METAHEURISTIC AND MULTIOBJECTIVE APPROACHES FOR SPACE ALLOCATION

by Jesus Dario Landa Silva, BEng, MSc

Thesis submitted to the University of Nottingham

for the degree of Doctor in Philosophy

School of Computer Science and Information Technology

November 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This thesis presents an investigation on the application of metaheuristic techniques to tackle the space allocation problem in academic institutions. This is a combinatorial optimisation problem which refers to the distribution of the available room space among a set of entities (staff, research students, computer rooms, etc.) in such a way that the space is utilised as efficiently as possible and the additional constraints are satisfied as much as possible. The literature on the application of optimisation techniques to approach the problem mentioned above is scarce. This thesis provides a description and formulation of the problem. It also proposes and compares a range of heuristics for the initialisation of solutions and for neighbourhood exploration. Four well-known metaheuristics (iterative improvement, simulated annealing, tabu search and genetic algorithms) are adapted and tuned for their application to the problem investigated here. The performance of these techniques is assessed and benchmark results are obtained. Also, hybrid approaches are designed that produce sets of high quality and diverse solutions in much shorter time than those required by space administrators who construct solutions manually. The hybrid approaches are also adapted to tackle the space allocation problem from a two-objective perspective. It is also revealed that the use of aggregating functions or relaxed dominance to evaluate solutions in Pareto optimisation, can be more beneficial than the standard dominance relation to enhance the performance of some multiobjective optimisers in some problem domains. A range of single-solution metaheuristics are extended to create hybrid evolutionary approaches based on the scheme of *cooperative local search*. This scheme promotes the cooperation of a population of local searchers by means of mechanisms to share the information gained during the search. This thesis also reports the best results known so far for a set of test instances of the space allocation problem in academic institutions.

This thesis pioneers the application of metaheuristics to solve the space allocation problem. The major contributions are: provides a formulation of the problem together with tests data sets, reports the best known results for these test instances, investigates the multiobjective nature of the problem and proposes a new form of hybridising metaheuristics.

ACKNOWLEDGEMENTS

To initiate a venture is relatively easy, it is enough to invigorate the fire of enthusiasm, to persevere on the venture until success is a different thing, that requires continuity and effort.

There is a big difference between being educated and being wise…education corresponds to science, wisdom corresponds to the conscience.

Thanks god because you gave me the strenght to make the decision to initiate this venture and the strength to complete it. Along the way, there were many times in which your love and company were essential to continue. Thanks for holding me in difficult times.

Thanks to my parents Sebastian and Teresa, from whom I have received so much love and guidance. Thanks to my brother Ulises and my sister Vianney, because our bonds are stronger in the distance and your support has always been there. Thanks to Nilo and Rosita, you are an exceptional example to follow and your love and advice are priceless. Thanks to all my relatives and friends because a part of what I am is because of you.

Love and friendship are essential to overcome difficult times and maintain hope in the future. Therefore, many thanks to all of you my friends, who have shared with me, so many moments of happiness, friendship and love. Among them, Alma Olvera, Iciar Olvera, Flor Torres, Rosy Loya, Tere Cruz, Lillian Tapia, Pedro Maria, German Blanco, Ralf Keuthen, Marina Aguilar, Emmanuela Cerfeda, Kirstin Elsner, Emma Dawson, Eric Soubeiga, Rafael Pulido, Koon Wah Kok and many others that will always be in my memory. All this time would not had been so enjoyable without your company. Special thanks to Majito Beltrán, you have been light in my life and the time spent with you is always delightful.

Thanks to the University of Nottingham for providing me with all the required resources to successfully carry out this research programme. Thanks to everyone in the School of Computer Science and Information Technology and particularly, to all the present and past members of the ASAP research group. Working in such a friendly and harmonious environment has contributed to make this a gratifying experience.

I am extremely grateful to Professor Edmund K. Burke, who has been an exceptional supervisor. His guidance and support along my PhD have helped me to achieve this important goal in my professional career. Also, thanks to Professor Peter Cowling whom gave me many valuable advises in the early stages of my PhD. My gratitute goes as well to Dr. Graham Kendall (internal examiner) from the University of Nottingham and Professor Peter Fleming (external examiner) from the University of Sheffield. Their valuable comments and suggestions during my PhD examination have been included in the final version of this thesis.

This PhD programme would have not been possible without the financial support of PROMEP ("Programa de Mejoramiento al Profesorado") and the UACH ("Universidad Autónoma de Chihuahua") in México. Thanks to both institutions for their sponsorship. I would like to thank the assistance received from all staff in PROMEP and UACH during my PhD. In particular, many thanks to Ing. José Luis Franco Rodriguez and to Dr. José Enrique Grajeda Herrera, present and previous vice-chancellors of UACH. Also, thanks to members of the administration at the "Facultad de Ingeniería" in UACH, Ing. Jesús Valles, Ing. Isela Aguirre, Ing. Jesús Mendoza and M.C. Martha Canales.

I am also very grateful to the various institutions and companies that provided us with data sets for the research carried out in this thesis. Thanks to the University of Nottingham, the University of Wolverhampton, the Nottingham Trent University and Real Time solutions Ltd for their assistance in this aspect.

And finally, as in my master thesis, many thanks to all those that continuously asked me **...how is the thesis going?...** because without knowing it, you encouraged me to persevere.

Chapter 1. Introduction

1.1. Background and Motivation

Office space allocation and the associated resource efficiency issues impact (to a greater or lesser extent) on all institutions from small companies to large multi-national organisations. In academic institutions, the distribution of the available room space among staff, research students and other resources such as lecture rooms, labs, storage rooms, etc., is a process that needs to be carried out on a regular basis because of the continuous changes that occur in this environment. For example, people leave the institution or move to another department/faculty, new lecture rooms or labs are required, offices for new staff or research students should be available, certain rooms are unavailable for various reasons, etc.

Since the available room space is usually restricted, an efficient functioning of the academic institution depends on, among other factors, having a good distribution of this space. A good distribution must ensure that all demanding resources are given the minimum required space, that the space is utilised as efficiently as possible and that the additional constraints are satisfied to as great an extent as possible. An efficient utilisation of the space requires that no resource is given too much room (space wastage) and no resource is given less room than the minimum required (space overuse). Additional constraints usually require that the allocation of resources to the available rooms meets specific conditions. For example, professors must not share offices, research students should be allocated near to their supervisor's office, lecture rooms must be located away from noisy areas, research groups should be located together, etc.

Besides achieving an efficient utilisation of the room space and the satisfaction of additional constraints, producing an adequate allocation requires taking into account other quality factors that are very difficult to evaluate. Space administrators need to consider the preferences of people when assigning offices so that they are satisfied with their working environment. They should also address aspects such as politics and future requirements when distributing the room space. That is, several

criteria (usually from various decision-makers) are employed to evaluate the quality of the space distribution.

Space allocation is a difficult task and a recent survey on this issue revealed that in most of the cases this process is carried out manually and it can take weeks or even months to be completed in this way (Burke and Varley, 1998). That survey showed that only a small proportion of higher education institutions in the UK use some form of computer aid when dealing with the space allocation problem. Usually, this aid consists of databases that maintain a record and drawings of all rooms and how they are being used, but no form of automated space allocation is implemented. Automating the space allocation process would permit space administrators to save time and effort. Moreover, if several solutions are obtained in a short computation time, this would allow the administrator to spend more time in the decision-making process to select the most appropriate allocation considering all the quality factors mentioned above. The application of heuristics to tackle this problem was suggested in (Burke and Varley, 1998b) as a first step towards the construction of a computer system to automate the space allocation process in academic institutions.

Space allocation is a combinatorial optimisation problem that has some similarities with classical knapsack problems (Martello and Toth, 1990) and is also related to scheduling problems such as academic timetabling (Wren, 1996). In the traditional knapsack problem, a set of objects of given sizes must be accommodated into a set of containers of given capacity so that the available capacity is utilised as efficiently as possible, but usually no additional constraints exist. In academic timetabling the problem is to accommodate a set of events into the set of available timeslots so that additional constraints are satisfied. In some cases, the construction of academic timetables also takes into account the allocation of rooms to events (Burke et al., 1996) which is obviously closely related to the space allocation problem.

The range of techniques that have been applied to tackle combinatorial optimisation problems can be classified in two general groups: exact methods and approximate (heuristic) methods (Papadimitriou and Steiglitz, 1999). Exact methods seek to solve a problem to guaranteed optimality but their execution on large real

world problems usually requires too much computation time. For practical use heuristic methods seek to find high quality solutions (not necessarily optimal) within reasonable computation times (Poole et al., 1998). Metaheuristics are a class of heuristic techniques that have been successfully applied to solve a wide range of combinatorial optimisation problems over the years (Glover and Kochenberger, 2003; Voss et al., 1999; Aarts and Lenstra, 1997; Osman and Kelly, 1996; Osman and Laporte, 1996; Rayward-Smith et al., 1996; Reeves, 1995).

This thesis describes an investigation into the development of metaheuristic approaches to automate the space allocation process in academic institutions. This work has been motivated by an interest in developing modern automated algorithms that tackle this problem in a more effective way than currently exists. In addition, given the relation of space allocation to other combinatorial problems such as knapsack and timetabling problems, this investigation may also benefit the development of optimisation techniques that can be applied to other such problems.

1.2. Aims and Scope

Since space allocation is a multiple criteria optimisation and decision process, where some of the criteria are not easily measurable (e.g. preference of people over certain rooms), it is very difficult to obtain an accurate model of this real-world problem. Even if the preferences are expressed in an objective function and optimal or near-optimal solutions are found, it is very likely that the decision-makers will modify these solutions before the final distribution of space is decided. These are some of the arguments in favour for the application of heuristic methods to obtain near-optimal solutions to the space allocation problem.

As expressed above, the space allocation process is very complex and the present thesis tackles one part of this process, the construction of allocations. That is, given a set of entities, to allocate them into the set of available rooms. Two main objectives are pursued when constructing an allocation: minimising the amount of space misuse (wastage and overuse) and minimising the number of constraint violations. Initially, this investigation considers finding one high-quality solution. Then, we address the

situation in which a set of high-quality allocations is required, so that the space administrators can select the most adequate.

The main aim of this thesis is to present an investigation on the application of metaheuristic approaches to solve the space allocation problem in academic institutions. To the best knowledge of the author, apart of (Burke and Varley, 1998b), no other work in this area has been published in the literature. Some reports are available on the application of some exact optimisation techniques to tackle the problem of distributing space in academic institutions (Ritzman et al., 1980; Benjamin et al., 1992; Giannikos et al., 1995). An additional aim here is to present a description and formulation of this problem that helps to better understand it for future research on this subject.

This thesis demonstrates the suitability of applying metaheuristic techniques for automating the space allocation process. Furthermore, several hybrid approaches have been designed as a result of this research and they are described and tested in this document. This thesis also describes a set of test instances of the space allocation problem and reports the best known results.

1.3. Overview of this Thesis

The remainder of this thesis is organised as follows. In the second chapter, a description and formulation of the space allocation process and the specific problem investigated here (the construction of allocations) is presented together with an insight into its relationship with other combinatorial optimisation problems.

Chapter three reviews the literature from two perspectives: the problem and the solution techniques. That is, a review of the published research on the subject of space allocation is presented together with an account and brief description of a range of metaheuristic approaches proposed in the literature. Chapter three also gives an introduction to the theory of algorithms complexity and the No Free Lunch theorem (NFL) of Wolpert and Macready (Wolpert and Macready, 1995; Wolpert and Macready, 1997).

An investigation into the application of a range of metaheuristics to the space allocation problem is presented in chapter four. This initial study aims to identify the strengths and weaknesses of various well-known techniques when used to solve this problem. Four approaches are investigated: iterative improvement, simulated annealing, tabu search and genetic algorithms. Constructive heuristics for initialising solutions and neighbourhood exploration heuristics are also designed, presented and tested in chapter four. Various recombination and mutation operators are also designed and evaluated for this problem.

In chapter five, hybrid metaheuristics for the space allocation problem are developed and tested. First, a single-solution hybrid approach is designed by combining some of the features of the algorithms studied in chapter four. Then, this algorithm is modified to produce two population-based variants in which a common annealing schedule is used to control the evolution of the whole population.

In chapter six, an investigation of the space allocation problem as a two-objective optimisation problem is carried out. That is, instead of using an aggregating function to assign fitness to solutions, the concepts of Pareto optimisation are used in order to produce a set of compromise solutions (Steuer, 1986). First, the multiobjective nature of the space allocation problem is investigated. Then, the suitability of the hybrid algorithms of chapter five to produce a set of compromise solutions is assessed. Finally, it is shown that the fitness evaluation method used to discriminate against solutions during the search, has an impact on the performance of some multiobjective optimisers. As a consequence, we suggest the use of relaxed dominance relations as alternative methods to assign fitness to solutions in multiobjective optimisation.

A scheme for extending single-solution local search algorithms towards hybrid evolutionary approaches is proposed in chapter seven. This scheme is based on the concept of *cooperative local search* which promotes the idea that an evolving population of local searchers share the information gained during the search. In this way, explorative capabilities from population-based methods can be combined with the intensification features of local search techniques without the need to design specialised recombination operators or repairing heuristics to maintain the feasibility

of solutions. This approach appears to hold significant promise for other problems particularly where recombination and repair present serious difficulties. Finally, conclusions and some directions for future work on this area are given in chapter eight.

1.4. Contributions of this Thesis

The contributions of this thesis are summarised as follows:

- A description and formulation of the space allocation problem in British universities is presented. From real data provided by some universities, six data sets have been prepared in a proposed format and these test instances have been made publicly available.

- For the first time, an investigation on the suitablilty of applying metaheuristics to solve the space allocation problem is presented. It is shown that these approaches can produce solutions of better quality than those generated manually by space officers and in a much shorter time.

- Two hybrid algorithms are presented, one point-based and one population-based, which produce the best known solutions for the test instances used in this thesis.

- For the first time, an investigation on the multiobjective nature of the space allocation problem is provided. A form of relaxed dominance is proposed and it is shown that using this form of evaluating solutions is beneficial in the multiobjective optimisation of this problem.

- A new form of hybridisation is proposed in which single-solution local search methods are extended to population-based variants. The result is a cooperative scheme in which a population of local searchers help each other to find better solutions.

Chapter 2. The Space Allocation Problem

2.1. Introduction

In combinatorial optimisation problems the aim is to find an optimal setting of a finite or countable infinite number of discrete entities (Papadimitriou and Steiglitz, 1999). The desired setting can be an arrangement, ordering, grouping, selection or distribution of the entities such that a number of requirements and perhaps constraints are satisfied. The complexity of many combinatorial problems is described by exponential functions and they are considered to be intractable or NP-complete (Garey and Johnson, 1979). Since there are no known polynomial bounded exact algorithms for solving this class of problems, heuristic algorithms are frequently applied with the aim of producing high-quality solutions in a reasonable amount of time (Baase, 1998). Chapter three presents a more detailed discussion of the theory of algorithms complexity including the P and NP classes. Among the class of important and difficult to solve combinatorial problems there are the *capacity allocation problems*. This refers to those problems in which the available capacity or amount of resources has to be distributed among a set of demanding entities. Examples of this type of problems are: the bin-packing problem, the knapsack problem and the generalised assignment problem (Martello and Toth, 1990; Kallarath and Wilson, 1997 chapter 7).

The particular *capacity allocation problem* that motivated the research for this thesis is the distribution of the available office space among staff, research students and other resources in academic institutions. When solving this problem, the goal is to find an allocation that optimises the space utilisation and satisfies (as far as possible) the additional requirements and constraints that may exist. To the best knowledge of the author, there are few publications in the literature reporting research on this problem. For example, (Giannikos et al., 1995) applied goal programming to automate the distribution of offices among staff in an academic institution. The management of space in academic institutions has also been subject of study from a different perspective: planning the layout of offices (Benjamin et al., 1992; Ritzman et al., 1980). The application of some heuristic algorithms to tackle the space allocation problem was explored in (Burke and Varley, 1998b).

In principle, the problem of distributing office space in academic institutions is very similar to two other capacity allocation problems: the multiple knapsack problem and the generalised assignment problem. These two capacity allocation problems are briefly described below in order to provide a background for a better understanding of the space allocation problem in academic institutions. Then, a detailed description and formulation of the space allocation problem is presented. Finally, the test data sets used in the experiments of this thesis are also described. The material presented in this chapter is included in the papers [Bur2000] and [Bur2003b] (see the appendix on page 199).

2.2. Related Problems

2.2.1. Multiple Knapsack Problem

In the multiple knapsack problem there are a number of items of given sizes and a number of knapsacks of given capacities. Each item has an associated profit and an associated weight. The goal is to fill each of the knapsacks with a subset of the items without exceeding the capacity of the knapsack and maximising the total profit. If an item is selected it can only be assigned to one knapsack. This problem is formulated as follows (Martello and Toth, 1990):

m = number of knapsacks
n = number of items
$c(i)$ = capacity of the knapsack i
$p(j)$ = profit associated to item j
$w(j)$ = weight associated to item j
$x(i,j)$ = 1 if item j is selected for knapsack i, 0 otherwise

$$\text{maximize} \qquad f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} p(j)x(i,j) \tag{2.1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} w(j)x(i,j) \le c(i) \quad i = 1,2,...,m \tag{2.2}$$

$$\sum_{i=1}^{m} x(i,j) \le 1 \qquad j = 1,2,\ldots,n \tag{2.3}$$

Because of the binary variable *x(i,j)*, this problem is also known as the 0-1 multiple knapsack problem (Hanafi et al., 1996).

