*, 22:683–699, 1991.

[25] S.E. Bechtold and L.W.Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990.

[26] D. D. Bedworth and J. E. Bailey. Personnel scheduling. In D. D. Bedworth and J. E. Bailey, editors, *Integrated production control systems: Management, Analysis and Design, 2nd Edition*, pages 387–420. Wiley - New York, 1987.

[27] O. Berman, R. C. Larson, and E. Pinker. Scheduling workforce and workflow in a high volume factory. *Management Science*, 43:158–172, 1997.

[28] D. P. Bertsekas, editor. *Dynamic Programming and Optimal Control: 2nd*

Edition, volume 1. Athena Scientific, 2000.

[29] D. P. Bertsekas, editor. *Dynamic Programming and Optimal Control: 2nd Edition*, volume 2. Athena Scientific, 2001.

[30] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, editors. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[31] A. Billionnet. Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, 114:105–114, 1999.

[32] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: overview and conceptual comparison. Technical Report TR/IRIDIA/2001-13, Institut de Recherches Interdisciplinaires et de Developpements en Intelligence Artificielle, IRIDIA, Brussels, Belgium, 2001.

[33] L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews - the state of the art. *Computers and Operations Research*, 10(2):63–211, 1983.

[34] D. J. Bradley and J. B. Martin. Continuous personnel scheduling algorithms: a literature review. *Journal Of The Society For Health Systems*, 2(2):8–23, 1990.

[35] H. J. Bremermann. The evolution of intelligence. the nervous system as a model of its environment. Technical Report 1, Contract 477(17), Department of Mathematics, University of Washington, Seattle, 1958.

[36] M. J. Brusco. Solving personnel tour scheduling problems using the dual all-integer cutting plane. *IIE Transactions*, 30:835–844, 1998.

[37] M. J. Brusco and L. W. Jacobs. A simulated annealing approach to the cyclic staff-scheduling problem. *Naval Research Logistics*, 40:69–84, 1993.

[38] M. J. Brusco and L. W. Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organizations. *European Journal of Operational Research*, 86:249–261, 1995.

[39] M. J. Brusco and L. W. Jacobs. Personnel tour scheduling when starting-time restrictions are present. *Management Science*, 44(4):534–547, April 1998.

[40] M. J. Brusco and L. W. Jacobs. Starting-time decisions in labor tour scheduling: An experimental analysis and case study. *European Journal of Operational Research*, 131:459–475, 2001.

[41] M. J. Brusco and T. R. Johns. Improving the dispersion of surplus labor in personnel scheduling solutions. *Computers and Industrial Engineering*, 28(4):745–754, 1995.

[42] M. J. Brusco and T. R. Johns. A sequential integer programming method for discontinuous labor tour scheduling. *European Journal of Operational Research*, 95:537–548, 1996.

[43] M. J. Brusco and T. R. Johns. Staffing a multiskilled workforce with varying levels of productivity: an analysis of cross-training policies. *Decision Sciences*, 29(2):499–515, 1998.

[44] M.J. Brusco and L.W.Jacobs. Optimal models for meal-break and start-time flexibility in continuous tour scheduling. *Management Science*, 46(12):1630–1641, 2000.

[45] M.J. Brusco, L.W.Jacobs, R.J. Bongiorno, D.V. Lyons, and B. Tang. Improving personnel scheduling at airline stations. *Operations Research*, 43(5):741–751, 1995.

[46] E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *Selected Papers of the 2nd Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'98)*, Lecture

Notes in Artificial Intelligence, pages 186–194. Springer, Berlin Heidelberg New York, 1998.

[47] E. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *International Journal of Applied Intelligence*, 15(3):199–214, November/December 2001.

[48] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. In F. Glover and G. A. Kochenberger, editors, *Handbook of metaheuristics*, pages 457–474. Kluwer Academic Publishers, 2003.

[49] E. K. Burke and G. Kendall. Comparison of meta-heuristic algorithms for clustering rectangles. *Computers and Industrial Engineering*, 37(1-2):383–386, 1998. Proceedings of the 24th International Conference on Computers and Industrial Engineering.

[50] E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock cutting problem. *The Journal of Operations Research*, 2003. To Appear.

[51] E. K. Burke, B. MacCarthy, S. Petrovic, and R. Qu. Knowledge discovery in hyper-heuristics using case-based reasoning on course timetabling. In *Full Proceedings of the Fourth International Conference on the Practice And Theory of Automated Timetabling, PATAT 2002*, pages 90–103, August 2002.

[52] E. K. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for examination timetabling. In *4th Asia-Pacific Conference on Simulated Evolution And Learning, SEAL 2002*. Nanyang Technology University, NTU Press, 2002. CD-ROM.

[53] E. K. Burke, J. D. Landa Silva, and E. Soubeiga. A hyperheuristic approach for multi-objective optimisation: an extended abstract. In *Metaheuristic In-*

ternational Conference MIC'2003, August 2003. Submitted.

[54] E.K. Burke and J. P. Newall. Solving examination timetabling problems through adaption of heuristic orderings. *Annals of Operations Research*, 2003. To appear.

[55] R.N. Burns and M.W. Carter. Work force size and single shift schedules with variable demands. *Management Science*, 31(5):599–607, 1985.

[56] X. Cai and K. N. Li. A genetic algorithm for scheduling staff of mixed skills under multi-criteria. *European Journal of Operational Research*, 125:359–369, 2000.

[57] M. J. Cavaretta and R. G. Reynolds. Discovering search heuristics for concept learning using version-space guided genetic algorithms. In *Proceedings of the Seventh Florida Artificial Intelligence Research Symposium*, pages 71–75, Florida, 1994.

[58] I. Charon and O. Hudry. The noising method: a new method for combinatorial optimisation. *Operations Research Letters*, 14:133–137, 1993.

[59] X. Chen and M. L. Bushnell, editors. *Efficient Branch and Bound Search with Application to Computer-Aided Design.* Kluwer Academic Publisher, 1995.

[60] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Brechenridge, CO, April 2000.

[61] S. Chien, A. Stechert, and D. Mutz. Efficient heuristic hypothesis ranking. *Journal of Artificial Intelligence Research*, 10:375–397, 1999.

[62] S. C. K Chu and E. C. H. Chan. Crew scheduling of light rail transit in hong kong: from modeling to implementation. *Computers and Operations Research*, 25(11):887–894, 1998.

[63] B. Codenotti, G. Manzini, L. Margara, and G. Resta. Perturbation: An efficient technique for the solution of very large instances of the euclidean tsp. *INFORMS Journal on Computing*, 8(2):125–133, 1996.

[64] C. A. Coello Coello. A short tutorial on evolutionary multiobjective optimization. In *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001*, Lecture Notes in Computer Science, Vol 1993, pages 21–40. Springer, 2001.

[65] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Publishers, 2002.

[66] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization.* McGraw-Hill Publishing Company, 1999.

[67] D. Corne and J. Ogden. Evolutionary optimisation of methodist preaching timetables. In *Second International Conference on the Practice And Theory of Automated Timetabling, PATAT'97*, volume 1408 of *Lecture Notes in Computer Science*, pages 142–155. Springer-Verlag, 1997.

[68] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Congress on Evolutionary Computation, CEC'02*, pages 1185–1190, 2002.

[69] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach for scheduling a sales summit. In *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, Lecture Notes in Computer Science, pages 176–190, Konstanz, Germany, August 2000. Springer.

[70] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Metaheuristic International Conference*

MIC'2001, pages 127–131, Porto, Portugal, July, 16-20 2001.

[71] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics. *The Journal of Heuristics*, 2002. Submitted.

[72] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In *Parallel Problem Solving from Nature VII, PPSN 2002*, Lecture Notes in Computer Science, pages 851–860, Granada, Spain, September, 7-11 2002. Springer-Verlag.

[73] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Second European Conference on Evolutionary Computing for Combinatorial Optimisation, EvoCop 2002*, Lecture Notes in Computer Science, pages 1–10, Kinsale, Ireland, April, 3-5 2002. Springer.

[74] P. Cowling, D. Ouelhadj, and S. Petrovic. Multi-agent systems for dynamic scheduling. In *PLANSIG Workshop 2000*, Open University, Milton Keynes, England, 2000.

[75] W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, -(117), 1963.

[76] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity analysis of production and allocation*, pages 339–347. Wiley, New York, 1951.

[77] G. B. Dantzig. A comment on edie's traffic delays at toll booths. *Operations Research*, 2:339–341, 1954.

[78] G. B. Dantzig. *Linear programming and extensions.* Princeton University Press, 1963.

[79] L. D. Davis, editor. *Handbook of genetic algorithms.* Van Nostrand Reinhold, 1991.