2.2.2. Generalised Assignment Problem

Another type of capacity allocation problem is the generalised assignment problem, which is very similar to the multiple knapsack problem described above. However, in the generalised assignment problem, the profit and weight associated with each of the items vary according to the knapsack for which it is selected. It is common that this problem be described in terms of assigning tasks to agents, assigning jobs to machines or any similar situation. Each agent has a given capacity and each task has a profit and a weight (capacity request) associated to each of the agents. The goal is to distribute all the tasks among the agents ensuring that the sum of weights of all the jobs assigned to each agent does not exceed the agent's capacity and the total profit is maximised. A formulation of the generalised assignment problem can be represented as follows (Martello and Toth, 1990):

m = number of agents
n = number of tasks
$c(i)$ = capacity of the agent i
$p(i,j)$ = profit associated to task j when assigned to agent i
$w(i,j)$ = weight associated to task j when assigned to agent i
$x(i,j)$ = 1 if task j is assigned to agent i, 0 otherwise

$$\text{maximize} \qquad f(x) = \sum_{i=1}^{m}\sum_{j=1}^{n} p(i,j)x(i,j) \qquad (2.4)$$

$$\text{subject to} \qquad \sum_{j=1}^{n} w(i,j)x(i,j) \le c(i) \qquad i = 1,2,...,m \qquad (2.5)$$

$$\sum_{i=1}^{m} x(i,j) = 1 \qquad j = 1,2,\ldots,n \qquad (2.6)$$

Note that, in this formulation, all the tasks have to be assigned to exactly one agent (constraint 2.6). However, in some variations of this problem, it may be permitted that some of the tasks are not assigned to any agent. In this case, equation 2.6 is replaced by equation 2.3 as in the multiple knapsack problem.

2.3. Space Allocation in Academic Institutions

In academic institutions, the distribution of the available room space among staff, research students, laboratories, teaching rooms, etc. is a difficult task because space is a demanded commodity and a variety of conflicting interests are present. Therefore, it is often crucial that the available room space be utilised as efficiently as possible. The available room space in buildings has to be distributed among a set of demanding entities. Each room is assigned with a functionality. For example, some offices are assigned to staff, research rooms for postgraduate students, laboratories, meeting rooms, lecture rooms, seminar rooms, common rooms, etc. In this thesis, the functionality assigned to each room is called an *entity* and each entity requires a certain amount of room space. The amount of room space demanded by each entity is measured (not surprisingly) by the floor area. For example, staff offices may require 12 m^2, computer rooms may need 3 m^2 per workstation, etc. In this problem, it is often the case that it is not possible to assign exactly the required space room to each demanding entity, i.e. space in rooms is often wasted or overused. In this problem there are also additional constraints that restrict the location of certain entities with respect to some rooms or with respect to other entities. For example, a laboratory might need to be allocated next to a lecture room, a professor should not be allocated in a shared room or postgraduate students and staff in a given research group should be allocated in nearby rooms.

Then, the space allocation problem can be seen as the distribution of the available room space among the demanding entities in such a way that the space utilisation is optimised and the additional constraints are satisfied. Constraints can be any of the two following types: *soft constraints* are rules that can be broken but penalised, while *hard constraints* cannot be violated at all.

In (Burke and Varley, 1998) a description of this problem was provided as a result of a questionnaire on the space allocation process that was sent to space administrators in ninety-six British Universities. Thirty-eight of the ninety-six universities replied and the paper describes and analyses the results of the questionnaire. In that paper, the authors stated that (in most of the surveyed universities) this process is carried out by a manual process and only a few British

universities use some kind of automated tool. They also showed that this problem as it actually appears in a wide range of British universities is very complex, highly constrained, contains multiple objectives, varies greatly among different institutions, requires frequent modifications due to the addition or removal of entities and/or rooms and has a direct impact on the functionality of the university.

2.3.1. Space Allocation in UK Universities

This section gives a brief description of the space allocation process in British universities. The paper by (Burke and Varley, 1998) gives more details about this process. In their work, Burke and Varley expressed that, allocating rooms to entities in UK universities is a multi-stage process that can be performed in three phases:

- The estates department or central committee allocates space to faculties and assigns common areas.
- Faculties assign areas to schools and departments.
- Departments allocate specific rooms to staff, research groups, research students and other entities.

However, in practice there is a lower phase when assigning rooms to entities. This is when the head of a research group distributes the office space among the members of the group. During any of these phases, the problem can be solved in different ways:

- Fitting all entities into a limited amount of room space. For example, when all the research student members of the same research group have to be allocated into a number of available rooms.
- Minimising the amount of room space required to allocate a set of entities. For example, when a department has to allocate all the needed teaching rooms in the most efficient way possible.
- Reorganising the existing allocation due to the variation of requirements and/or constraints. For example, a lecturer that is promoted to professor will require a bigger office and the students that he supervises may also need to be relocated.

- Reorganising the existing allocation because of the addition/removal of entities. For example, new staff and additional teaching rooms have to be allocated.

- Reorganising the existing allocation because of a change in available room space. For example, if new rooms are constructed, rooms are resized or rooms are assigned to a different authority (department/school/faculty).

The need for reorganising the distribution of room space is a situation that academic institutions face more frequently that many large organisations due to the dynamic nature of the space distribution in universities (e.g. PhD students and post-doctoral research assistants usually only require space for a three year period). In this case, the economic cost and disturbance caused due to the changes made are very important additional objectives that should be minimised. This often impedes our ability to find very high quality utilisation of the space due to the fact that it is far too costly to completely move everyone around every year or so. The quality of the initial allocation usually has an impact on how much reorganisation is required at a later date when the conditions of the allocation change. Continual reorganisations on a small scale usually result in a bad overall utilisation of space. However, large reorganisations are time consuming and costly. The amount of disruption that should be allowed must be controlled to balance the quality of the new allocation and the difficulty in implementing it.

Although some variations may exist, the various entities that need to be allocated to rooms are usually common in academic institutions. There are approximately 30 different types of entities and among them there are: staff offices, research offices, storage/equipment/administrative rooms, library space, recreational/amenity rooms, lecture rooms, meeting rooms, laboratories and others.

All institutions prefer (and usually insist) that rooms allocated to the same department/faculty/school are located close to one another but of course this is not always possible. The level of closeness depends on the size of the group but complete buildings are often allocated to single or related groups. Where space is not too limited or groups are small, different groups may be allocated to different floors within shared buildings, but sometimes even floors have to be shared between groups. Some institutions have very different views as to what constitutes a good

allocation. An example presented in (Burke and Varley, 1998) is that most new universities (former polytechnics) in the UK are perfectly happy for lecturing staff to share offices. In most old universities, this is unlikely to be accepted.

Some academic institutions express a requirement to ensure that certain entities are allocated near to other entities. For example, departmental secretaries near to heads of departments, group leaders near to their research groups, etc. Departments may also require that all the lecture and meeting rooms are located close to each other or that all staff offices are on the same floor. The grouping conditions may be different according to the problem. For example, entities can be required to be together (same room), adjacent (next door rooms) or nearby (neighbouring rooms).

Sometimes, when allocating a specific entity to a room, additional requirements must be met. For example, lecture/examination rooms may need to have disabled access or audio visual aid facilities; library space may need to be located in a quiet area away from busy rooms and noisy equipment, etc. Such information must be available to judge whether additional costs or work must be committed before implementing the allocation (Diminnie and Kwak, 1986).

2.3.2. Manual Approach to Space Allocation

The manual process for allocating space in academic institutions varies from one case to another but it can be briefly described as follows (Burke and Varley, 1998):

In most UK academic institutions there is a centralised office that regulates the space distribution and assigns areas of space to faculties, schools, departments, etc. Space officers and administrators (heads of departments, group leaders, etc.) at different levels are in charge of the construction of an allocation. Then, the space necessary for each entity, the available space in rooms, the constraints that must be satisfied (hard constraints), those that are desirable to satisfy (soft constraints) and additional requirements are determined. With the aid of floor plans and room databases, information about the available areas of space is obtained (size, location, proximity, etc.). Entities are allocated to rooms in order of importance according to the specific situation. The satisfaction of space requirements and constraints is verified each time an entity is allocated. During this iterative process changes might

be necessary in order to produce a solution that satisfies as many requirements and constraints as possible. The evaluation of a solution involves multiple criteria and in some cases this criteria may come from different decision-makers. Due to the nature of this manual process, it is common that weeks or months are necessary to obtain a final solution.

2.3.3. The Multiobjective Nature of the Problem

The objectives pursued during the process of space allocation and the criteria used to evaluate the quality of an allocation depend on the problem instance. For example, while some academic institutions have a preference for optimising space utilisation, others have a preference for achieving a better functionality in the distribution of rooms. The satisfaction of preferences is another objective that is very difficult to measure and that is also important to consider when deciding how to assign room space. Of course, it is commonly the case that several conflicting objectives are present and then a compromise must be found. Moreover, the conditions for considering a solution as feasible also depend on the problem instance. In some cases it may be required to accommodate all the entities to the available space even if all the requirements/constraints cannot be fully satisfied. In other cases it may be that these requirements/constraints must be accomplished at the expense of some entities being left unallocated.

The constraints that limit the ways in which the room space can be distributed are also very problem-specific. For example, entities that must be allocated nearby each other or to the same room, preferences for allocating certain entities to specific rooms, entities that need to be allocated in a non-sharing basis, etc. Some of the constraints may be in conflict with each other or in conflict with the objectives. For example, it may be that a professor has to be allocated near to a laboratory and also near to their research students but there are no rooms that satisfy both constraints and space utilisation may also be affected. Considering the situation in which the available room space cannot be modified (i.e. construction work is not considered), the quality of an allocation can be measured in terms of the following aspects (not necessarily in this order of importance):

§ Number of allocated entities.

- Space utilisation, measured in terms of the amount of space wasted (areas of space not used) and the amount of space overused (entities with less space allocated to them than needed).

- Degree of satisfaction of additional requirements.

- Degree of satisfaction of the constraints.

Even when the evaluation function is carefully designed and takes into account all the different criteria, their relative importance and the way in which the space officers use these criteria to measure the quality of the allocation, a crucial observation can be made:

The best evaluated solutions produced by an automated system in the space allocation problem are not always the ones that would be finally selected by the space officers to be implemented in the real world.

The expert administrator often knows certain "constraints" which are not (or cannot, for political reasons) be built into the objectives. An example (which does occur) might be that two members of staff have a personality clash and cannot be located together. It might be politically sensitive to have this as a stated constraint. The administrator just *keeps it in his mind* when making the allocation. This observation leads us to the view that while automated space allocation methods certainly have huge potential for exploitation in higher education they are being developed to *aid* the administrators rather than to *replace* them.

It can be seen that due to the existence of a variety of conflicting objectives and constraints, requirements, feasibility conditions and evaluation criteria, the problem of distributing the room space in academic institutions is a complex multiobjective combinatorial optimisation problem. In the next section a formulation of the space allocation problem as approached in this thesis is presented.

2.4. Problem Formulation

As mentioned in section 2.3.1, the space allocation process is commonly carried out in three stages. In this thesis only the last stage is considered, that is, the allocation of

specific entities to rooms. This process is carried out with the aim of maximising the space utilisation and the satisfaction of specific requirements and constraints. The data required in this case includes:

§ Space requirements, i.e. the amount of space (floor area) that should be assigned to each entity.

§ Room size, i.e. the amount of space (floor area) that is available in each room for allocating entities.

§ Proximity relations between rooms, i.e. information that specifies, for each room, the list of rooms that are adjacent, near and distant.

§ Additional requirements and constraints, i.e. specific requirements and constraints (hard and soft) that impose limitations on how the entities can be allocated.

2.4.1. Types of Constraints

It is assumed here that all the entities for a given problem instance must be allocated using the available room space only. That is, feasible solutions must have, besides all hard constraints satisfied, all entities allocated. Since no additional space is available, some of the room space will be misused (wasted or overused). The types of constraints that exist in the test data sets used in this investigation are listed below. These data sets were prepared using real data from British universities and are described in detail in the next section. However, as explained above, different requirements and constraints may be applicable to different problem instances.

§ *Not sharing*. This is a unary constraint indicating that the entity should not share the room with other entities. For example, when senior or lecturing staff should have private offices. This may be hard in some cases and soft in others.

§ *Be located in*. This is a binary constraint indicating that there is a preference for allocating a specific entity to a specific room. For example, the situation in which it would be convenient that a computer room be allocated in a room with appropriate layout. This is considered a soft constraint in this thesis because when it must be satisfied, the entity is pre-allocated to the indicated room.

- *Be adjacent to*. This is a binary constraint indicating that one specific entity should be allocated adjacent to another. For example, when secretarial staff should be allocated in a room next to senior staff. When it is used, this is often a hard constraint but it can also be considered a soft one.

- *Be away from*. This is a binary constraint indicating that one specific entity should be allocated away from another entity or from a certain room. For example, when is preferred to allocate a lecture room away of noisy areas or communal rooms. This may be hard or soft.

- *Be together with*. This is a binary constraint indicating that two specific entities should be allocated in the same room. For example, this applies to the case when two researchers working on the same project should be in the same room. This is often soft.

- *Be grouped with*. This is a *q*-ary constraint indicating that a group of people should be allocated in the proximity of each other. For example, when all the members in the same research group should be allocated in a set of rooms that are close together. This is often as soft constraint.

Most of the constraint types listed above can be set as hard or soft depending on the particular problem instance. The exception is the constraint *be located in* which is always set as a soft constraint in the tests data sets used in this thesis. The reason for this is that in the cases where this constraint is set as hard, it is enough to fix the allocation of the given entity to the specified room.

2.4.2. Evaluation of an Allocation

Given the diversity in the criteria that space administrators use when evaluating the quality of the room space distribution in each particular case, it is very difficult to design an evaluation function that incorporates all the criteria with the adequate weighting. Besides, as explained in the previous sections, it is frequently the case that the final decision on which allocation will be implemented is affected by subjective criteria (and sometimes politics). Two overall (and often conflicting) objectives are aimed at in the space allocation problems considered in this thesis:

Minimise the space misuse. This objective is measured in terms of the space wasted and the space overused and it is equivalent to maximising the space utilisation. Here, wasting space is considered less serious than overusing space, therefore the weight for each unit of wasted space is one while the weight for each unit of space overused is two.

Minimise the violation of soft constraints. This objective is measured as minimising the penalty for violating the soft constraints. The penalties applied for the violation of each type of soft constraint are shown below. These penalty values were adjusted by experimentation following guidelines from space officers regarding the usual relative importance between these constraints in real world problems.

Soft Constraint	Penalties
not sharing	50
be located in	20
be adjacent to	10
be away of	10
be together with	10
be grouped with	5

The space allocation problem as described above can be formulated as follows:

m = number of available rooms

n = number of entities to allocate

h = number of hard constraints of the form $Z(k) = true$

s = number of soft constraints $Z(r) = true$

$c(i)$ = capacity or size of room i

$w(j)$ = space requirement of entity j

$x(i,j)$ = 1 if entity j is assigned to room i, 0 otherwise

$$\text{minimise} \qquad F(x) = \left(F1(x) + F2(x)\right) \tag{2.7}$$

$$\text{subject to} \qquad \sum_{i=1}^{m} x(i, j) = 1 \quad j = 1,2,\ldots,n \tag{2.8}$$

$$Z(k) = true \qquad k = 1,2,\ldots,h \tag{2.9}$$

where $$F1(x) = \sum_{i=1}^{m} \left(WP(i) + OP(i) \right) \tag{2.10}$$

$$F2(x) = \sum_{r=1}^{s} SCP(r) \tag{2.11}$$

Equations (2.10) and (2.11) measure space misuse and violation of soft constraints respectively. *WP(i)* expresses the penalty if the room capacity is wasted while *OP(i)* expresses the penalty if the room capacity is overused.

For the i^{th} room, there is space wastage if

$$c(i) > \sum_{j=1}^{n} w(j)x(i, j) \tag{2.12}$$

and then the penalty is given by

$$WP(i) = c(i) - \sum_{j=1}^{n} w(j)x(i, j) \tag{2.13}$$

For the i^{th} room, there is space overused if

$$c(i) < \sum_{j=1}^{n} w(j)x(i, j) \tag{2.14}$$

and then the penalty is given by

$$OP(i) = 2 \left(\sum_{j=1}^{n} w(j)x(i, j) - c(i) \right) \tag{2.15}$$

SCP(r) is the penalty applied if the r^{th} soft constraint is violated. A solution or allocation is represented by a vector $\pi = [\pi(1),\pi(2),\ldots,\pi(j)]$ where each element $\pi(j) \in \{1,2,\ldots,m\}$ for $j = 1,2,\ldots,n$ indicates the room to which the j^{th} entity has been allocated.

It can be noted from the formulation given above, that when only the space utilisation is considered, this problem is very similar to the multiple knapsack problem and the generalised assignment problem. What makes the academic space allocation more complicated to formulate and to solve is the existence of additional constraints that are also very problem-specific.

2.4.3. A Metric for Population Diversity

It was noted above that in the process of allocating room space in academic institutions it might be required to provide several solutions so that one allocation can be selected. Therefore, it is important to measure the degree of similarity between solutions in this problem. The metric used in this thesis to measure the degree of difference between two vectors representing allocations is described next. Space administrators suggested this metric as a meaningful way to express the variety of a set of allocations. From the perspective of space administrators, it is important to distinguish the number of positions in which two vectors representing allocations are different, i.e. the number of entities that are allocated to different rooms. For example, the following three vectors represent allocations that are completely different from each other: π_1={c,a,b,a}, π_2={b,b,a,c} and π_3={a,c,c,b}. Then, for a population of solutions, the percentage of non-similarity or variety used here as a diversity measure is given by eq. 2.16.

$$V(p) = \frac{\sum_{j=1}^{n} \frac{D(j)-1}{p-1}}{n} \cdot 100 \qquad (2.16)$$

where *D(j)* is the number of different values in the j^{th} position for all vectors and *p* is the population size. This metric measures the diversity of a set of allocations with respect to the solution space. Diversity in the solution space is the diversity that matters in this context so that the decision-makers can be provided with a set of competitive solutions and compare them in terms of their structure before selecting the final allocation (maybe after making some manual changes).