[80] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun. The space shuttle ground processing scheduling system. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 423–449, San Francisco, 199. Morgan Kaufmann.

[81] M. K. Dhodhi, I. Ahmad, and R. Storer. Shemus: synthesis of heterogeneous multiprocessor system. *Microprocessors and Microsystems*, 19(6):311–319, 1995.

[82] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(8):934–944, 1995.

[83] B. Dodin, A. A. Elimam, and E. Rolland. Tabu search in audit scheduling. *European Journal of Operational Research*, 106:373–392, 1998.

[84] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computaion*, 1(1):53–66, 1997.

[85] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, Politecnico di Milano, Italy, 1991.

[86] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.

[87] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, 22(1):25–40, 1995.

[88] D. Dowling, M. H. M. Krishnamoorthy, and D. Sier. Staff rostering at a large international airport. *Annals of Operations Research*, 72:125–147, 1997.

[89] K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.

[90] K. A. Dowsland and J. M. Thompson. Solving a nurse scheduling problem with knapsacks, networks and tabu search. *Journal of the Operational Research Society*, 51:825–833, 2000.

[91] G. Ducote and E. M. Malstrom. A design of personnel scheduling software for manufacturing. *Computers and Industrial Engineering*, 37:473–476, 1999.

[92] G. Dueck and T. Scheuer. Threshold accepting - a general-purpose optimisation algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[93] M. Duque-Antón. Constructing efficient simulated annealing algorithms. *Discrete Applied Mathematics*, 77:139–159, 1997.

[94] F. F. Easton and N. Mansour. A distributed genetic algorithm for deterministic and stochastic labor scheduling problems. *European Journal of Operational Research*, 118:505–523, 1999.

[95] F. F. Easton and D. F. Rossin. A stochastic goal program for employee scheduling. *Decision Sciences*, 27(3):541–568, 1996.

[96] F. F. Easton and D. F. Rossin. Overtime schedules for full-time service workers. *Omega International Journal of Management Science*, 25(3):285–299, 1997.

[97] S. Eilon and D. J. Cotterill. A modified si rule in job shop scheduling. *International Journal of Production Research*, 7(2):135–145, 1968.

[98] A. Ernst, P. Hourigan, M. Krishnamoorthy, G. H. N. Mills, and D. Sier. Ros-

tering ambulance officers. In *Proceedings of the 15th National Conference of the Australian Society for Operations Research, Gold Coast*, pages 470–481, 1999.

[99] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: a review of applications, methods and models. *European Journal of Operational Research*, 2001.

[100] B. Faaland and T. Schmitt. Cost-based scheduling of workers and equipment in a fabrication and assembly shop. *Operations Research*, 41(2):253–268, 1993.

[101] V. Fabian. Simulated annealing simulated. *Computers Math. Applic.*, 33(1/2):81–94, 1997.

[102] H.L Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pages 375–382, San Mateo, 1993. Morgan Kaufmann.

[103] H.L Fang, P. Ross, and D. Corne. A promising hybrid ga/ heuristic approach for open-shop scheduling problems. In A. Cohn, editor, *Eleventh European Conference on Artificial Intelligence*. John Wiley & Sons, 1994.

[104] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[105] P. Festa and M. G. C. Resende. Grasp: an annoted bibliography. In P. Hansen C. C. Ribeiro, editor, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[106] E. Fink. How to solve it automatically: selection among problem-solving methods. In *Fourth International Conference of AI Planning Systems*, pages 128–136, 1998.

[107] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institue of Technology, May 10-12 1961.

[108] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.

[109] C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.

[110] D. B. Fogel. Asymptotic convergence properties of genetic algorithms and evolutionary programming. analysis and experiments. *Cybernetics and Systems*, 25:389, 1994.

[111] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5:3, 1994.

[112] D. B. Fogel, editor. *Evolutionary computation: The Fossil Record*. IEEE Press, 1998.

[113] G. B. Fogel and D. B. Fogel. Continuous evolutionary programming. analysis and experiments. *Cybernetics and Systems*, 26:79, 1995.

[114] L. J. Fogel, A. J. Owens, and M. J. Walsh, editors. *Artificial intelligence through simulated evolution*. Wiley, New York, 1966.

[115] A. S. Fraser. Simulation of genetic systems by automatic digital computers. ii. effects of linkage on rates under selection. *Australian Journal of Biological Sciences*, 10:492–499, 1957.