	Five strings representing allocations						
	A	A	A	A	A	A	A
	A	A	B	B	A	B	B
	A	B	B	C	B	C	C
	A	B	B	C	B	D	D
	A	B	B	C	C	D	E
D(j)	1	2	2	3	3	4	5
(*D(j)* – 1) / (*p* – 1)	0	0.25	0.25	0.50	0.50	0.75	1
V(p) = (3.25 / 7) x 100 = 46.42 %							

Table 2.1. Calculation of the population variety *V(p)*.

The non-similarity metric described above is an indication of the diversity in the allocation of entities to different rooms within a population of solutions. The way in which the population variety is calculated using the string representations of solutions is illustrated in figure 2.2. Consider the population of five strings (p = 5) representing allocations for a problem where seven entities have to be allocated (n = 7) and there are five available rooms (m = 5). The way in which the number of different rooms *D(j)* used within the population to allocate each of the entities and the population variety *V(p)* are calculated is illustrated below. Other population diversity metrics are described in (Morrison and De Jong, 2001).

2.5. Test Data Sets From UK Universities

Real data corresponding to the administration of academic space allocation in some of their schools/departments was available from the following universities: University of Nottingham, Nottingham Trent University and University of Wolverhampton. Using these data sets and following suggestions from space administrators, several test data sets were prepared for this investigation. These test data sets were designed to reflect different degrees of difficulty so that the performance of the algorithms proposed here could be assessed under different conditions. A brief description of the original data sets provided by the universities mentioned above and the test data sets prepared is given below.

University of Nottingham

This data corresponds to the distribution of offices in the School of Computer Science and Information Technology during the 1999-2000 academic year. There are 131 rooms with sizes ranging from 4.2 m^2 to 437.4 m^2 and distributed over one building with three floors. The total of 158 entities to be allocated are distributed as follows: 15 research rooms, 11 laboratories, 12 meeting rooms, 16 storage rooms, 6 professors, 1 reader, 5 senior lecturers, 25 lecturers, 16 research staff, 10 secretaries, 1 teaching assistant, 8 technicians and 32 research students. The space requirements of these entities range from 4 m^2 to 437 m^2. There are 263 constraints of which 111 are hard constraints and 152 are soft constraints. This is the most complete data set because all the information about the proximity between rooms is available and this permits us to make an accurate evaluation of the satisfaction of proximity constraints

(*be adjacent to*, *be away from* and *be grouped with*). This data set is called **nott1** in this thesis.

Nottingham Trent University

This data corresponds to a subset of the real distribution of space in the Chaucer Building during the 2000-2001 academic year. There are 73 rooms with sizes ranging from 9.94 m^2 to 132.43 m^2 and distributed over four floors. There is no information available on the physical proximity between rooms within each floor. Rooms are considered to be close to each other if they are located in the same floor and only this is considered to evaluate the satisfaction of proximity constraints. The total of 151 entities to be allocated are distributed as follows: 9 co-ordinators, 6 professors, 7 managers, 81 lecturers, 7 senior administrators, 32 administrative assistants and 9 technicians. The space requirements of these entities range from 3 m^2 to 18 m^2. There are 211 constraints, 80 hard constraints and 131 soft constraints. This data set is called **trent1** in this thesis.

University of Wolverhampton

This data corresponds to the distribution of offices in the SC Building in the Telford campus during the 1999-2000 academic year. There are 115 rooms with sizes ranging from 0.79 m^2 to 185.26 m^2. There is no information available about the physical proximity between rooms. There are 115 entities to be allocated including laboratories, staff rooms, computer rooms, teaching rooms, store rooms and common rooms but there is not a clear classification of this group of entities. There are 115 additional constraints, all of them sharing hard constraints. This data set is considered to be the least constrained and, in a sense, the easiest problem to solve. The reason for this is that the number of rooms and entities is the same and all the hard constraints forbid entities to share a room. Obviously this implies that a feasible solution is a one-to-one mapping between *n* and *m* and the goal is then to achieve an optimal utilisation of the available space. This data set is called **wolver1** in this thesis.

Summary of Test Data Sets

In addition to the three data sets described above, three more were prepared for the experiments carried out in this investigation. These three additional test data sets are subproblems of the **nott1** instance and were prepared to investigate various aspects on the performance of the metaheuristic approaches studied in this thesis. The **nott1** test instance was selected because it contains all information about proximity of rooms and it also includes a great variety of soft and hard constraints that permitted us to design tests problems with different degrees of difficulty. Some of the specific features of the three additional data sets are as follows. The test instance **nott1a** is highly constrained but the size of the problem (*n*,*m*) was reduced with respect to the original data set **nott1**. In the test instance **nott1b**, the number of hard constraints has been reduced considerably with respect to the number of soft constraints. Finally, the test instance **nott1c** is a smaller problem in which also the number of entities to allocate equals the number of available rooms ($n = m$). Table 2.2 below summarises the features of all the six test data sets. For more details refer to the following web site: http://www.cs.nott.ac.uk/~jds/research/spacedata.html.

	nott1		**nott1a**		**nott1b**		**nott1c**		**trent1**		**wolver1**	
n	158		142		104		94		151		115	
m	131		115		77		94		73		115	
constraints	*h*	*s*	*h*	*s*	*h*	*s*	*h*	*s*	*h*	*s*	*h*	*s*
not sharing	100	58	100	58	46	58	84	10	80	71	115	--
be allocated in	--	35	--	35	--	9	--	35	--	19	--	--
be adjacent to	5	15	5	15	4	10	5	15	--	5	--	--
be away from	6	14	5	12	1	2	5	12	--	--	--	--
be together with	--	20	--	20	--	20	--	--	--	36	--	--
be grouped with	--	10	--	10	--	9	--	10	--	--	--	--
total	111	152	110	150	51	108	94	82	80	131	115	--

Table 2.2. Characteristics of the test problems used in this thesis.

Chapter 3. Literature Review

3.1. Introduction

This chapter discusses previous work on applying computer optimisation techniques for the problem of allocating and/or planning space in academic institutions and it also looks at some applications for the optimisation of space in other scenarios such as industrial facilities and supermarkets. This chapter provides more evidence of the importance, complexity and diversity of this problem. Also, in this chapter the relation between the academic space allocation problem and other combinatorial problems is considered because previous research on similar problems has underpinned some of the ideas for the investigation presented in this thesis. In addition, an overview of complexity theory, the No Free Lunch theorem and metaheuristics is also presented in this chapter. Some of the sections in this chapter have been included in papers already published or submitted as follows. Sections 3.2 and 3.3 can be found in [Bur2001] while section 3.5.14 can be found in [Bur2003b] (see the appendix on page 199).

3.2. Previous Research on Space Optimisation

There are only a few reported applications in the literature on the optimisation of space usage in academic institutions. Ritzman et al. presented one of the earliest studies on the automated planning of academic facilities (Ritzman et al., 1980). Their application concentrated on the reassignment of 144 offices to 289 members in 6 academic departments of staff within the Ohio State University. Although the overall goal was to make the reassignment of offices as fair as possible, six conflicting objectives were identified:

- Assign enough offices to each department so that there is enough room space for all its members.
- Minimise the deviation of the assigned space to each department from the given space requirements.
- Equally distribute the offices equipped with air conditioning among the various departments.

- Minimise the physical distances between the rooms assigned to each department and its administrative office.
- Ensure that each department obtains a fair share of the high quality offices available.
- Minimise the number of reassignments, i.e. the number of offices assigned to a department which were not previously occupied by its staff members.

Ritzman et al. decided not to establish a-priory the preferences for each of the above objectives. In order to deal with the multiple objectives, they used a mixed-integer goal programming model to formulate the problem and linear programming as the solution method. An interactive program was implemented which permitted the decision-makers to obtain and compare different alternative layouts before producing a final compromise solution. The authors highlighted the importance of producing the various layouts in an interactive process because it permitted the administrators to be in command of the solution process and to have a set of alternative solutions from which to chose the most appropriate one.

Benjamin et al. also applied a linear programming approach but in their case the problem was not the distribution of rooms but the planning of a computer integrated manufacturing laboratory (Benjamin et al., 1992). The new laboratory was constructed due to the expansion of the department of engineering manufacturing at the University of Missouri-Rolla. The overall goal of this new lab was to stimulate the interest for teaching and research and after some debate and discussion it was decided that 15 sections would be located in the new laboratory. In addition to the desired space to be allocated to each of the sections, the following five goals (some of them conflicting) were previously specified:

- Increase the student use of the laboratory facilities.
- Develop new courses relying on the laboratory facilities.
- Stimulate the graduate-level and funded research.
- Increase the awareness of industry of the concepts developed in the laboratory.

§ Enhance the university's public image.

Before applying a linear goal programming algorithm to solve this planning problem, the goals listed above were prioritised and the authors highlighted that this required a substantial amount of time and knowledge from the decision-makers. In particular, they noted that the preference levels assigned to each goal by the different decision-makers revealed some inconsistencies in the subjective comparison between the goals. Therefore, extra work was required in order to review and adjust these preferences before setting the final values.

Another application of integer goal programming to the optimisation of academic space was reported in (Giannikos et al., 1995). The problem in this case consisted of reorganising the distribution of the academic space at the University of Westminster in the UK. Five objectives were identified and prioritised according to the preferences established by the decision-makers. The objectives are listed below in non-increasing order of their importance:

§ Assign enough offices to each school according to the standards in order to allocate lecturers, researchers and heads of school.

§ Allocate the adequate type of offices to schools according the standards.

§ Assign each office to only one school, i.e. only members of the same school can share a room.

§ Minimise the number of people that have to be relocated to reduce the disturbance during the transition period.

§ Minimise the distances from the rooms assigned to each school to its administrative centre.

In addition to these objectives, two additional hard constraints were imposed:

§ All heads of school must be allocated to an office with the exact requirements specified in the standards.

§ Each office can be used by members of staff in the same level or category. The three levels are: i) heads of school and similar, ii) heads of division and iii) lecturers, researchers and similar.

One of the observations that the authors made was that after comparing the actual distribution of offices with the one produced with the automated method it was clear that the space was being used in an inefficient way (at least according to the objectives and preferences defined). Although the authors did not indicate that the proposed solution was implemented, they highlighted that their ultimate goal was to provide the managers with a decision support tool to evaluate the current distribution of space and explore alternative allocations.

In all the studies mentioned above it is recognised that it is virtually impossible to allocate space in a way that conflicts of interest are completely eliminated due to the complex multiobjective nature of the problems. This reinforces the necessity for presenting to the decision-makers, a set of good solutions that can be used to negotiate and design the final space distribution.

3.3. Other Space Optimisation and Related Problems

3.3.1. Space Planning

The optimal utilisation of physical space is a goal not only in academic institutions but also in many other scenarios that range from industrial and commercial environments (Francis et al., 1992) to computer systems (Romero and Sanchez-Flores, 1990). Of course, the actual conditions, requirements and constraints may be very different from those present in the academic context. For example, in the facilities layout problem it is required to assign objects to locations considering distances and interactions between the objects. The objects can be physical facilities or activities such as administrative functions or personnel. Examples of the application of metaheuristics to facility layout problems can be found in (Bland, 1999; Bland, 1999b).

Sometimes, facility layout problems involve not only assigning the objects to locations but also designing the physical layout of the space, i.e. to partition the available space before assigning each partition (Kim and Kim, 1998). Most of the facility layout problems refer to the industrial and commercial scenarios where the main goals are to minimise the operation costs and to maximise the operational efficiency. An example is the planning and allocation of storage space to inventory in factories in order to minimise the costs of handling material, see (Larson and Kusiak, 1995; Kusiak, 2000). A review of heuristic approaches including constructive heuristics, iterative improvement strategies, simulated annealing, genetic algorithms and some other hybrid heuristics for solving facility layout problems is available in (Liggett, 2000).

3.3.2. Shelf Space Allocation

Among the applications of space management in commercial scenarios, the automated allocation of shelf space to products in supermarkets is an area that has received particular attention. The problem in this context is to select the products (and their quantities) to be placed on the shelves and then to determine where each product will be located taking into consideration retailing and operational requirements. A detailed description and elaborated model of the shelf space allocation problem are presented in (Yang and Chen, 1999) and examples of automated approaches to tackle this problem can be found in (Zufryden, 1986) and (Yang, 2001).

3.3.3. Constrained Variants of Knapsack Problems

There are some variants of capacity allocation problems that include other constraints apart from those related to the capacity of the container. These variants are mentioned here because the approaches investigated in this thesis can eventually be considered for capacity allocation problems with additional constraints. For example, a variant of the bin-packing problem in which there is a limit on the number of items that can be assigned to each bin is presented in (Kellerer and Pferschy, 1999) and some heuristics with guaranteed performance to solve that problem are analysed too. In the knapsack sharing problem, each item belongs to one or more owners therefore the objective function needs to be modified accordingly since each owner aims to

maximise the profit of his items (Yamada and Futakawa, 1997). In (Dawande et al., 2000) an analysis of the complexity and performance of approximation algorithms for the multiple knapsack problem with assignment restrictions is presented. In that variant each item can only be assigned to a subset of the available knapsacks. Another constrained variant of knapsack problems is the daily photograph scheduling problem (Vasquez and Hao, 2001). That problem consists of scheduling a subset of photographs from a set of candidate photographs to be taken by cameras in an earth observation satellite. The problem is modelled as a variant knapsack problem where in addition to the capacity constraints (memory available in the system) there are logic constraints that prevent certain combinations of photographs to be taken.

3.3.4. Related Scheduling Problems

Scheduling problems include a wide range of combinatorial optimisation problems and to some extent the academic space allocation problem can be considered within this group of problems. Scheduling can be described as the arrangement of objects (people, tasks, vehicles, lectures, exams, meetings, etc.) into a pattern in space-time in such a way that constraints are satisfied and certain goals are achieved (Wren, 1996). In most scheduling problems the goals include the creation of feasible schedules, efficient utilisation of available resources and the maximisation of schedule quality according to some predefined criteria. A schedule can be a sequence of processing jobs in production machines, an events timetable, an employee roster, a transport services routing or timetable, the assignment of events to places, etc. Among scheduling problems there are the following well-studied classes:

- Production scheduling: job shop, flow shop, open shop, etc.
- Transport scheduling or vehicle routing such as railway scheduling and bus timetabling.
- Personnel scheduling or timetabling such as nurse rostering, crew scheduling, etc.
- Maintenance scheduling such as electricity line maintenance and generator maintenance.

§ Events scheduling or timetabling such as examinations, courses, sport events, etc.

Some timetabling problems also involve assigning space resources to events. For example, room assignment is sometimes considered to be part and parcel of academic timetabling problems such as examination and course timetabling (Schaerf, 1999). Since the academic space allocation problem refers to efficiently assigning entities to rooms subject to additional constraints it can certainly be seen as related to some of the scheduling problems described above.

Considerable research has been carried out over the years in the area of automated scheduling and timetabling particularly in the application of metaheuristic techniques to solve these types of problems (e.g. Nagar et al., 1995; Burke et al., 1996; Dowsland, 1998; Colorni et al., 1998; Bagchi, 1999; Di Caspero and Schaerf, 2001; Varela et al., 2001; T'kindt and Billaut, 2002). Therefore it is important to consider the similarities that some of these problems have with the academic space allocation problem since some of the ideas and experiences can prove to be useful in this research area. It is not within the scope of this thesis to provide a survey or classification of scheduling problems or scheduling techniques investigated by other researchers. Instead, brief descriptions and references are provided whenever ideas and previous results that have been published in the literature are used in this thesis.

3.4. Complexity Theory and the No Free Lunch Theorem

3.4.1. Algorithms Complexity

The theory of algorithm complexity is concerned with the identification of problems that are computationally easy to solve and problems that are computationally hard to solve (Garey and Johnson, 1979; Rayward-Smith, 1986). This theory is also concerned with identifying those algorithms that are efficient and those that are inefficient from a computational point of view. From a broad perspective, the efficiency of an algorithm is assessed in terms of the computing resources that are needed to execute the algorithm and this includes execution time and space. The execution time is the number of steps that the algorithm takes to process the input and give an answer. The space is an indication of the amount of memory that is

needed to run the algorithm. However, in the theory of algorithms complexity the efficiency of algorithms is usually expressed in terms of its time complexity.

The time complexity is described by a function of the size of the input, which relates to the size of the problem instance. More specifically, the time complexity for an algorithm is described by its worst-case behaviour, which is the maximum number of basic operations that the algorithm is expected to perform for an input of size *n*. The time complexity of an algorithm is expressed using the notation $O(g(n))$ which is defined as follows. A function $f(n)$ is said to be $O(g(n))$ if there is a constant k such that $|f(n)| \leq k \cdot |g(n)|$ for $n \geq 0$. In other words, $O(g(n))$ refers to functions that do not grow faster than $g(n)$ and the $O(g(n))$ notation indicates that the algorithm's worst-case time complexity is bounded by $g(n)$.

Algorithms that have a time complexity described by a polynomial function (e.g. $O(4n)$, $O(n^3)$, etc.) are considered efficient because they can be run in reasonable amount of time for inputs of considerable size. However, if the time complexity of the algorithm is described by an exponential function (e.g. $O(3^n)$, $O(n^{\log n})$, etc.) then the algorithm is considered inefficient because it can be run in a reasonable amount of time only for inputs of small length, but for larger inputs running the algorithm becomes impractical. The difference between polynomial time algorithms and exponential time algorithms is the rate at which their computational time complexity grows given an increase in the size of the input (*n*). Remember, that the time complexity of an algorithm refers to the worst-case performance. There are some polynomial time algorithms that are not very useful in practice because *n* is typically large in practical instances. Also, there are exponential time algorithms regarded as useful because they can run quickly in practice due to small values of *n* encountered in practical instances.

3.4.2. Problem Complexity – The P and NP Classes

The computational complexity of a problem is determined by the best algorithm that can be found to solve the problem (Garey and Johnson, 1979). At a high level of abstraction, if a polynomial time algorithm can be found for a given problem, then the problem is considered tractable or not so hard. But if no such algorithm can be

found for the problem, i.e. only exponential time algorithms can be constructed, the problem is considered intractable or very hard even if the problem is solvable. The theory of computational complexity has been developed considering mainly decision problems. Most optimisation problems can be expressed as a decision problem. A decision problem is a problem for which the answer is 'yes' or 'no' according to whether the input satisfies the given conditions in the problem. Some examples of decision problems are given below.

EVEN. Given a natural number *n*, is *n* an even number? The answer is 'yes' if *n* is even or 'no' if *n* is odd.

PRIME. Given a natural number *n*, is *n* a prime number? The answer is 'yes' if *n* is prime or 'no' if *n* is composite.

SATISFIABILITY. Given a Boolean expression $f(x_1, x_2,\ldots x_n)$, can the variables $x_1, x_2,\ldots x_n$ be fixed to values that make the value of *f* true? The answer is 'yes' if there is a setting of the variables that makes *f* true and 'no' otherwise.

HAMILTONIAN CYCLE. Given a graph *G(V,E)* with *N* nodes, is there a cycle of edges in *G* that includes each of the *N* nodes? The answer is 'yes' if such cycle exists and 'no' otherwise.

The space allocation problem described in chapter 2 can also be stated as a decision problem:

SPACE ALLOCATION. Given *n* entities and *m* available rooms, is it possible to construct an allocation of the *n* entities to the *m* rooms in such a way that all existing constraints (hard and soft) are satisfied and the space misuse is at most *W*? The answer is 'yes' is such an allocation exists and 'no' otherwise.

In the rest of this section, we refer to decision problems simply as problems. There are two classes in which problems are classified, the P and NP classes (Garey and Johnson, 1979, Rayward-Smith, 1986). The class P includes all those problems for which an efficient (polynomial time) deterministic algorithm has been found. The class NP includes all those problems for which a *non-deterministic* polynomial time algorithm is known to solve the problem (NP stands for non-deterministic

polynomial). A non-deterministic algorithm can be described as consisting of two stages. The first stage guesses a structure for the problem and the second stage verifies if the given structure is or is not a solution to the problem. Then, the algorithm is said to be a non-deterministic polynomial time algorithm if for each instance of the problem there is a guess that can be verified by the deterministic phase for answer 'yes' in a polynomial time.

Then, if P are problems solved in polynomial time by deterministic algorithms and NP are problems solved in polynomial time by non-deterministic algorithms, the question is whether P = NP or P ≠ NP. In fact, this is the most important open question in computational complexity theory. It is clear that P ⊆ NP, which means that non-deterministic algorithms are more powerful that deterministic algorithms. If there is a deterministic algorithm for a problem, a non-deterministic one can be constructed by simply not using the guessing stage.

For many problems proved to be in the class NP no efficient algorithm has been found. This strengthens the belief that P ≠ NP but this conjecture is still not proven. There are many problems known to be in NP for which no efficient algorithm has been found and these problems are considered NP-hard in the strong sense. Examples of these problems are the multiple knapsack problem and the generalised assignment problems described in chapter 2 and it is generally believed that no efficient algorithm exists for these (and all other NP-hard) combinatorial problems, i.e. they are intractable.

If it is true that P ≠ NP, then the problems in the set NP − P are intractable. Therefore, when tackling a particular problem, it is important to know if the problem belongs to the class of tractable or intractable problems. One way of doing this is to determine whether the problem of interest in or not related to another problem that is already known to be tractable or intractable. Reducing one problem to another is the technique used to demonstrate if the two problems are related or not. Reduction is to provide a transformation that permits to map one instance of the first problem into one instance of the second problem. This transformation permits to convert one algorithm that solves one problem into an algorithm that solves the other problem.

There is an important class of problems in NP, this is the class NP-complete. The first work towards the theory of NP-completeness was reported by Cook in 1971 (Cook, 1971). Among other results, Cook proved that any problem in NP can be reduced to the satisfiability problem. This means that if there is an efficient algorithm to solve the satisfiability problem, then any problem in NP can also be solved by an efficient algorithm. These problems are said to be NP-complete and are considered the hardest in NP in a sense. This is because if no single NP-complete problem has an efficient algorithm to solve it, then none of them has an efficient algorithm and they are all intractable. Many problems have been proven to be NP-complete (or reduced to the satisfiability problem) but it is still not proved that these problems are intractable. However, it is generally assumed that finding an efficient algorithm for any problem in NP-complete is unlikely.

Then, if a problem is NP-complete and P ≠ NP then the problem belongs to the set NP – P. In other words, the problem (and all in NP-complete) could belong to P only if P = NP. Then, if it is assumed that NP-complete problems are intractable, i.e. P ≠ NP, then when a problem is known to be NP-complete the focus should not be on finding efficient algorithms. Instead one should aim to design algorithms that produce high-quality solutions with no guaranteed optimality, i.e. design useful algorithms to tackle the problem in practice.

3.4.3. Approaches to Solve Optimisation Problems

As discussed above, the complexity of a problem and the complexity of an algorithm to solve the problem gives an indication of how hard it is to solve the problem from a computational view point. An exact algorithm is capable of solving a given instance of a combinatorial optimisation problem to optimality. However, the time complexity of some exact algorithms is bounded by an exponential function, which makes these algorithms inefficient. The interest and practical significance of the concept of NP-complete problems lies in the widespread belief that an efficient algorithm for solving such problems does not exist and that algorithms that produce high quality (or near-optimal) solutions in a reasonable amount of time are then needed. Such algorithms are known as heuristic methods (as well as a number of similar names).

A heuristic is defined in (Reeves, 1995) as a "*technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is*". Examples of heuristics are constructive algorithms (also known as greedy methods). These are very simple heuristics that construct the solution in a series of steps based on the strategy of making the best decision (based on a certain criterion) at each step. Another example of heuristic methodology is local search (also known as neighbourhood search) where neighbouring solutions are explored in an attempt to improve the solution (although worse solutions can be accepted as an interim step – see below for more details). A gentle introduction to heuristic approaches is provided in (Michalewicz and Fogel, 2000).

More advanced heuristic approaches called metaheuristics have been widely developed and applied to a variety of optimisation problems over the last two decades or so (e.g. Glover and Kochenberger, 2003; Voss et al., 1999; Aarts and Lenstra, 1997; Osman and Kelly, 1996; Osman and Laporte, 1996; Rayward-Smith et al., 1996; Reeves, 1995). A metaheuristic is described in (Voss et al., 1999 page ix) as "*an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method*".

When solving combinatorial optimisation problems, there are exact algorithms that, given enough time, can guarantee to find an optimal solution. There are also very specialised heuristics that exploit knowledge of the problem domain and produce solutions of good quality. There are also metaheuristics that are not designed specifically for a particular problem but are considered general approaches that can be tuned for any problem. Some metaheuristics may need tuning while others act as a black box because they can be implemented with none or very little information about the problem being solved. An example of such black-box approach is random search, which can be used to compare the performance of other algorithms.

3.4.4. The No Free Lunch Theorem

The interest on developing metaheuristic approaches for difficult combinatorial optimisation problems such as the one tackled in this thesis is because of the time complexity of these problems and because of the implications of the No Free Lunch Theorem (NFL) of Wolpert and Macready (Wolpert and Macready, 1995; Wolpert and Macready, 1997). The NFL theorem states that the averaged performance across all possible problems is the same for all algorithms. In other words, considering all possible problems, all algorithms perform equally and therefore, no distinction can be made between two algorithms because there are as many problems for which one algorithm performs better than the second one as for which the reverse is true. However, in some circumstances the comparison of two algorithms A_1 and A_2 can be made. If there are some problems for which the solutions obtained by A_1 are much better than those obtained by A_2, then if the NFL theorem holds, it may be the case that there are many problems for which A_2 performs better than A_1 but only for a small amount. Hence, if the problems in our interest are those for which A_1 is better than A_2, then it is possible to make a distinction between the two algorithms.

The above implies that it is essential to incorporate knowledge of the problem domain into the algorithm. Otherwise, the algorithm is as likely to perform better than random search as it is likely to perform worse. One conclusion that can be obtained from the NFL theorem is that to solve any problem, the algorithm needs to be adapted by taking into consideration the specific characteristics of the problem. This motivates the interest in the investigation of applying and adapting metaheuristics approaches to the space allocation problem in this thesis.

3.5. Review of Metaheuristic Approaches

3.5.1. Introduction

This section presents a brief overview of some of the most well known and successful metaheuristic approaches presented in the literature. The aim is to provide a consistent theoretical background on the field of metaheuristics for combinatorial optimisation to underpin the rest of this thesis. A review on the main concepts,

terminology, classifications, algorithms description and relevant applications is presented.

3.5.2. Classification of Metaheuristics

There are several possible classifications of heuristics and metaheuristics but one that is commonly used and that certainly allows us to embrace most metaheuristics including their hybrids is: *single-solution approaches* and *population-based approaches* also called *single-point* and *multiple-point* respectively (Blum and Roli, 2001). Examples of single-solution methods are: basic local search (deterministic iterative improvement), simulated annealing, tabu search, greedy randomised adaptive search procedure, variable neighbourhood search, guided local search, iterated local search and others. Population-based methods include: genetic algorithms, scatter search, ant colony systems, memetic algorithms, evolutionary strategies (although some of them are single-solution), particle swarm systems, cultural algorithms, etc. If a single-solution approach is hybridised with a population-based approach (e.g. a memetic algorithm can be defined to be a genetic algorithm incorporating local search) then the result is, of course, a population-based approach.

Sometimes, researchers classify heuristic and metaheuristic approaches into *nature-inspired* and *non-nature inspired* and many refer to the first group as evolutionary algorithms. While these algorithms are commonly conceptualised as those approaches that simulate various aspects of natural evolution (Bäck et al., 1997), some researchers argue that a fundamental characteristic of evolutionary algorithms is that they handle a population of individuals (Calegari et al., 1999; Hertz and Klober, 2000). As noted in (Blum and Roli, 2001), sometimes it is difficult to clearly identify the genesis of an algorithm. In addition, many hybrid metaheuristics do not fit well into the above classification.

An alternative classification of heuristic approaches is based on whether the algorithms use memory during the search (Taillard et.al, 2001). In that classification, memory is considered to be any mechanism that is explicitly used to store a set of solutions or parts of solutions. Taillard et al. sketch adaptive memory programming approaches as those algorithms that perform the following steps. First, the algorithm initialises the memory. Then, in an iterative process, the algorithm generates new

provisional solutions using the data stored in the memory, improves these generated solutions using local search and updates the memory using the pieces of knowledge brought by the new generated solutions.

3.5.3. Constructive Heuristics

Constructive (greedy) heuristics exist for many combinatorial optimisation problems and some of these methods can produce an acceptable or acceptably good solution in a reasonable computation time, depending upon the problem solving situation in hand. Although in most cases the solutions produced are not considered to be near-to-optimal, they can be improved in a subsequent more intensive search if the initial solutions are constructed using a greedy heuristic (Burke et al., 1998; Corne and Ross, 1996). A constructive heuristic builds a solution progressively in a number of iterations. It is commonly the case that the number of iterations equals the number of variables in the combinatorial optimisation problem. Then, in each iteration, the heuristic assigns a value to one of the variables until a complete solution is constructed. The heuristic selects the value that maintains the solution's feasibility and produces the best result based on a predefined criterion. The suitability of initialising each variable is calculated using the predefined criterion at the beginning of the process and the order is maintained static during the construction. This means that for the same problem instance and the same predefined criterion, a greedy heuristic generates the same solution every time it is executed.

3.5.4. Simple Local Search

Once a solution is initialised either randomly or with a constructive heuristic, it can be iteratively improved using local search heuristics that explore the neighbourhood of the present solution (e.g. Aarts and Lenstra, 1997). The neighbourhood of a solution is the set of solutions that are close to it in *some sense*. The local optima is the best solution(s) in the defined neighbourhood. Then, local search is also known as neighbourhood search. The global optima is a term used to describe the best solution(s) with respect to the whole solution space. Plateaus are regions of the solutions space where no neighbourhood is better but a number of them are as good as the present solution.

A neighbourhood function or neighbourhood structure maps each solution $x \in S$ into a set of solutions $N(x) \in S$ where S is the solution space, $N(x)$ is the neighbourhood of x and each solution in $N(x)$ is a neighbour of x. For example, many combinatorial optimisation problem solutions can be represented as sequences or partitions. These solution representations permit the use of k-exchange neighbourhood structures, i.e. by exchanging k elements in a given sequence or partition a neighbour solution is produced. A move in local search is the change defined by the neighbourhood structure that is made to the current solution in order to produce a neighbouring solution. Given a solution x, each neighbourhood structure specifies a set of solutions that are "close" to x. The neighbourhood size $|N(x)|$ is the number of neighbouring solutions that can be reached from the solution x.

Then, local search heuristics attempt to improve the current solution by exploring neighbourhoods. The first important choice is the neighbourhood structure(s). A given neighbourhood with a manageable size has a certain *strength*. A strong neighbourhood produces local optima that are largely independent of the quality of the initial solution while a weak neighbourhood produces local optima that is highly correlated to the initial solution (Papadimitriou and Steiglitz, 1982). The next choice is how to explore solutions in the neighbourhood(s) and some of the possible ways are described below.

Deterministic Iterative Improvement

The basic local search strategy or deterministic iterative improvement assumes a given neighbourhood and an initial solution. One neighbour is generated in each iteration and it replaces the current solution only if it is better. The search finishes when no better neighbouring solution is found.

First and Best Iterative Improvement

Exploring only one neighbouring solution often leads to poor local optima. An option is to generate a subset of the neighbouring solutions or all of them depending on the size of the neighbourhood. If the first neighbouring solution that is better than the current one is accepted, one obtains a *first iterative improvement* algorithm. When

the best of all the neighbours is selected, the approach is called *best iterative improvement* algorithm.

Iterative improvement algorithms are also referred to as hill-climbing methods in maximisation problems or as descent methods in minimisation problems. Iterative improvement algorithms can be described using the pseudocode shown in figure 3.1.

Step 1. Generate initial current solution *x*.
Step 2. Explore neighbourhood of current solution *x* and generate candidate solution *x'*.
Step 3. If fitness(*x'*) > fitness(*x*) then *x* = *x'*.
Step 4. If stopping condition met finish, otherwise go to Step 2.

Figure 3.1. Iterative improvement algorithm. Deterministic improvement explores only one neighbour in step 2 while first improvement and best improvement explore a set of them.

Other Extensions to Local Search

Local search heuristics that accept only improving solutions are simple and easy to implement but they often produce local optima of low quality. Various strategies to avoid getting stuck in poor local optima have been incorporated into local search producing a number of metaheuristic approaches. These strategies aim to establish an adequate compromise between intensification and diversification. Intensification refers to *focusing the search into certain regions of the solution space* while diversification refers to *expanding the search by exploring unvisited regions of the solution space*. The intensification and diversification mechanisms can be fundamental components of the searching method or additional strategies incorporated by the designer with or without knowledge of the problem domain. A dynamic and adaptive compromise between the intensification and diversification phases is commonly regarded as desirable to achieve good results, but very few metaheuristics actually incorporate such a mechanism. Strong diversification strategies are good for sampling the solution space and identifying promising areas while strong intensification strategies are good for focusing and exploring these promising areas in search of elite solutions.

In (Vaessens et al., 1998) a local search template that attempts to capture most of the variants of local search algorithms was proposed. In that template the authors identified the following strategies that contribute to the design of more elaborate local search procedures:

§ Generate all or a subset of the solutions in the given neighbourhood structure.

§ Restart the search from different generated initial solutions.

§ Use more elaborate criteria to even accept non-improving solutions.

§ Replace the current solution by a population of current solutions.

§ Design more than one neighbourhood structure to be used during the search.

The local search template mentioned above classifies algorithm variations based on two aspects: the number of current solutions (point-based and population-based) and the number of search strategies or neighbourhood structures used (single-level and multi-level). More elaborate methods such as genetic algorithms (see section 3.5.12) are described in the template as an instance of single-level population-based algorithms.

3.5.5. Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP is an iterative process that combines a randomised constructive heuristic and local search and is based on the strategy of restarting the search from different initial solutions (Glover and Kochenberger, 2003). In each iteration, a solution is generated with the randomised constructive heuristic and then the solution is improved by means of local search. The best solution over all iterations is kept and reported as the result at the end of the search. At each constructive step the suitability of each remaining non-initialised variable is calculated according to the status of the partial solution. Then, the variables are sorted according to their suitability and a sublist is formed. From this restricted candidate list the next variable to be initialised is chosen at random instead of selecting the most suitable one as in a greedy heuristic.