[116] A. S. Fraser. Simulation of genetic systems by automatic digital computers. iv. epistasis. *Australian Journal of Biological Sciences*, 13:329–346, 1960.

[117] A. Gaballa and W. Pearce. Telephone sales manpower planning at quantas.

Interfaces, 9(3):1–9, 1979.

[118] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Science*, 8:156–166, 1977.

[119] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[120] F. Glover. Tabu search - part i. *ORSA Journal of Computing*, 1(2):190–206, 1989.

[121] F. Glover. Tabu search - part ii. *ORSA Journal of Computing*, 2(1):4–32, 1990.

[122] F. Glover. Scatter search and star paths: beyond the genetic metaphor. *OR Spektrum*, 17:125–137, 1995.

[123] F. Glover. A template for scatter search and path relinking. In J. K. Hao, E. Lutton, E. M. A. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, Lecture Notes in Computer Science 1363, pages 3–51. Springer, 1998.

[124] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics.* Kluwer Academic Publishers, 2003.

[125] F. Glover and M. Laguna, editors. *Tabu search.* Kluwer Academic Publishers, 1997.

[126] David E. Goldberg. *Genetic algorithms in search, optimization & machine learning.* Addison Wesley, 1989.

[127] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull American Mathematical Society*, 64:275–278, 1958.

[128] R. E. Gomory. Solving linear programming problems in integers. In R. Bellman and M. Hall, editors, *Combinatorial Analysis, Proceedings of the tenth*

Symposium in applied mathematics of the American Mathematical Society, 1960.

[129] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent advances in mathematical programming.* McGraw-Hill. New York, 1963.

[130] R. E. Gomory. All-integer integer programming algorithm. In J.F. Muth and E.L. Thompson, editors, *Industrial Scheduling*, pages 193–206. Prentice Hall, Englewood Cliffs, N. J., 1963.

[131] M. Gopalakrishnan, S. Gopalakrishnan, and D. M. Miller. A decision support system for scheduling personnel in a newspaper publishing environment. *Interfaces*, 23(4):104–115, 1993.

[132] J. Gratch and S. Chien. Adaptive problem-solving for large-scale scheduling problems: a case study. *Journal of Artificial Intelligence Research*, 4:365–396, 1996.

[133] P. Greistorfer. A tabu scatter search metaheuristic for the arc routing problem. *Computers and Industrial Engineering*, 2001. To appear.

[134] J. Gu. Efficient local search with search space smoothing: a case study of the traveling salesman problem (tsp). *IEEE Transactions on Systems, Man and Cybernetics*, 24(5):728–735, 1994.

[135] L. Han, G. Kendall, and P. Cowling. An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *4th Asia-Pacific Conference on Simulated Evolution And Learning, SEAL 2002.* Nanyang Technology University, NTU Press, 2002. CD ROM.

[136] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical methods in Combinatorial Optimization*, Captri, Italy, 1986.

[137] P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[138] E. Hart and P. Ross. A heuristic combination method for solving job-shop scheduling problems. In A. E. Eiben, T. Back, M. Schoenauer, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 845–854. Springer-Verlag, 1998.

[139] E. Hart, P. Ross, and J. A. D. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computing*, 6(1):61–80, 1998.

[140] E. Hart, P. Ross, and J. A. D. Nelson. Scheduling chicken catching- an investigation into the success of a genetic algorithm on a real world scheduling problem. *Annals of Operations Research*, 92:363–380, 1999.

[141] K. Hasse. *Advanced column generation techniques with applications to marketing, retail and logistics management.* PhD thesis, University of Kiel, 1999. Habilitation Thesis.

[142] D. Henderson, S. H. Jacobson, and W. Johnson. The theory and practice of simulated annealing. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 287–319. Kluwer Academic Publishers, 2003.

[143] M. Herdy. Application of the evolution strategy to discrete optimization problems. In *First International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 496 of *Lecture Notes in Computer Science*, pages 188–192. Springer-Verlag, 1991.

[144] J. H. Holland, editor. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[145] J. H. Holland, editor. *Adaptation in natural and artificial systems.* University of Michigan Press, 1992. Second edition, MIT Press 1992.

[146] R. Howick and M. Pidd. Sales force deployment models. *European Journal of Operational Research*, 48(3):295–310, 1990.

[147] J. Hueter and W. Swart. An integrated labor-management system for taco bell. *Interfaces*, 28(1):75–91, 1998.