Step 1. Start with an empty solution *x*.
Step 2. Calculate suitability of each non-initialised variable.
Step 3. Sort the non-initialised variables and generate the restricted candidate list.
Step 4. Select and initialise one element at random from the restricted candidate list.
Step 5. If the solution *x* is still incomplete go to Step 2.
Step 6. Apply *Local Search* to solution current solution *x* to generate *x'*.
Step 7. Memorise the best solution found so far.
Step 8. If stopping condition met then finish, otherwise go to Step 1.

Figure 3.2. Greedy randomised adaptive search procedure.

Figure 3.2 shows the pseudocode of the GRASP metaheuristic. The local search phase in the greedy randomised adaptive search procedure (step 6) can be any simple or more elaborated improvement method. Important conditions that should be met for GRASP to be successful are that the constructive and the local search phases must complement each other well and the latter should generate solutions that lie in promising areas of the solution space.

3.5.6. Guided Local Search (GLS)

Guided local search (see pseudocode in figure 3.3) is a metaheuristic that employs the strategy of modifying the search landscape by changing the objective function (Glover and Kochenberger, 2002). The purpose of using modified objective functions in guided local search is to escape from the local optimal by gradually reducing its attractiveness. The algorithm starts with an initial solution that is improved by local search until a local optima is found. Then, in each iteration the original objective function *f(x)* is adapted to obtain the modified objective function *f'(x)* and the local search is restarted.

Step 1. Generate initial current solution x.
Step 2. Apply *Local Search* to solution x to generate local optima x^* and using $f'(x)$.
Step 3. Modify the objective function $f'(x)$ according to the search history.
Step 4. If stopping condition met then finish, otherwise go to Step 2.

Figure 3.3. Guided local search metaheuristic.

The guided local search method is very simple and the critical component is the tactical change induced in the objective function, which is now explained in more detail. First, it is necessary to identify a set of q properties or features that may (or may not) be present in a solution and which serve to discriminate between solutions. Then, weights are associated to the q solution features to establish their relative importance. The modified function $f'(x)$ is given by:

$$f'(x) = f(x) + \lambda \sum_{i=1}^{q} p_i I_i(x) \qquad (3.1)$$

where p_i is the weight or penalty parameter associated to feature i, $I_i(x)$ is a Boolean indicator of whether the feature i is present or not in the solution x and λ is the

regularisation parameter that established a balance between the importance of solution features with respect to the original objective function *f(x)*. To adapt the objective function, some of the *q* penalty parameters are increased in each iteration. The penalties changed are those corresponding to the solution features that have a maximum utility. This utility is given by,

$$utility\,(i) = I_i\,(x) \cdot \frac{c_i}{1 + p_i} \qquad (3.2)$$

where c_i is the cost assigned to each feature *i* measuring its relative importance with respect to the other solution features.

Adapting the penalty parameters is a critical design decision when implementing guided local search because this will affect how the objective function and hence the search landscape is adapted during the search. The strategy for changing the penalty parameters is normally very dependent on the problem domain but it should encourage the use of the search history and avoid making the search landscape too rugged.

3.5.7. Iterated Local Search (ILS)

Iterated local search is a metaheuristic that combines local search with a perturbation operator (Glover and Kochenberger, 2003) The algorithm starts with an initial solution and performs local search until a local optimum is found. Then, the current solution is perturbed and a different local optimum is obtained by performing local search. Finally, acceptance criteria based on the search history are used to decide whether the perturbed solution or the new local optimum becomes the current solution in the next iteration. Figure 3.4 shows the pseudocode of this metaheuristic.

Step 1. Generate initial current solution *x*.
Step 2. Apply *Local Search* to solution *x* to generate local optima x^*.
Step 3. Perturb solution x^* to produce *x'*.
Step 4. Apply *Local Search* to *x'* to generate new local optima x'^*.
Step 5. If the acceptance criteria is satisfied then $x^* = x'^*$.
Step 4. If stopping condition met then finish, otherwise go to Step 3.

Figure 3.4. Iterated local search metaheuristic.

The way in which the perturbation operator, acceptance criteria and search history are designed and implemented permits a high degree of flexibility for tuning

iterated local search according to the problem domain. The perturbation operation must be designed in such a way that escaping from the local optima to explore other areas of the solution space is possible without turning into a completely random restart. The acceptance criterion can simply be to accept the new local optimum if it is better than the best solution so far or it can be a more elaborate criterion based on threshold acceptance (see section 3.5.9 below).

3.5.8. Variable Neighbourhood Search (VNS)

The variable neighbourhood search metaheuristic is based on the strategy of using more than one neighbourhood structure during the search (Mladenovic and Hansen, 1997). The main idea is to change the neighbourhood structure in a systematic way as the search progresses. First, *k* neighbourhood structures need to be defined. The algorithm is made of three phases: *shaking*, *local search* and *move* (see pseudocode in figure 3.5).

During shaking a random solution is generated from the current solution using the k^{th} neighbourhood structure. In the local search phase this randomly generated solution is improved and if it is better than the current solution it replaces it. In the move phase the next neighbourhood to be used is chosen based on whether or not the previous local search phase was successful or not. Intensification is achieved by the local search while the systematic change of the neighbourhood structure acts as a diversification mechanism. It is important to design good neighbourhood structures of increasing cardinality that present different views of the search landscape and allow the shaking phase to generate new starting solutions that lie near new local optima. There exist other variants of variable neighbourhood search such as variable neighbourhood decomposition search (VNDS), skewed variable neighbourhood search (SVNS) and given the flexibility of the technique, other variants of this algorithm can be employed (Hansen and Mlandenovic, 2001).

Step 1. Generate initial current solution x.
Step 2. Select neighborhood structure N_1, i.e. $k = 1$.
Step 3. Generate candidate solution x' from x using the neighborhood structure N_k.
Step 4. Apply *Local Search* to solution x' to generate x^*.
Step 5. If fitness(x^*) > fitness(x') then $x = x^*$ and $k = 1$.
Step 6. If fitness(x^*) < fitness(x') then $k = k + 1$.
Step 7. If $k < k_{max}$ then go to Step 3.
Step 8. If stopping condition met finish, otherwise go to Step 2.

Figure 3.5. Variable neighbourhood search metaheuristic.

3.5.9. Threshold Acceptance Algorithms

Threshold acceptance algorithms are modified versions of improving heuristics where non-improving solutions are also accepted if a given condition is met. Figure 3.6 shows the pseudocode for this technique. The condition is that the fitness difference between the current and the non-improving candidate solution be smaller that a given threshold. The threshold can be fixed during the whole search:

$$\text{threshold}\ (t + 1) = \text{threshold}\ (t)$$

or it can be varied as the searches progresses:

threshold $(t) \geq$ threshold $(t + 1)$ and lim $(t \rightarrow \infty)$ threshold $(t) = 0$ for the iteration t.

Step 1. Generate initial current solution x.
Step 2. $t = 0$.
Step 3. threshold(t) = $f(t)$.
Step 4. Generate candidate solution x' from current solution x.
Step 5. If fitness(x') – fitness(x) < threshold(t) then $x = x'$.
Step 6. $t = t + 1$.
Step 5. If stopping condition met finish, otherwise go to Step 3.

Figure 3.6. Threshold acceptance metaheuristic. In step 3, *f(t)* gives the threshold for the iteration *t*.

3.5.10. Simulated Annealing (SA)

Simulated annealing is an optimisation method that was inspired from the Metropolis algorithm for statistical mechanics (Metropolis et al., 1953). Simulated annealing is a metaheuristic that attempts to avoid getting stuck in poor local optima by exploring other areas of the solution space (Kirkpatrick et al., 1983, Aarts and Korst, 1998) and it is a probabilistic version of threshold acceptance. The main idea is that improving candidate solutions are always accepted while non-improving candidate solutions are accepted with a certain probability. This probability of accepting non-improving solutions is calculated according to the current *temperature* of the algorithm.

Step 1. Generate initial current solution x.
Step 2. Temperature = Initial Temperature.
Step 3. Generate candidate solution x' from current solution x.
Step 4. If fitness(x') > fitness(x) then $x = x'$.
Step 5. If fitness(x') ≤ fitness(x) then calculate Acceptance Probability.

Step 5.1 If Acceptance Probability > random [0,1] then $x = x'$.
Step 6. Update Temperature according to Cooling Schedule.
Step 7. If stopping condition met finish, otherwise go to Step 3.

Figure 3.7. Simulated annealing metaheuristic. In step 5 the acceptance probability is calculated according to the current temperature while in step 6 the current temperature is updated according to the cooling schedule.

The algorithm starts with a high initial temperature, which corresponds to a high probability of accepting non-improving solutions. The temperature is gradually decreased as the search progresses so that the probability of accepting non-improving solutions is also reduced. At temperature zero the algorithm operates like an improving heuristic, i.e. only improving solutions are accepted. The search process can remain at temperature zero until the stopping condition or it can be reheated, i.e. the temperature is increased and reduced periodically. Two specific decisions have to be made for this algorithm: a) the choice of cooling schedule, i.e. the initial temperature and rules for varying it during the search and b) the choice of acceptance probability function, i.e. how to determine, according to the current temperature, the probability of accepting non-improving solutions. Figure 3.7. shows the pseudocode for the simulated annealing metaheuristic.

The Cooling Schedule

In general, the cooling schedule is determined by:

a) Initial temperature.
b) Decrement step, i.e. number of iterations between temperature decrements.
c) Cooling factor, i.e. the proportion of the temperature reduction.
d) Reheating step, i.e. number of iterations after which the temperature is increased to the initial temperature or to another value.

Some cooling schedules reduce the temperature in a monotonic fashion and it has been suggested that optimal cooling schedules may not be monotonic (Reeves, 1995). The selection of an adequate cooling schedule and all its associated parameters has been extensively studied and has been found to be dependant upon the problem domain. An analysis and comparison of various cooling schedules (when the computing time is limited) is provided in (Strenski and Kirkpatrick, 1991). The performance of a simulated annealing algorithm with different cooling schedules

on the course timetabling problem is investigated in (Elmohamed et al., 1998) while (Thompson and Dowsland, 1996; Thompson and Dowsland, 1996b) carried out a similar comparison on the examination timetabling problem. Some examples of cooling schedules that have been proposed and investigated in the literature are described as follows.

Arithmetic Cooling Schedule.

$$T_i = (T_{i-1}) - \Delta T \qquad (3.3)$$

T_i is the new temperature value, T_{i-1} is the previous temperature value and ΔT is the amount of temperature reduction, which is usually kept constant.

Geometric Cooling Schedule.

$$T_i = \alpha \cdot T_{i-1} \text{ where } 0 < \alpha < 1, \text{ with } \alpha \approx 1. \qquad (3.4)$$

or

$$T_i = (\alpha \cdot T_{i-1}) / (1 + \alpha \cdot T_{i-1}) \text{ where } 0 < \alpha < 1, \text{ with } \alpha \approx 0. \qquad (3.5)$$

T_i is the new temperature value, T_{i-1} is the previous temperature value and α determines the cooling factor.

Quadratic Cooling Schedule.

$$T_i = a \cdot i^2 + b \cdot i + c \text{ where } a = \frac{T_1 T_f}{I_{total}}, b = 2\frac{T_f - T_1}{I_{total}}, c = T_1 \qquad (3.6)$$

T_1 and T_f are the initial and final temperature values respectively while I_{total} is the total number of iterations of the algorithm.

Heuristic Cooling Schedules.

Heuristic cooling schedules reduce the temperature by taking into account the history of the search. One example of a heuristic temperature control is reheating as a function of the cost as described in (Elmohamed et al., 1998). In that cooling schedule, the temperature is raised according to the specific heat. The specific heat is a measure of the variance of the fitness values of the solutions visited at a given temperature level. At each temperature level T_i the average fitness of the visited

solutions is denoted by $F(T_i)$ and $\sigma^2(T_i)$ denotes the variance of the fitness at that temperature level. Then, the specific heat at the temperature level T_i is given by

$$C_H(T_i) = \frac{\sigma^2 T_i}{T^2} \tag{3.7}$$

The temperature at which the specific heat is maximum can then be found and it is denoted by $T(C_H^{max})$. The cooling schedule reheats the temperature after a predefined number of iterations without improvement (reheating step) according to the following equation:

$$T_{i+1} = k \cdot F_{best} + T(C_H^{\max}) \tag{3.8}$$

where k is a tuneable parameter and F_{best} is the best fitness so far. The temperature can be decreased using an arithmetic or geometric cooling schedule.

Another example of a heuristic cooling schedule is the adaptive cooling described also in (Elmohamed et al., 1998). Here, the temperature reduction is controlled based on the specific heat as given by equation 3.9 and then reheating may or may not be used.

$$T_{i+1} = T_i \cdot \exp^{-\frac{aT_i}{\sigma(T_i)}} \tag{3.9}$$

where a is a tuneable parameter and $\sigma(T_i)$ is the standard deviation of the fitness at temperature level T_i.

(Aarts and Korst, 1998) proposed a cooling schedule that reduces the temperature very quickly during the first iterations and then, as the temperature decreases, the reduction rate is also slowed down. The temperature is reduced according to the following formula:

$$T_{i+1} = \frac{T_i}{1 + \frac{T_i \ln(1+\delta)}{\delta^*}} \tag{3.10}$$

where δ^* is the maximum difference between the global optimum (if known) and any feasible solution and δ is theoretically the maximum proportional change allowed for any temperature level. Suggested values are $\delta^* = 3\sigma_i$ where σ_i is the standard

deviation of the current solution fitness value while using the temperature level T_i and $\delta = 0.1$.

Other heuristic cooling schedules are those described in (Osman, 1995) and given below by equations 3.11 and 3.12.

$$T_{i+1} = T_{reset} \text{ and } T_{reset} = \max\left(\frac{T_{reset}}{2}, T_{found}\right) \qquad (3.11)$$

T_{reset} is a higher temperature than the current value T_i and T_{found} is the temperature value at which the best solution so far was found. The temperature is incremented using the above relation only after the whole neighbourhood (assuming this can be done) has been explored and no better solution has been found. Increasing the temperature permits us to escape from the current neighbourhood but without too much deviation from the best solution visited so far.

$$T_{i+1} = \frac{T_i}{(1 + \beta_i T_i)} \text{ and } \beta_i = \frac{T_1 - T_f}{(\alpha + \gamma\sqrt{i}) \cdot T_1 \cdot T_f} \qquad (3.12)$$

T_1 and T_f are the initial and final temperature values respectively and suggested values for α and γ are: $\alpha = |N(x)| \cdot N(x)_{\text{feasible}}$ and $\gamma = |N(x)|$, where $N(x)_{feasible}$ is the total number of feasible moves in the neighbourhood $N(x)$ of the current solution.

Acceptance Probability Function

As with the cooling schedule, several functions to calculate the acceptance probability have been proposed, but the most widely used is the Boltzmann-like distribution (Aarts and Korst, 1998):

Acceptance probability = exp (- $\Delta F/T_i$) where ΔF = fitness(x') – fitness(x) and T_i is the current temperature.

Remarks

Broadly speaking, simulated annealing can find good solutions for a wide variety of problems, it is easy to implement and is capable of handling almost any optimisation problem and any constraint. On the other hand, some of the difficulties reported with

this method are long run times, the need for fine-tuning and the necessity for good neighbourhood structures design. An interesting research avenue is the challenge to design parallel versions of the simulated annealing algorithm. This is a task that, although promising, is not trivial because of the intrinsic sequential nature of the algorithm (Abramson, 1991).

3.5.11. Tabu Search (TS)

Tabu search is a metaheuristic that attempts to guide the search in a systematic and intelligent way by using flexible and adaptive memory structures and some intensification and exploration strategies (Glover 1986; Glover et al., 1993; Glover and Laguna, 1997; Hansen 1986). The main components of tabu search are: *short-term memory*, *long-term memory* and *intensification and diversification strategies*. Short-term memory is used to forbid revisiting solutions and then avoid cycling and being trapped in poor local optima. Long-term memory is used as a kind of learning process to generate intensification and diversification strategies. Long-term memory is used to collect information during the overall search process that permits the identification of common properties in good visited solutions and also to attempt to visit solutions with varying properties from those already visited. The implementation of both short-term and long-term memory is *based* on four principles: *recency*, *frequency*, *quality* and *influence*. While the recency principle is an indication of how recent it was that certain solutions were visited, the frequency principle is an indication of how often those solutions were visited. The quality principle refers to keeping information about visited solutions with good fitness values to identify good solution components and stimulate more intensive search in promising areas of the solution space. Finally, influence is used to identify those changes induced in the solutions structure that have proven to be more beneficial. Figure 3.8 shows the pseudocde for the tabu search metaheuristic.

Step 1. Generate initial current solution x.
Step 2. Initialize the *Tabu List*.
Step 3. While set of candidate solutions X' is not complete.
    Step 3.1. Generate candidate solution x' from current solution x using the *strategies for intensification and diversification*.
    Step 3.2. Add x' to X' only if x' (or associated attributes) is not tabu or if at least one *Aspiration Criterion* is satisfied.
Step 4. Select the best candidate solution x^* in X'.
Step 5. If fitness(x^*) > fitness(x) then $x = x^*$.

Step 6. Update *Tabu List*, *Aspiration Criteria* and *Intensification and Diversification Strategies*.
Step 7. If stopping condition met finish, otherwise go to Step 3.

Figure 3.8. Tabu search metaheuristic.

Short-term Memory

This component is usually implemented by maintaining a list that contains the most recently visited solutions. In most combinatorial optimisation problems, managing a list of visited solutions is not very efficient. Therefore, instead of the solution only some of its attributes (moves, components, etc.) are stored. This list is called the *tabu list* and the information stored there is used to forbid revisiting solutions for a certain number of iterations. The *tabu list size* defines how many recently visited solutions or their attributes are classified as tabu and the *tabu tenure* indicates for how long (usually measured in terms of the number of iterations) each element of the list remains tabu. Then, during the local search only those moves that are not tabu will be explored unless the tabu move satisfies the predefined *aspiration criteria*. These *aspiration criteria* are used because the attributes in the tabu list may also be shared by unvisited good quality solutions. A common *aspiration criterion* is better fitness, i.e. the tabu status of a move in the tabu list is overridden if the move produces a better solution.