[148] R. Hung. Scheduling a workforce under annualized hours. *International Journal of Production Research*, 37(11):2419–2427, 1999.

[149] R. Hung and H. Emmons. Multiple-shift workforce scheduling under the 3-4 compressed workweek with a hierarchical workforce. *IIE Transactions*, 25(5):82–89, September 1993.

[150] L.W. Jacobs and M.J. Brusco. Overlapping start-time bands in implicit tour scheduling. *Management Science*, 42(9):1247–1259, 2000.

[151] A.I.Z. Jarrah, J.F. Bard, and A.H. deSilva. Solving large-scale tour scheduling problems. *Management Science*, 40(9):1124–1144, 1994.

[152] B. Jaumard, F. Semet, and T. Vovor. A generalised linear programming model for nurse scheduling. *European Journal of Operational Research*, 107(1):1–18, 1998.

[153] D. E. Joslin and D. P. Clements. Squeaky wheel optimisation. *Journal of Artificial Intelligence*, 10:353–373, 1999.

[154] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[155] E. G. Keith. Operator scheduling. *AIIE Transactions*, 11(1):37–41, 1979.

[156] J. D. Kelly and L. Davis. Hybridizing the genetic algorithm and the k nearest neighbors classification algorithm. In *Fourth International Conference on Genetic Algorithms*, pages 377–383, 1991.

[157] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hy-

perheuristics. In *4th Asia-Pacific Conference on Simulated Evolution And Learning, SEAL 2002*, pages 667–671. Nanyang Technology University, NTU Press, 2002. CD ROM.

[158] G. Kendall, E. Soubeiga, and P. Cowling. The principles of hyperheuristics and their applications in the real world. *The Journal of Scheduling*, 2002. Submitted.

[159] G. Kendall, E. Soubeiga, and P. Cowling. Hyperheuristics: a robust optimisation method for real-world scheduling. *European Journal of Operational Research*, 2003. In preparation.

[160] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systen Technical Journal*, 49:291–308, 1970.

[161] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[162] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[163] C. Koulamas, S. R. Antony, and R. Jaen. A survey of simulated annealing applications to operations research problems. *Omega International journal of Management Science*, 22(1):41–56, 1994.

[164] Koza, Benett, Andre, and Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco, CA, 1999.

[165] Koza, Benett, Andre, Keane, and Brave. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. Morgan Kaufmann, San Francisco, CA, 1999.

[166] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[167] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[168] J. R. Koza. *Genetic Programming II Videotape: The Next Generation.* MIT Press, Cambridge, MA, 1994.

[169] S. Kumar and S. Arora. Efficient workforce scheduling for a serial processing environment: a case study at minneapolis star tribune. *Omega International Journal of Management Science*, 27:115–127, 1999.

[170] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Seventh International Conference on Machine Learning*, pages 511–518, 2000.

[171] M. Laguna and R. Mart'i. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal of Computing*, 11(1):44–52, 1999.

[172] W. B. Langdon. Scheduling planned maintenance of the national grid. In T. C. Fogarty, editor, *Evolutionary Computing*, number 993 in Lecture Notes in Computer Science, pages 132–153. Springer-Verlag, 1995.

[173] W. B. Langdon. Scheduling maintenance of electrical power transmission networks using genetic programming. In J. R. Koza, editor, *Late-breaking papers, Genetic Programming Conference*, 1996.

[174] H. C. Lau. On the complexity of manpower shift scheduling. *Computers and Operations Research*, 23(1):93–102, 1996.

[175] V. J. Leon and R. Balakrishnan. Strength and adaptability of problem-space neighborhoods for resource-constrained scheduling. *OR Spektrum*, 17(2/3), 1995.

[176] D. Lesaint, C. Voudouris, and N. Azarmi. Dynamic workforce scheduling for british telecommunications plc. *Interfaces*, 30(1):45–56, 2000.

[177] T. Liang and B. Buclatin. Improving the utilization of training resources through optimal assignment in the u.s. navy. *European Journal of Operational Research*, 33:183–190, 1988.

[178] C. K. Y. Lin. The development of a workforce management system for a hotline service. *Computers and Industrial Engineering*, 37:465–468, 1999.

[179] S. Lin and B. W. Kernighan. An efficient heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1971.

[180] S. E. Ling. Integrating genetic algorithms with a prolog assignment program as a hybrid solution for a polytechnique timetable problem. In *Parallel Problem Solving from Nature, 2*, pages 321–329, 1992.

[181] J. Loucks and F. Jacobs. Tour scheduling and task assignment of a heterogeneous workforce: a heuristic approach. *Decision Science*, 22(4), 1991.

[182] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.

[183] R. Love and J. Hoey. Management science improves fast food operations. *Interfaces*, 20(2):21–29, 1990.

[184] V. Mabert. A case study of encoder shift scheduling under uncertainty. *Management Science*, 25(7):623–631, 1979.

[185] O. C. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[186] O. C. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[187] O. C. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*,

11:219–224, 1992.

[188] A. J. Mason, D. M. Ryan, and D. M. Panton. Integrated simulation, heuristic and optimisation approaches to staff scheduling. *Operations Research*, 46/1:161–175, 1998.

[189] N. Metropolis, A. W. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[190] K. Miettinen. Some methods for nonlinear multi-objective optimization. In *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001*, Lecture Notes in Computer Science, Vol 1993, pages 1–20. Springer, 2001.

[191] H. Millar and M. Kiragu. Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European Journal of Operational Research*, 104(3):582–592, 1998.

[192] R. G. J. Mills and D. M. Panton. Scheduling of casino security officers. *OMEGA International Journal of Management Science*, 20(2):183–191, 1992.

[193] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161–205, 1988.

[194] N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[195] A. Mockus, J. Mockus, and L. Mockus. Adapting stochastic and heuristic methods for discrete optimization problems. *Informatica*, 5(1):123–166, 1994.

[196] A. Mockus, J. Mockus, and L. Mockus. Bayesian approach adapting stochastic and heuristic methods for discrete optimization. In *Abstracts, Second world*

Meeting, pages 10–11, Alicante, Spain, 19-22 August 1995. International Society for Bayesian Analysis.

[197] A. Mockus, J. Mockus, and L. Mockus. Discrete optimization, information based complexity and bayesian heuristics approach. In *International Sysmposium on Operations Research with Applications in Engineering Technology and Management, ISORA'95*, Beijing, China, 19-22 August 1995.

[198] J. Mockus. *Bayesian approach to global optimization.* Kluwer Academic Publishers, Dordrecht-Boston-London, 1989.

[199] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–366, 1994.

[200] J. Mockus. A set of examples of global and discrete optimization, part i. *Informatica*, 8:237–264, 1997.

[201] J. Mockus. *A set of examples of global and discrete optimization: application of Bayesian heuristic approach.* Kluwer Academic Publishers, Dordrecht-Boston-London, 2000.

[202] J. Mockus, W. Eddy, A. Mockus, L. Mockus, and G. Reklaitis. *Bayesian heuristic approach to discrete and global optimization.* Kluwer Academic Publishers, 1997.

[203] J. Mockus and L. Mockus. Bayesian approach to global optimization and applications to multi-objective constrained problems. *Journal of Optimization Theory and Applications*, 70(1):155–171, July 1991.

[204] S.L. Moondra. An l.p. model for work force scheduling for banks. *J. Bank Res.*, 7(4):299–301, 1976.

[205] P. Moscato. On evolution, search, optimisation, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 826, California Institute

of Technology, Caltech Concurrent Computation Program, 1989.

[206] P. Moscato and J. F. Fontanari. Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability*, 18:747–771, 1990.

[207] G. Mould. Case study of manpower planning for clerical operations. *Journal of the Operational Research Society*, 47(3):358–368, 1996.

[208] R. Narasimhan. An algorithm for single shift scheduling of hierarchical workforce. *European Journal of Operational Research*, 96:113–121, 1996.

[209] R. Narasimhan. An algorithm for multiple shift scheduling of hierarchical workforce on four-day or three-day workweeks. *INFOR*, 38(1):14–32, February 2000.

[210] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristic International Conference MIC'2001*, Porto, Portugal, July 16-20 2001. Kluwer. Submitted.

[211] A. Nareyek. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In *Metaheuristic International Conference MIC'2001*, pages 211–215, Porto, Portugal, July 16-20 2001.

[212] I. Norenkov and E. Goodman. Solving scheduling problems via evolutionary methods for rule sequence optimization. 2nd World Conference on soft Computing, WSC2, June 1997. Online paper downloadable at http://garage.cps.msu.edu/projects/scheduling.html (paper GARAGe97-05-01).