Long-term Memory

The long-term memory component is implemented by keeping a history of the overall search process based on the four principles mentioned above. Then, by storing information about the *recency*, *frequency*, *quality* and *influence* of solutions, moves or other attributes, it is possible to tune the strategies that will attempt to guide the search in a more intelligent way.

Intensification and Diversification Strategies

An example of an intensification strategy is that after identifying components of good quality solutions and moves that have had the most influence towards these solutions, the search is intensified around certain areas of the solution space and using these beneficial moves. An example of a diversification strategy is that after

identifying moves that have been accepted more frequently, the search is directed towards other areas by forcing moves that have not been used so frequently.

Remarks

The variety of principles that are incorporated in tabu search and the flexibility in which they can be implemented are factors that have contributed to the successful application of this metaheuristic to a wide range of combinatorial optimisation problems (Reeves, 1995; Glover and Laguna, 1997). In fact, tabu search can be better conceptualised as a framework rather that a method. This is because each of its components can be designed specifically for the target application following the principles and suggested refinements that have emerged as a result of the experience from practitioners and researchers in various fields.

3.5.12. Genetic Algorithms (GA)

Genetic algorithms were in essence proposed by Holland in his book *Adaptation in Natural and Artificial Systems* (Holland, 1975). However, the ideas of using evolution and recombination for optimisation were proposed even earlier by Bremmermann (Bremmermann, 1962). A genetic algorithm is a population-based method that is based on the principles of natural evolution (e.g. Goldberg, 1989; Man et al., 1999; Michalewicz, 1999). The main idea in genetic algorithms is to generate a population of individuals and then, during a number of iterations (generations) to evolve this population by means of *self-adaptation* and *recombination*. Figure 3.9 shows the general framework of a genetic algorithm.

Step 1. Generate initial population.
Step 2. Evaluate population.
Step 3. *Select* individuals that will act as parents.
Step 4. Apply *Recombination* to create offspring.
Step 5. Apply *Mutation* to offspring.
Step 6. *Select* parents and offspring to form the new population for the next generation.
Step 7. If stopping condition met finish, otherwise go to Step 2.

Figure 3.9. The genetic algorithm framework.

Mutation and crossover are the two basic genetic operators used for implementing self-adaptation and recombination respectively. Crossover refers to the generation of one or more individuals (offspring) from the recombination of two or

more solutions in the current population (parents) and its purpose is the propagation of good solution components (genetic material) from parents to offspring. Mutation refers to small random variations of the solution and its purpose is to add diversity to the population. At each generation, some parents are selected and then recombined to generate the offspring. Some of the children may be mutated before adding them to the next generation. If *elitism* is implemented, some high quality individuals are selected to survive from one generation to the next one. The *selection mechanism* used to choose the parents aims to enforce the principle of survival of the fittest and therefore, acts as an intensification strategy. Recombination and mutation aim to encourage exploration and act as a diversification strategy. It is expected that a genetic algorithm will be capable of evolving the population and then converging towards solutions of high quality. Among the specific components that have to be carefully selected when designing effective genetic algorithms are the following: a) individual encoding, b) selection mechanisms, c) genetic operators, d) replacement scheme and d) constraint handling techniques.

Individual Encoding

An individual in genetic algorithms is usually a solution, a partial solution or a set of them. The representation of individuals in genetic algorithms is called the *chromosome*. Selecting an appropriate chromosome is an important issue because such representation should be suitable for the effective functioning of the genetic operators and perhaps the constraint-handling mechanism. Common representations for combinatorial problems are binary strings (including gray coded strings) and permutations of integer numbers but more complex structures are often designed to represent individuals for real world problems (Coley, 1999; Goldberg, 1989).

Selection Mechanism

Several mechanisms exist for selecting individuals that will act as parents (Coley, 1999). For example, a common method is fitness-proportional selection where the probability of individuals for being chosen is proportional to their fitness. Another common method is tournament selection where two or more individuals compete among themselves for the right to become parents. In rank-based selection the

individuals are assigned a reproductive probability that depends on the rank they are given based on some criteria.

Genetic Operators

Mutation and crossover can also be implemented in many ways (Chambers, 2001). For example, a common way of implementing mutation is to select one or more positions in the chromosome and then modify them with a given (usually low) probability. The single-point and multi-point crossover operators are among the most well known and frequently used. In these operators one or more points (respectively) are selected at random to split the chromosome of the parents into sections and then recombine these sections to create the offspring (Goldberg, 1989).

Replacement Scheme

Once the crossover and mutation operators have been applied it is necessary to decide which individuals from the last generation will be replaced by the new offspring to form the new population. A non-elitist strategy replaces all individuals in the current population while an elitist strategy maintains the best individuals so that their genetic material can be transferred to the next generations (Man et al., 1999).

Constraint Handling

In constrained problems, the application of recombination (crossover) and random variations (mutation) to individuals makes the creation of infeasible solutions very likely with genetic algorithms. Constraint handling techniques for genetic algorithms can be grouped into three categories (Michalewicz, 1999):

1. Allow the violation of constraints but penalise them.
2. Apply special repairing heuristics to correct infeasible solutions.
3. Use special individual representations to guarantee or increase the probability of generating only feasible solutions or use problem specific operators that preserve the feasibility of solutions.

Remarks

Genetic algorithms are regarded as methods that are suited for exploring large solution spaces. It can be said that genetic algorithms are a general technique that can produce acceptable results in relatively short time and there exist many ways to design the main components mentioned above (Goldberg, 1989; Coley, 1999). However, in order to obtain high quality results it is generally acknowledged that it is required to design good genetic operators and to perform fine parameters tuning (Bäck, 1996). These algorithms have been applied to a variety of applications including optimisation, design and creative systems (Goldberg, 1989; Davis, 1991; Chambers, 2001; Bentley and Corne, 2002).

3.5.13. Other Evolutionary Algorithms (EA)

Although there is no universally accepted definition of evolutionary algorithms, some classifications have been proposed, see for example (Calegari et al., 1999; Hertz and Klober, 2000). Here, we refer to evolutionary algorithms as methods that handle a population of solutions, iteratively evolve the population by applying phases of *self-adaptation* and *co-operation* and employ a *coded representation* of the solutions. The genetic algorithm described above is one of several types of evolutionary algorithms that exist. Some of the key evolutionary approaches are described below.

Evolutionary Strategies

While genetic algorithms emphasize recombination (high crossover probability) as the main search mechanism and usually use self-adaptation (low mutation probability) only as a supportive mechanism, evolutionary strategies emphasize both mechanisms as fundamental for searching. Another difference is that while genetic algorithms usually operate on the encoded representation of a solution (*genotype*), evolutionary strategies operate on the solution itself (*phenotype*) (Bäck, 1996; Bäck et al., 1997). The basic notation $(\mu + \lambda)$ES where μ is the number of parents and λ is the number of offspring, represents an evolutionary strategy that in each generation selects the best μ individuals from the $\mu + \lambda$ individuals (parents and offspring) in

total. The modified notation (μ,λ)ES indicates that λ offspring are generated from the μ parents but the best μ individuals are selected only from the λ offspring.

Scatter Search and Path Relinking

The scatter search and path relinking metaheuristic (Laguna, 2002) consists of two phases. In the first phase, one or more feasible solutions are generated which serve as seed solutions. Then, a reference set containing the best solutions found so far in terms of fitness and diversity is created as follows: trail solutions are generated using the seed or the trial solutions. Then, these trial solutions are improved by means of local search before using them to update the reference set. It may be that the trail solutions and their improved versions are infeasible. Then, it will be necessary to apply repairing heuristics to these infeasible solutions. Once the reference set is created, the algorithm enters the second phase where a subset of solutions is created by recombination of solutions in the reference set. The combination of solutions is based on generalized path constructions in the Euclidean (scatter search) or neighbourhood space (path relinking). These newly generated solutions are then improved and used to update the reference set. This process continues until the stopping criteria are satisfied.

Memetic Algorithms

The term memetic algorithms (MA) has been used to identify a broad class of hybrid metaheuristics: evolutionary algorithms that incorporate local search heuristics, specialised recombination/mutation operators and/or other "helpers" specifically designed to exploit the knowledge of the problem domain (e.g. Moscato 1989; Moscato, 1999). While genetic algorithms are inspired by the metaphor of *genes*, memetic algorithms are inspired by the metaphor of *memes*. A gene is the unit of genetic information that is propagated biologically between generations during the evolution process. A meme is the unit of conceptual information (knowledge, ideas, behaviour, customs, etc.) that is transmitted by imitation from one generation to the next one. Then by incorporating the available knowledge about the problem into an evolutionary algorithm, the working metaphor is that of evolving a population both biologically and culturally. Since the term memetic was introduced some time after researchers have started to study this kind of hybrids, it is common that names such

as genetic local search, hybrid genetic algorithms an others are used when referring to memetic algorithms (e.g. Reeves, 1996; Ishibuchi et al., 1997; Falkenauer, 1996; Burke et al., 2000; Jaszkiewicz, 2002).

Ant Colony Optimisation

The ant colony optimisation (ACO) metaheuristic (Dorigo et al., 1996) is inspired by the behaviour of ants when finding the shortest path between a food source and their nest. Ants deposit a substance called *pheromone* while exploring paths and also use the level of concentration of pheromone to decide which path to follow. Since the pheromone evaporates as time passes, the concentration is stronger in the shortest paths making them more attractive for other ants that also contribute to enhance the attractiveness of the path. An ant colony optimisation algorithm consists of a set of artificial ants that incrementally construct solutions by adding components to their solutions. There exist several variants of algorithms based on the ant colony optimisation framework. For more references see (Dorigo et al., 1996; Blum and Roli, 2001).

Particle Swarm Optimisation

A swarm of individuals exploring a large solution space can benefit from sharing the experiences gained during the search with the other individuals in the population. This social behaviour has inspired the development of the particle swarm optimisation algorithm (PSO) (e.g. Kennedy and Eberhart, 1999). In most versions of this metaheuristic, individuals are not selected to survive or die in each generation. Instead, all the individuals learn from the others and adapt themselves by trying to *imitate the behaviour of the fittest individuals*. However, selection can also be implemented to simulate the *social rejection* of those individuals that are not well adapted to the group performance.

Cultural Algorithms

Cultural algorithms have been developed inspired by the way in which *cultural evolution* is achieved in social systems (Reynolds, 1999). In the evolution of social systems and in particular human societies, culture is a vehicle for transmitting information at three levels: between generations, between populations and between

individuals in the same population. In a social system some individuals may have more experience and knowledge, which are of high value for the society. Then, these individuals are voted to have a deeper influence in the population beliefs and hence the cultural evolution of the society. More instances of these desirable individuals may be promoted while those individuals who are not so desirable may be eliminated. The culture or beliefs of the society is then adjusted and used to guide the evolution of the population in each generation.

3.5.14. Hybrid Metaheuristics

With the exception of memetic algorithms, the metaheuristics described above can be considered *pure* in the sense that they are not a combination of two or more approaches. When applying metaheuristics to solve an optimisation problem, one way to pursue success is to adapt the technique using knowledge from the problem domain. This adaptation can be achieved by modifying its components and/or tuning its parameters. Another approach that is commonly adopted is to combine two or more algorithms to develop a hybrid approach better suited for the given problem. Hybrid metaheuristics have proven to be successful in many optimisation problems and particularly in practical or real-world problems. It is not within the scope of this thesis to provide an extensive survey on hybrid metaheuristics. Instead, the reader is referred to some of the surveys and compilations of metaheuristics applications available in the literature (Glover and Kochenberger, 2003; Voss et al., 1999; Aarts and Lenstra, 1997; Osman and Kelly, 1996; Osman and Laporte, 1996; Rayward-Smith et al., 1996; Reeves, 1995).

The hybridisation of metaheuristics has been proposed at various levels and in different ways. For example, the components of one metaheuristic can be embedded into another (using tabu lists within a genetic algorithm) or one metaheuristic can be used as a component to enhance the performance of another (simulated annealing as the local search phase in variable neighbourhood search). The many ways in which metaheuristics can be combined makes it very difficult to describe or list all of them. Instead, it is perhaps more effective to differentiate between the designing principles used. In order to achieve this, it would be useful to have a nomenclature or framework that covers and permits the description of the majority of the hybrids

proposed in the literature. Some attempts towards this have been made although it seems that still no common scheme for classifying hybrid metaheuristics has been adopted among researchers.

For example, (Hertz and Klober, 2000) proposed a framework for describing a wide range of evolutionary algorithms including their hybrids with local search. Seven main features are identified and used to describe an evolutionary algorithm with their framework: individuals, evolution process, neighbourhood, information sources, infeasibility, intensification and diversification. The authors illustrate their framework by using it to describe various evolutionary algorithms including genetic algorithms, scatter search and ant systems. In their final remarks the authors note that *"a good description of the main features of evolutionary algorithms can help to understand the philosophy of the method and better analyse the reasons that explain the good performance of a particular evolutionary algorithm"*. A similar taxonomy called Table of Evolutionary Algorithms (TEA) was proposed by (Calegari et al., 1999) to compare the principles of various evolutionary algorithms also including some hybrids.

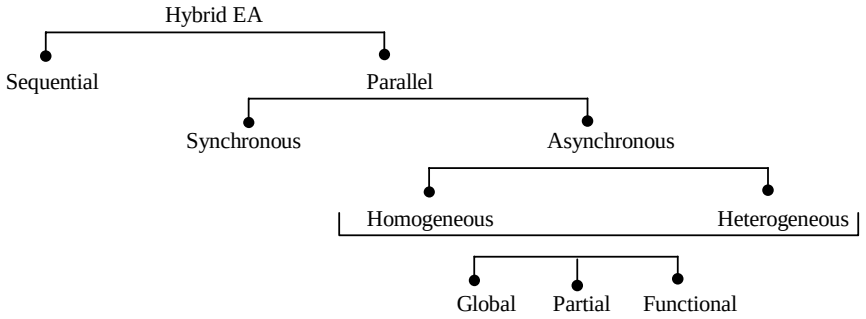


Figure 3.10. Hierarchy of hybrid evolutionary algorithms of (Preux and Talbi, 1999).

Another taxonomy of hybrid metaheuristics (focused also on evolutionary algorithms) is the one proposed by (Preux and Talbi, 1999). In their scheme the authors define the hierarchy shown in figure 3.10. *Sequential* hybrids refer to a set of algorithms that are applied one after another. For example, solutions initialised with a greedy heuristic are then evolved with a genetic algorithm and the final population improved by a local search method such as tabu search. The classification of *parallel* hybrids is more elaborate. The algorithms can be precisely synchronised

(*synchronous* hybrids) or cooperating with no specific coordination mechanism (*asynchronous* hybrids). In the *asynchronous* approach, *homogeneous* hybrids occur if all the cooperating algorithms are the same while in *heterogeneous* hybrids the cooperating algorithms are different. From a different perspective, the *parallel asynchronous* hybrids are divided into the following categories: *global*, *partial* and *functional*. A *global* implementation occurs when all the algorithms search the same solution space. In a *partial* approach the solution space is decomposed and each algorithm searches a part of it. In the *functional* hybrid, each of the algorithms solve a different problem.

In their paper, Preux and Talbi not only illustrate how some previously proposed hybrid algorithms can be classified using their taxonomy, but they also argue that the building blocks induced by their scheme can be combined in other ways to inspire other hybridisations (like the ones proposed in this thesis). They note that *"parallel asynchronous hybrid algorithms are strongly appealing for three main reasons: cooperation of individuals proves an efficient strategy on the long run, the stochasticity induced by the asynchronous cooperation has not been thoroughly explored as yet and the model ideally meets the requirements of implementation on parallel computers"*. A classification scheme similar to the one by Preux and Talbi but including many more references to hybrid metaheuristics was suggested by (Talbi, 2002).

The local search template of (Vaessens et al., 1998) is another classification scheme that attempts to capture most of the variants of local search algorithms. This template is based on three features: the *number of levels* (different searching strategies and neighbourhoods), the *population size* (point-based and population-based) and the *cluster size* (number of current solutions used to generate candidate solutions). Using their template, the authors describe algorithms such as tabu search, simulated annealing, threshold accepting, genetic algorithms, genetic local search and others. Note that they include genetic algorithms within their template although some researchers may argue that these algorithms are not local search methods. At the time of publication Vaessens et al. noted that *"some hybrids induced by their template had not been proposed or were not well known yet"*. In particular they suggest that multi-level local search algorithms deserve special attention since

existing techniques that fall into this classification have proven to be successful. These include genetic local search and other algorithms using more than one strategy or neighbourhood structure.

3.5.15. Evaluating the Performance of Metaheuristics

Metaheuristics are approximate algorithms and many of them will produce solutions of various qualities in different runs on the same problem instance. The stochastic nature of metaheuristic approaches is one of the main reasons for this (not unusual) behaviour. Obviously, if the optimal solutions for the problem are known, the performance of the metaheuristic technique can be assessed by comparing the solutions obtained by the metaheuristic technique to the optimal solutions. If the optimal solutions for the problem being solved are not known, assessing the quality of the solutions obtained using metaheuristics can be done in two ways: by referring to known upper and lower bounds or by referring to benchmark results (Reeves, 1995). Three aspects are of particular interest when assessing the performance of metaheuristic methods: *effectiveness*, *efficiency* and *robustness*. Effectiveness usually refers to the quality of the solutions produced by the method. Efficiency usually refers to how much computation time and memory the method uses. Robustness usually refers to how consistent the method is in producing the same or very similar results over many runs on the same problem instance.

Chapter 4. General Metaheuristic Approaches

4.1. Introduction

This chapter presents an initial investigation into applying metaheuristics to automate the solution of the academic space allocation problem described in chapter 2. The aim of this initial study is to assess the suitability of applying some well-known heuristic search methods in order to have an insight into the difficulty of solving the space allocation problem. Before starting an investigation into heuristic search, several decisions have to be made. The following aspects should be considered:

§ The selection of solution representation and associated data structures.

§ The definition of neighbourhood structures.

§ The implementation of efficient fitness evaluation routines.

§ The design of solution initialisation strategies.

§ The selection of search algorithms.

§ The tuning of algorithm parameters.

The rest of this chapter describes how these issues were addressed with respect to the problem studied here. The following four search techniques were selected to carry out this initial investigation:

§ Iterative improvement local search.

§ Simulated annealing algorithm.

§ Tabu search algorithm.

§ Genetic algorithm.

These methods were chosen because they have been applied to a great variety of problems, are considered robust in the general sense, many papers exist that provide guidelines for implementing them and various comparative studies between these and other techniques exist in the literature. See for example (Corne and Ross, 1995;

Hasan et al., 2000; Youssef et al., 2001). These four techniques can be considered as general search methods that need to be adapted and tuned for specific applications in order to obtain good results (Pirlot, 1996). From the work presented in this chapter, the material corresponding to the tabu search metaheuristic is included in the [Bur2003b] paper, while the material corresponding to the other three approaches is included in the [Bur2000] paper (see the appendix on page 199).

A considerable number of publications report on the improvement and tuning of the various components of the techniques above to make them more effective, efficient and robust. For example, in genetic algorithms different replacement policies have been proposed to manage the incorporation of the new generated individuals into the next generation and the preservation of the fittest individuals (elitism) from the current generation (Bäck et al., 1997). Several selection mechanisms, genetic operators and techniques for tuning the probabilities of these genetic operators have also been investigated (e.g. Tuson and Ross, 1998; Julstrom, 1995). As mentioned in section 3.5.10, with respect to simulated annealing, various cooling schedules including both deterministic and adaptive approaches have been studied in order to control the variation of the acceptance probability (Aarts and Korst, 1998; Ingber, 1996). In tabu search, there are different implementations of short-term and long-term memory structures or tabu lists including the use of learning techniques. The incorporation of preferred candidate lists, i.e. lists of promising moves or attributes of moves has also been explored. The aspiration criteria to be used when overriding the tabu status of a candidate move is another aspect that has received attention from the community and the use of different aspiration criteria during the search in an adaptive way has also been proposed (Glover and Laguna, 1997).

Various researchers have carried out experiments to compare the performance of the above metaheuristics on different problem domains. For example, the performance of simulated annealing, tabu search and a genetic algorithm are compared when solving an unconstrained Pseudo-Boolean function in (Hasan et al., 2000). In that paper the authors conclude that, after extensive experiments using well-tuned parameters for the three methods, the genetic algorithm performed the best although no reasons for this were identified. Another comparison between these

three metaheuristics was carried out in (Youseef et al., 2001) for the floorplanning of VLSI circuits. Four aspects were taken into account: quality of the best solution found, progress of the search, progress of the best fitness and number of solutions discovered at successive intervals of the fitness function. In that paper, the authors aimed to study the behaviour of the three algorithms instead of demonstrating the superiority of one of them. However, they reported that the best performance was exhibited by tabu search, closely followed by the genetic algorithm while simulated annealing stayed far behind. It was also noted that the genetic algorithm required the most effort with respect to the complexity of implementation and tuning of parameters.

4.2. Solution Representation and Data Structures

There are, in general, two types of solution representations for combinatorial problems: direct and indirect representations (also called explicit and implicit respectively). A direct representation encodes a solution while an indirect representation encodes the steps to construct a solution. For the academic space allocation problem investigated here, it was decided to represent an allocation or solution using the direct encoding described in section 2.4.2 where a solution x is a string in which each position represents an entity and the value in that position indicates the room to which the entity has been allocated. Other representations (e.g. each position in the string representing a room) were also considered, but having a string where each position represents an entity makes it easier to maintain the feasibility of solutions in terms of the condition that each entity must be allocated to exactly one room (eq. 2.8). In addition to this direct vector representation, it was also decided to design an appropriate data structure in which to keep all the information corresponding to the problem instance being solved (penalties, list of entities, list of rooms, etc.) and the details of each particular allocation or solution (penalties, used rooms, fitness, etc). All the information is organised using a data structure based on linked lists as described below.

The data for a problem instance is organised in three lists of objects corresponding to the following groups: entities, rooms and constraints.

§ Entities. This list holds the details of each entity to be allocated: name, associated weight level or priority, owner department, associated group, etc.

§ Rooms. This list holds the details of each room: id, capacity, building, floor, list of adjacent rooms, type of room, special features, etc.

§ Constraints. This list holds details of each constraint (hard and soft): label, description, associated penalty, associated entities/rooms, etc.

The lists described above hold information about the problem instance being solved but another data structure is required to keep details of an allocation or solution. The data structure used to represent a solution or allocation is based on the three objects described below.

§ EntityGene. This includes: fitness statistics for this entity, pointer to the respective entity in the global entities list, pointer to the RoomGene to which this entity is allocated, pointer to the next EntityGene that is allocated to the same room and pointer to the first ConstraintGene in the list of constraints affecting this entity.

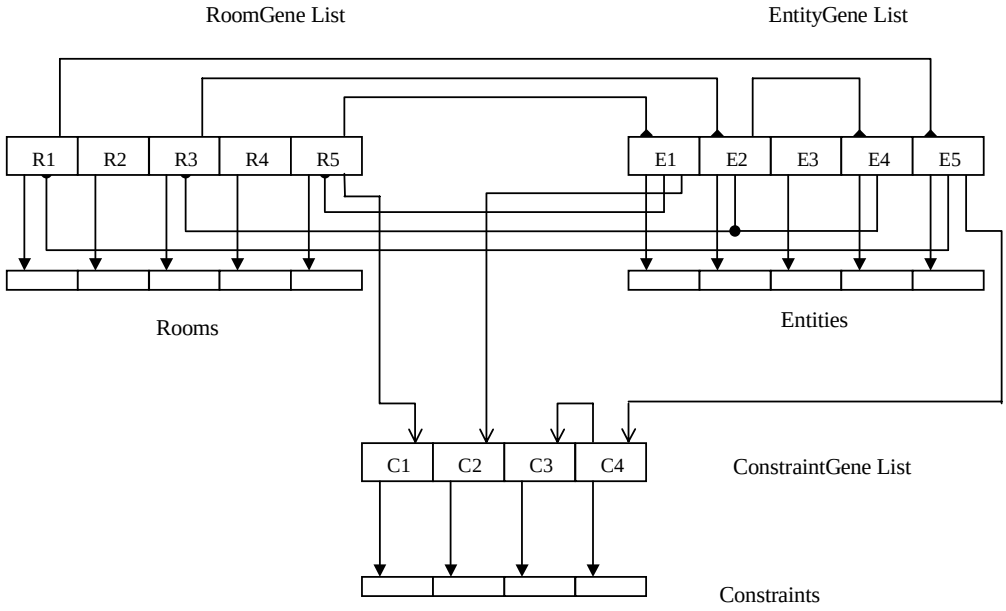
§ RoomGene. This includes: fitness statistics for this room, pointer to the corresponding room in the global rooms list, pointer to the first EntityGene in the list of entities allocated to this room and pointer to the first ConstraintGene in the list of constraints affecting this room.

§ ConstraintGene. This includes: pointer to the corresponding constraint in the global constraints list and pointer to the next ConstraintGene that is also assigned to the same entity or the same room.

Using the structures described above, the linked list model shown in figure 4.1 holds all the details of the problem instance and the allocation or solution. Note that the lists Entities, Rooms and Constraints are common to all solutions and are created only once. In this example, the problem consists of allocating the 5 entities (E1 to E5) to the 5 available rooms (R1 to R5) subject to 4 constraints (C1 to C4). Entity E1 is allocated to room R5, entity E2 and E4 are allocated to room R3 and entity E5 is allocated to room R1. Room R2 is empty and entity E3 is not allocated. Constraint

C1 applies to room R5, constraint C2 applies to entity E1 and constraints C3 and C4 apply to entity E5.

This data structure, based on linked lists, has the flexibility to easily change details about the problem instance and the solution. Also, the linked list model permits the easy implementation of local search moves that maintain the feasibility of the solution in terms of hard constraints (eq. 2.9) and the implementation of efficient solution evaluation routines as it is described later in this chapter. Other



researchers have also found that the use of linked lists is advantageous for representing combinatorial optimisation problems and their solutions (Randall and Abramson, 2001). So, an allocation or solution for the academic space allocation problem is represented in this thesis using the string of length *n* and stored using the data structure shown in figure 4.1. The combination of the string and the linked list structure helps to maintain the feasibility conditions in this problem.

Figure 4.1. Data structure used for the space allocation problem. The global lists Entities, Rooms and Constraints hold data corresponding to the problem instance being solved. The linked lists of genes hold details of a particular allocation or solution.

4.3. Neighbourhood Structures

Three neighbourhood structures or moves are defined to perform local search. These structures are given below together with their respective size in terms of the number of entities to allocate (n) and the number of available rooms (m).

- *Relocate* an entity to a different room. $|N_R| = n\,(\,m - 1\,)$.
- *Swap* the rooms between two entities. $|N_S| = n\,(\,n - 1\,) / 2$.
- *Interchange* the allocated entities between two rooms. $|N_I| = m\,(\,m - 1\,) / 2$.

In the above, n is the number of entities to allocate, m is the number of available rooms, and N_R, N_S and N_I refer to the relocate, swap and interchange neighbourhoods respectively. These neighbourhood structures are naturally associated to the problem studied in this thesis. They were selected so that targeted changes can be implemented in the existing allocation and the feasibility of solutions is fully or nearly maintained. Also, more elaborate moves or chains of moves can be generated from these three basic neighbourhood structures. A *feasible* move modifies the solution maintaining the feasibility conditions while a *suitable* move is a *feasible* move that also improves the solution quality. From the description of the problem given in chapter two, it can be noted that the improvement of solution quality can be achieved by reducing the amount of space misuse and/or by reducing the violation of soft constraints. Then, given the types of constraints in this problem, it is also possible to design specific moves or neighbourhood structures aimed to improve the solution quality. However, it was decided not to have such a high degree of specialisation so that the metaheuristic approaches proposed in this thesis could eventually be applicable to different problem instances (given the variety of soft constraints) and perhaps similar problem domains.

4.4. Fitness Evaluation Routines

After modifying a solution by means of the moves described above or using the genetic operators described later in this chapter, the fitness of the new allocation has to be calculated. Unfortunately, an exact evaluation of the new fitness cannot be carried out locally. The reason for this is that not only the entities and rooms involved in the move have to be taken into account but also the entities and rooms

affected indirectly by the modification. The level of satisfaction of other soft constraints not directly related to the implemented move may also be affected and then, an exact fitness evaluation would require consideration of all the soft constraints. Such an exact fitness evaluation routine is very time consuming and therefore, an approximate fitness evaluation routine, also known as delta evaluation (Corne et al., 1994), is also implemented. Such approximate evaluation takes into account the changes in space utilisation and the changes in the soft constraints satisfaction directly related to the entities and rooms involved in the selected move. But this approximate evaluation does not consider the potential changes produced in the soft constraints satisfaction related to other entities and rooms not involved in the move.

Consider the situation illustrated in figure 4.2 where the selected move is to *swap* the assigned rooms between entities E3 and E4. The approximate evaluation takes into account the changes in space utilisation in rooms R5 and R6 and the change in the satisfaction of constraint C2 but not the satisfaction of constraint C1, which is also affected. The purpose of implementing two fitness evaluation routines (approximate and exact) is to use each of them in the appropriate case so that the search can be performed more efficiently. The exact evaluation is used when an improved solution has been found, in order to update the solution fitness accurately. The approximate evaluation is used while exploring the neighbourhood of a solution, in order to carry out a quick assessment of the suitability of implementing a move.

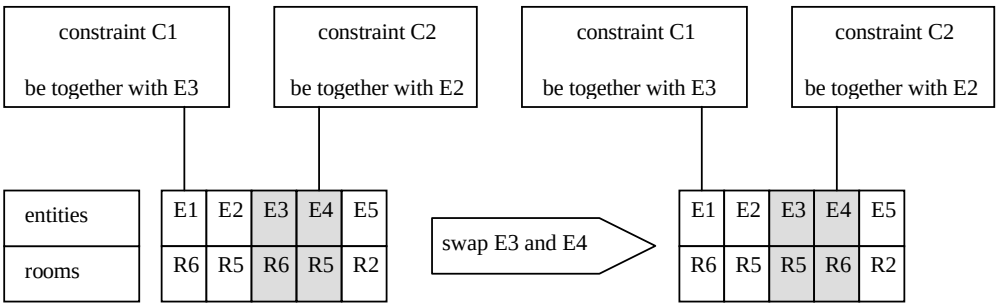


Figure 4.2. The approximate fitness evaluation routine. This procedure only considers the entities and rooms involved in the implemented move.

4.5. Constructive Heuristics and Neighbourhood Exploration

In this section, the heuristics employed to initialise an allocation and to perform neighbourhood search are described. Various degrees of greediness and exploration can be used when performing initialisation and neighbourhood search respectively. That is, there exist a number of strategies for constructing an initial solution that ranges from random selection to complete greedy heuristics including peckish methods, i.e. a greedy heuristic that occasionally makes mistakes (Corne et al., 1994). Similarly, while performing neighbourhood search, the selection of the next neighbouring solution can be done at random or after evaluating all the solutions in the neighbourhood.

Several researchers have noted that a trade-off needs to be established between the size of the neighbourhood and the efficiency and effectiveness of the exploration (Liu, 1999; Marett and Wright, 1996). Another aspect that must be considered is the connectedness of the solution space and the difficulty to explore it. The degree of intensification used to construct initial solutions and to explore the neighbourhood can have an effect on the performance of the metaheuristic used to perform the search (Dowsland, 1996; Corne et al., 1994).

4.5.1. Constructive Heuristics

The initialisation of an allocation is accomplished by iteratively allocating entities to rooms. Two selections have to be made: the next entity to allocate and then, the room to which the entity should be allocated. The constructive heuristic can vary from complete random selection of both the entity and the room to a greedy strategy that selects the best assignment. The following simple heuristics were implemented here:

AllocateBestAll. Selects the pair (unallocated entity,room) that produces the largest improvement in the solution fitness and allocates the entity to the room.

AllocateRnd-Rnd. Selects an unallocated entity and room at random and allocates the entity to the room.

AllocateRnd-BestRnd. Selects an unallocated entity at random, then explores a number of randomly selected rooms evaluating the suitability of each of them. Then, the chosen entity is allocated to the best of the subset of explored rooms.

AllocateWgt-BestRnd. The unallocated entities are sorted in decreasing order of their weight (required space). In each iteration, the unallocated entity with the largest weight is selected (breaking ties at random) and the room to allocate this entity is chosen using the same procedure as in the heuristic AllocateRnd-BestRnd.

AllocatePrty-BestRnd. The unallocated entities are sorted in decreasing order of their importance (for example managers, professors, technicians, etc.). In each iteration, the unallocated entity with the highest priority is selected (breaking ties at random) and the room to allocate this entity is chosen with the same procedure as in the heuristic AllocateRnd-BestRnd.

AllocateCsrt-BestRnd. This heuristic was designed specifically to allocate entities subject to hard constraints. If the selected unallocated entity is subject to hard constraints that limit the possible rooms to which this entity can be allocated (e.g. *Be located in* or *Be together with*), the feasible assignment that produces the best improvement in the allocation fitness is implemented. By using this heuristic to allocate entities subject to hard constraints, it is easier to guarantee the feasibility of the generated initial solutions.

Note that all the above heuristics select the entity to be allocated and then search for an adequate room using random or tournament selection. Heuristics selecting first the room to fill and then searching for adequate entities for the chosen room were also tried. However, the strategy of driving the initialisation by entity selection instead of by room selection produced better results overall. The main reason observed for this was that most of the constraints (soft and hard) are associated to entities rather than to rooms. This means that there is more flexibility when searching a room for a given entity and achieving satisfaction of constraints even with a small detriment in the room space utilisation efficiency.

4.5.2. Neighbourhood Structure Selection

The first step is to decide which type of neighbourhood structure (*relocate*, *swap* or *interchange*) to use and then to explore the chosen neighbourhood with a predetermined strategy. A heuristic was designed to select the type of neighbourhood structure or move before initiating the neighbourhood exploration to select the actual move. This heuristic is shown in figure 4.3.

Note that the strategy shown in fig. 4.3 considers the cases when all the entities are already allocated and also when there are unallocated entities. Although it was specified in chapter 2 that one of the feasibility conditions considered in this thesis is that all the entities in the problem instance must be allocated (eq. 2.8), the heuristic presented in figure 4.3 permits flexibility to consider other problem conditions (for example, when unallocated entities are allowed) and adaptability to various search strategies (considering or not infeasible solutions during the search). Moreover, this heuristic can be used for initialising solutions as well as for neighbourhood exploration. In this case, the heuristic tries to allocate as many entities as possible to produce a feasible solution by using the *allocate* move, but it also tries to avoid getting stuck by examining the *relocate*, *swap* and *interchange* neighbourhoods when no more allocate moves are possible.