[213] I. P. Norenkov. Scheduling and allocation for simulation and synthesis of cad system hardware. In *Proceedings EWITD 94, East-West International Conference, ICSTI*, pages 20–24, Moscow, 1994.

[214] P. J. O'Grady and C. Harrison. A general search sequencing rule for job shop sequencing. *International Journal of Production Research*, 5:961–973, 1985.

[215] I. H. Osman and J. P. Kelly, editors. *Meta-heuristics: Theory & Applications.* Best papers of the 1995 Metaheuristic International Conference. Kluwer Academic Publishers, 1996.

[216] I. H. Osman and G. Laporte. Metaheuristics: a bibliography. *Annals of Operations Research*, pages 513–623, 1996.

[217] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[218] P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimisation.* Oxford University Press, 2002.

[219] G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–280, 1999.

[220] S. Petrovic and R. Qu. Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems. In *Sixth International Conference on Knowledge-based Intelligent Information and Engineering Systems, KES 2002*, September 2002. To appear.

[221] M. Pirlot. General local search methods. *European Journal of Operational Research*, 92:493–511, 1996.

[222] L. S. Pitsoulis and M. G. C. Greedy randomized adaptive search procedures. Technical report, AT&T Labs Research, 2001.

[223] M. Poliac, E. Lee, J. Slagle, and M. Wick. A crew scheduling problem. In *IEEE 1st International Conference on Neural Networks*, pages 779–786, 1987.

[224] G. Rabideau, S. Chien, J. Willis, and T. Mann. Using iterative repair to automate planning and scheduling of shuttle payload operations. In *Innovative Applications of Artificial Intelligence, IAAI 99*, July 1999.

[225] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative

repair planning for spacecraft operations using the aspen system. In *International Symposium on Artificial Intelligence Robotics and Automation in Space, iSAIRAS 99*, 1999.

[226] D. Rafaeli, D. Mahalel, and J. N. Prashker. Heuristic approach to task scheduling: ‘weight‘ and ‘improve‘ algorithms. *International Journal of Production Economics*, 29:175–186, 1993.

[227] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern heuristic search methods*. Wiley, Chichester, 1996.

[228] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[229] C. R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. Blackwell, Oxford, 1993.

[230] C. R. Reeves and J. E. Rowe, editors. *Genetic Algorithms: Principles and perspectives*. Kluwer, Norwell, MA, 2001.

[231] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.

[232] C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMS Journal of Computing*, 2002.

[233] J. Rosenhead, editor. *Rational Analysis for a Problematic World - Problem Structuring Methods for Complexity, Uncertainty and Conflict*. Wiley, London, 1989.

[234] P. Ross, S. Schulenburg, J. G. Marin-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In *Proceedings of the Genetic and Evolutionary Computation COnference, GECCO'02*. Morgan-Kauffman, 2002.

[235] S. Sachdev. *An exploration of A-teams.* PhD thesis, Carnegie Mellon University, 1998.

[236] A. Schaerf. Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man and Cybernetics Part A:systems and Human*, 29/4:368–377, 1999.

[237] J. D. Schaffer and L. J. Eshelman. Combinatorial optimisation by genetic algorithms: The value of the genotype/phenotype distinction. In *First International Conference on Evolutionary Computation and its Applications EvCA'96*, pages 110–120, Moscow, RUSSIA, June 24-27 1996. Presidium of the Russian Academy of Sciences, Springer-Verlag.

[238] J. D. Schaffer and L. J. Eshelman. Combinatorial optimisation by genetic algorithms: the value of the genotype/phenotype distinction. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search*, pages 85–97. John Wiley and Sons, 1996.

[239] S. Schindler and T. Semmel. Station staffing at pan american world airways. *Interfaces*, 23(3):91–98, 1993.

[240] J. A. M. Schreuder. Assigning magistrates to sessions of the amsterdam criminal court. In E. Burke and W. Erben, editors, *Proceedings of the third international conference on the Practice and Theory of Automated Timetabling*, page 333, 2000.

[241] H. P. Schwefel. *Evolutionsstrategie und numerische optimierung.* PhD thesis, Technische Universität Berlin, 1975.

[242] J. F. Shapiro. *Mathematical programming: structures and algorithms.* John Wiley & Sons, 1979.

[243] M. T. Shing and G. B. Parker. Genetic algorithms for the development of real-time multi-heuristic search. In S. Forrest, editor, *Fifth International Confer-*

ence on Genetic Algorithms, pages 575–572, San Mateo, 1993. Morgan Kaufmann.

[244] D. Smith. Bin packing with adaptive search. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 202–207. Lawrence Erlbaum Associates, Publishers, 1985.

[245] J. Smith. Co-evolving memetic algorithms: inital investigations. In *Parallel Problem Solving from Nature VII, PPSN 2002*, Lecture Notes in Computer Science, pages 537–546, Granada, Spain, September, 7-11 2002. Springer-Verlag.

[246] S. F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, pages 422–425, Karlsruhe, West Germany, August 1983.

[247] S. F. Smith. Opis: A methodology and architecture for reactive scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 423–449, San Francisco, 1994. Morgan Kaufmann.

[248] M. V. Solodov and B. F. Svaiter. Descent methods with line search in the presence of perturbations. *Journal of Computational and Applied Mathematics*, 80:265–275, 1997.

[249] R. H. Storer, S. W. Flanders, and S. D. Wu. Problem space local search for number partitioning. *Annals of Operations Research*, 63:465–487, 1996.

[250] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[251] R. H. Storer, S. D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal of Computing*, 7(4):453–467, 1995.

[252] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Improvements, and New Applications.* PhD thesis, Darmstadt University of Technology, Department of Computer Science, 1998.

[253] T. Stützle and H. H. Hoos. The max-min ant system and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 19997 IEEE International Conference on Evolutionary Computaion (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, 1997.

[254] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998. A Bradford Book.

[255] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 333–349, New York, 1991. Van Nostrand Reinhold.

[256] S. Talukdar, S. Murthy, and R. Akkiraju. Asynchronous teams. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 537–556. Kluwer Academic Publishers, 2003.

[257] P. E. Taylor and S. J. Huxley. A break from tradition for the san francisco police: patrol officer scheduling using an optimization-based decision support system. *Interfaces*, 19(1):4–24, 1989.

[258] V. Tcheprasov, E. Goodman, W. Punch, G. Ragatz, and I. Norenkov. A genetic algorithm to generate a pro-active scheduler for a printed circuit board assembly. In *First International Conference on Evolutionary Computation and its Applications EvCA'96*, pages 232–244, Moscow, RUSSIA, June 24-27 1996. Presidium of the Russian Academy of Sciences, Springer-Verlag.

[259] H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In *Genetic and Evolutionary Computation COnference, GECCO'99*, pages 635–642, 1999.

[260] G. M. Thompson. Labor scheduling using npv estimates of the marginal benefit of additional labor capacity. *Journal of Operations Management*, 13:67–86, 1995.

[261] G. M. Thompson. A simulated-annealing heuristic for shift scheduling using non-continuously available employees. *Computers and Operations Research*, 23(3):275–288, 1996.

[262] G. M. Thompson. Assigning telephone operators to shifts at new brunswick telephone company. *Interfaces*, 27(4):1–11, 1997.

[263] G.M. Thompson. Improved implicit optimal modeling of the labor shift scheduling problem. *Management Science*, 41(4):595–607, 1995.

[264] J. M. Thompson and K. A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25(7-8):637–648, July 1998.

[265] J. M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, July 1982.

[266] E. Tsang and C. Voudouris. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters*, 20:119–127, 1997.

[267] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. A local search template. *Computers and Operations Research*, 25(11):969–979, 1998.

[268] Y. van den Berg and D. Panton. Personnel shift assignment: existence conditions and network models. *Networks*, 24:385–394, 1994.

[269] V. Vidal and L. Sorensen. Soft methods in primary schools: Focusing on it strategies. *International Transactions in OR*, 2001.

[270] R. V. Vohra. The cost of consecutivity in the (5, 7) cyclic staffing problem.

IIE Transactions, 29:942–950, 1987.

[271] S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-heuristics: advances and trends in local search paradigms for optimisation.* Best papers of the Metaheuristic International Conference 1997. Kluwer Academic Publishers, 1999.

[272] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, 1997.

[273] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, 1999.

[274] S. C. Wheelwright and S. Makridakis. *Forecasting methods for management.* John Wiley & Sons Inc, 1973.

[275] S. W. Wilson. Zcs: a zeroth order classifier system. *Evolutionary Computation*, 2:1–18, 1994.

[276] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[277] D. Wolpert and W. G. MacReady. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[278] G. Zoutendijk. *Mathematical programming methods.* North-Holland Publishing Company, 1976.

[279] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 423–449, San Francisco, 1994. Morgan Kaufmann.