Step 1. If all *n* entities are allocated then do
    Step 1.1. Select the move type at random: *relocate*, *swap* or *interchange*.
Step 2. If not all *n* entities are allocated then do
    Step 2.1. If the number of *attempts* ≥ *maximum attempts* permitted then do
        Step 2.1.1. If the previous selected move type was *allocate* then select a move between *relocate*, *swap* and *interchange* at random.
        Step 2.1.2. If the previous selected move type was not *allocate* then select the *allocate* move.
        Step 2.1.3. Set the number of *attempts* equal to zero.
    Step 2.2. If the number of *attempts* < *maximum attempts* permitted then do
        Step 2.2.1. If the previous selected move type was not *allocate* then select a move between *relocate*, *swap* and *interchange* at random.
Step 3. Explore the neighbourhood and return a move of the selected type.

Figure 4.3. Local search heuristic $\mathbf{H_{LS}}$ selects the type of move or neighbourhood structure and then explores the selected neighbourhood to find a move. The number of *attempts* refers to the number of previously consecutive failed (i.e. no accepted) moves. The value *maximum attempts* refers to the maximum number of failed *attempts* permitted.

To select the type of move, this heuristic takes into account the current state of the allocation and the history of success in applying each type of move. The type of move that is undertaken in each iteration, depends on the number of allocated entities and the number of prior failed attempts to find a feasible move of the selected type. That is, if all entities are allocated in the current solution, only the moves *relocate*, *swap* and *interchange* are explored. In the case that not all entities are allocated, a certain number of *maximum attempts* normally set to *n*/10 (decided by preliminary experimentation) is given to either of the three move types. For example, suppose that in the current solution there are still 5 unallocated entities from a total of 100 in

the allocation problem. Then, if after 20 failed attempts, none of these entities have been successfully allocated, the algorithm examines the feasibility of modifying the solution using the *relocate*, *swap* and *interchange* moves up to a maximum of 20 failed attempts. The number of failed modification attempts is set to zero when a move has been accepted by the driving metaheuristic (e.g. iterative improvement, simulated annealing or tabu search).

4.5.3. Neighbourhood Exploration

Once the neighbourhood structure (type of move) is chosen, exploring the selected neighbourhood consists of visiting one, some or all the solutions in the vicinity of the current solution and selecting one of them. As in solution initialisation, the neighbourhood exploration strategy can vary from random (visiting one neighbour at random) to exhaustive (visiting all neighbours). The heuristics that were implemented in this thesis to carry out the neighbourhood exploration are described below.

RelocateRnd-Rnd. Selects an allocated entity and a room at random and moves the entity from its previous room to the selected room.

RelocateRnd-BestRnd. Selects an allocated entity at random, then explores a number of randomly selected rooms evaluating the suitability of each of them to relocate the selected entity. Then, the chosen entity is allocated to the best of the subset of explored rooms.

RelocatePnty-BestRnd. The allocated entities are sorted in non-increasing order of their individual penalties (violation of soft constraints). In each iteration, the allocated entity with the highest penalty is selected and the room to relocate this entity is chosen with the same procedure as in the heuristic RelocateRnd-BestRnd.

SwapRnd-Rnd. Two entities allocated to different rooms (so that the swap move can take place) are selected at random. Then, the assigned rooms are swaped between these two entities.

SwapRnd-BestRnd. Selects one allocated entity at random, then explores a number of randomly selected entities allocated to a different room evaluating the suitability

of the swap. Then, the pair that produces the largest improvement in the solutions fitness is selected.

SwapPnty-BestRnd. The allocated entities are sorted in non-increasing order of their individual penalties (violation of soft constraints). In each iteration, the allocated entity with the highest penalty is selected and the entity to implement the swap is chosen with the same procedure as in the heuristic SwapRnd-BestRnd.

InterchangeRnd-Rnd. Two rooms are selected at random and the interchange move is conducted betweent these two rooms.

InterchangeRnd-BestRnd. Selects one non-empty room at random, then explores a number of randomly selected non-empty rooms evaluating the suitability of the interchange. Then, the pair of rooms that produces the largest improvement in the solution fitness is selected and the interchange move is conducted.

InterchangePnty-BestRnd. The non-empty rooms are sorted in non-increasing order of their individual penalties (space misuse and violation of soft constraints). In each iteration, the room with the highest penalty is selected and the room to implement the interchange is chosen with the same procedure as in the heuristic InterchangeRnd-BestRnd. Then, the interchange move is conducted using these two rooms.

The number of rooms explored when the BestRnd variant is used in the above moves was set to $n/3$ by preliminary experimentation. The various neighbourhood structures and heuristics described above permit the implementation of the heuristic $\mathbf{H_{LS}}$ (figure 4.3) in many different ways considering (or not) infeasible solutions and using different degrees of intensification. As mentioned above, the neighbourhood exploration is carried out faster because the approximate fitness evaluation routine is used. Another mechanism used in this thesis to speed up the neighbourhood search was to estimate the percentage of space that may be wasted or overused when implementing the selected move and to consider the move only if this percentage of misused space is within certain limits (±50% of the required space $w(j)$ for the j^{th} entity). If this space deviation is not calculated, the move is evaluated even if the rooms involved in the move are too big or too small for implementing the move. The selection of a suitable move with the above neighbourhood search heuristics does not

imply that the current solution will be improved. The moves are locally evaluated with the approximate fitness evaluation routine and the selected move is passed to the driving metaheuristic which will decide whether the move is accepted or not after the exact fitness of the new solution is calculated. That is, the $\mathbf{H_{LS}}$ local search heuristic samples the neighbourhood and returns a promising move to the driving metaheuristic. The following sections describe the metaheuristics implemented in this thesis.

4.6. Iterative Improvement Algorithm

The iterative improvement local search that was implemented in this thesis is shown in figure 4.4. By selecting different heuristics to explore the neighbourhood in the $\mathbf{H_{LS}}$ heuristic, this iterative improvement local search can be implemented with various degrees of neighbourhood exploration.

Various configurations were compared in order to select the best one. The experiments and results are described later in this chapter. Although this iterative improvement local search approach is quite simple, it is used in this thesis as a non-trivial algorithm against which to compare the performance of other more elaborate approaches.

Step 1. Generate initial current solution *x*.
Step 2. Generate candidate solution *x'* using the $\mathbf{H_{LS}}$ heuristic.
Step 3. If fitness(*x'*) > fitness(*x*) then *x* = *x'*.
Step 4. If stopping condition met finish, otherwise go to Step 2.

Figure 4.4. Iterative improvement local search uses the $\mathbf{H_{LS}}$ heuristic for neighbourhood sampling.

4.7. Simulated Annealing

Simulated annealing is a metaheuristic approach that has been applied to many optimisation problems. In particular, there are several papers in the literature reporting on the performance of this approach on scheduling related problems and the correlation between the observed performance of this algorithm and the neighbourhood exploration strategies, cooling schedules and acceptance probability functions used. For example, (Liu, 1999) studied the impact of different combinations of the neighbourhood structure size and cooling schedules on the

performance of simulated annealing for the flowshop scheduling problem. Liu observed that large neighbourhood sizes were more appropriate for fast annealing processes, small sizes did better for slow annealing processes while variable sizes gave the best results with respect to the effectiveness of the whole process.

Step 1. Generate initial current solution *x*.
Step 2. Set *temperature* = initial temperature.
Step 3. Generate candidate solution *x'* using $\mathbf{H_{LS}}$ heuristic.
Step 4. ΔF = fitness(*x'*) - fitness(*x*).
Step 5. If $\Delta F > 0$ then $x = x'$.
Step 6. If $\Delta F \leq 0$ then do
    Step 6.1. Calculate acceptance probability = exp (- ΔF/*temperature*).
    Step 6.2 If acceptance probability > random [0,1] then $x = x'$.
Step 7. Update *temperature* according to the cooling schedule.
Step 8. If stopping condition met finish, otherwise go to Step 3.

Figure 4.5. The simulated annealing approach uses the $\mathbf{H_{LS}}$ heuristic to explore the neighbourhood and the Boltzman-like distribution as the acceptance probability function.

The simulated annealing algorithm that was implemented in this thesis is described in figure 4.5. Several of the cooling schedules proposed in the literature were tested in the preliminary experiments carried out in this thesis. However, no significant difference was observed in the performance of the metaheuristic and therefore only the arithmetic and geometric cooling schedules are considered here due to their simplicity and good performance.

4.8. Tabu Search

As was the case with the simulated annealing algorithm, tabu search has also been applied to a great number of optimisation problems including many scheduling related problems. Many ways to implement the four main components of tabu search, *short-term memory*, *long-term memory* and *intensification* and *diversification strategies* have been proposed and compared in the literature (e.g. Glover and Laguna, 1997). The common strategy to implement *short-term memory* and *long-term memory* in tabu search is to store move attributes rather than to store visited solutions (which is not very efficient). One disadvantage of storing move attributes is that by forbidding certain moves, solutions that have not yet been visited may be avoided and some solutions may still be re-visited since they might be generated by a different sequence of moves. The heuristic $\mathbf{H_{LS}}$ uses three types of moves and

therefore the attributes that define the move that has been implemented may be different. For example, the attributes describing a *relocate* move can be the entity together with the previous and new assigned rooms. In the case of the *swap* move, the attributes can be the two entities being swapped together with their corresponding assigned rooms. The attributes for describing an *interchange* move can be the two rooms being involved in the move together with the corresponding allocated entities in each of the rooms. Of course, simplified attributes could be used to describe the moves and then all moves sharing the same attributes would be considered to be the same. Even the same set of attributes could be used to describe the three types of moves, such as the entity together with its previous room and its new assigned room. Various strategies of storing move attributes were tried, but managing lists of moves attributes is another aspect that contributed to slowing down the neighbourhood exploration. Therefore, instead of dealing with lists of moves, a mechanism that maintains pools of genes (parts of solutions) was used to implement the *short-term memory*, the *long-term memory* and the *intensification and diversification strategies* in the tabu search algorithm. Other researchers have also used matrices to store parts of solutions in order to implement short-term memory and long-term memory (Diaz and Fernandez, 2001; White and Xie, 2001).

4.8.1. Matrices of Tabu and Attractive Genes

Two matrices of size n x m are used and in both of them the cell (j,i) corresponds to the allocation of the j^{th} entity to the i^{th} room for j = 1,..,n and i = 1,..,m. The matrix M_T stores those pairs (entity,room) that will be considered as tabu for a number of iterations while the matrix M_A stores those pairs (entity,room) that will be considered attractive during the search. The tabu matrix M_T is updated each time a move suggested by the heuristic $\mathbf{H_{LS}}$ produces a detriment in the fitness of the current solution while the attractive matrix M_A is updated each time the move produces an improvement.

Updating a cell in M_T means setting its value to *current_iteration* + *tenure* so that a move involving the pair (entity,room) corresponding to that cell is set as tabu for *tenure* number of iterations. Some researchers have proposed the random variation of the *tenure* value within certain limits (Di Caspero and Schaerf, 2001; Schaerf,

1999b). Preliminary experiments carried out in this thesis for tuning the tabu search parameters, showed that a *tenure* value of around *n* and kept constant throughout all the iterations produced good results. Updating a cell M_A refers to incrementing the value of the cell in one unit, i.e. $M_A(j,i) = M_A(j,i) + 1$. In each type of move, the cells that are updated are the ones corresponding to the pairs (entity,room) after implementing the move. For example, if the 6^{th} entity is relocated from the 2^{nd} to the 4^{th} room, then the value in the cell $M_A(6,4)$ is incremented in one if the move produced a better solution but if the move generated an inferior solution the value in the cell $M_T(6,4)$ is set to the value *current_iteration* + *tenure*. Note that in a *swap* move two cells are updated while in an *interchange* move more cells can be updated.

The tabu matrix acts as the short-term memory component while the attractive matrix acts as the long-term memory component. Since both matrices store pairs (entity,room), this mechanism can be regarded as a way of memorising parts of allocations or genes that come from bad solutions (M_T) or good solutions (M_A).

4.8.2. Intensification and Diversification Strategies

Commonly, the intensification strategies incorporated in tabu search implementations use the short-term memory for exploring the neighbourhood of promising solutions. In the case of diversification, various strategies have been proposed. For example, one common way is to identify unvisited areas of the solution space with the aid of the memory components and then encourage the exploration of these areas. Some researchers have suggested to periodically change the weights in the fitness function during the search, a mechanism known as strategic oscillation (Costa, 1994; Alvarez-Valdes et al., 2000; Diaz and Fernandez, 2001; Schaerf, 1999b). Another way to diversify the search is to replace the current solution with the best solution so far after a number of non-improving iterations (Higgins, 2001). Tabu relaxation has also been proposed for diversification and it consists of re-initialising the tabu lists after a number of non-improving iterations (White and Xie, 2001).

In the tabu search algorithm implemented here, the matrices M_T and M_A are used to implement the strategies for intensifying and diversifying the search as described next. In the heuristic $\mathbf{H_{LS}}$, the neighbourhood exploration attempts to find a feasible move of the selected type (step 3 in figure 4.3). If a feasible move is found and its

attributes are considered tabu according to M_T, another move is sought unless the *aspiration criterion* is satisfied. The aspiration criterion used here is that the candidate solution generated by the move should be better (measured with the approximate fitness evaluation routine) than the current solution. If the neighbourhood exploration cannot find a feasible move, then a *relocate* move is heuristically created using the information stored in M_A. To do this, an entity j is selected at random and the highest value in the j^{th} row is identified in M_A (corresponding to the most attractive room i to allocate entity j). If the entity j is not already allocated to room i then the move proposed is to relocate the entity to that room (provided it is feasible). If this assignment already exists in the current solution, another entity is selected at random and the same process in carried out until a feasible *relocate* move is found. The tabu search implemented in this thesis is described in figure 4.6.

Step 1. Generate initial current solution x.
Step 2. Initialise the tabu and attractive matrices M_T and M_A.
Step 3. Explore a set of candidate solutions as follows. Generate a set of candidate solutions X' from current solution x using the modified $\mathbf{H_{LS}}$ heuristic. As described above, the modified version incorporates the *intensification and diversification strategies* using the *memory components* M_T and M_A. Select the best candidate solution x' from the set X' only if x' (associated move attributes) is not tabu or if the a*spiration criterion* is satisfied.
Step 5. If fitness(x') > fitness(x) then $x = x'$.
Step 6. Update tabu and attractive matrices M_T and M_A.
Step 7. If stopping condition met finish, otherwise go to Step 3.

Figure 4.6. The tabu search approach uses matrices to store parts of good and bad solutions in order to implement the short-term and long-term memory components.

4.9. Genetic Algorithm

A simple genetic algorithm was designed as described in figure 4.7 but several ways to implement its components were compared in order to obtain a relatively well-tuned version of this metaheuristic for the problem investigated here. The subsections below describe the various components of this genetic algorithm in more detail.

Step 1. Generate an initial current population.
Step 2. Evaluate the current population.
Step 3. Until the new population if completed do the following:
    Step 3.1. Select two individuals that will act as parents.

Step 3.2. If crossover probability ≥ random [0,1] then recombine the two selected parents to create two offspring.
Step 3.3. If crossover probability < random [0,1] then copy the two selected parents as the offspring.
Step 3.4. Apply the mutation operator with a given probability to the offspring.
Step 3.5. Copy the two offspring to the new population.
Step 6. Apply the elitist strategy consisting on replacing the worst individual in the new population with the best individual in the current population.
Step 7. Copy new population to the current population.
Step 8. If stopping condition met finish, otherwise go to Step 2.

Figure 4.7. The genetic algorithm approach implemented in this thesis.

4.9.1. Selection of Parents

Two variants were tried: fitness proportional selection (also called roulette-wheel selection) and tournament selection (Coley, 1999). Both selection methods produced comparable results and it was decided to use tournament selection with a tournament size between 2 and 5.

4.9.2. Genetic Operators

Four crossover operators were implemented and compared: *single-point*, *uniform*, *heuristic uniform* and *heuristic non-uniform*. Both single-point and uniform operators are well-known and their descriptions can be found in the literature (Coley, 1999). In the heuristic uniform operator, each pair of corresponding genes (entity,room) in both parents are compared in terms of their local fitness, i.e. the fitness of the corresponding entity. The gene with the highest fitness is copied to one of the offspring while the other gene is copied to the second offspring. In the heuristic non-uniform crossover operator, the first step is to copy both parents to the two offspring and then identify a number of genes (a parameter set usually to $n/5$) with the lowest fitness (the fitness of the entity) in each offspring. Then, for each of the offspring, these less fit genes are copied from the other offspring. That is, suppose that the 5^{th} entity is allocated to the 6^{th} room in the first offspring and allocated to the 9^{th} room in the second offspring. Assuming that this gene has been identified as one of the less fit (the penalty due to the violation of soft constraints is high for this pair) in the first individual, then the 5^{th} entity will be relocated from the 6^{th} to the 9^{th} room in the first offspring.

Both the single-point and uniform crossover operators performed reasonably well but very elaborate routines for repairing the allocations (satisfy the feasibility conditions imposed by the hard constraints) were required. On the other hand, the heuristic uniform and heuristic non-uniform operators produced solutions with not too many hard constraint violations (so are easily repaired) due to the fact that the fitness of each gene also reflects the degree of hard constraints violations. The heuristic non-uniform crossover operator was the one that produced the best results overall and it was selected for the final implementation of the genetic algorithm.

The mutation operator implemented here is a simple mechanism in which for each gene in the chromosome and with certain probability, a new room is selected at random and assigned to the corresponding entity. If the chosen room is not feasible for allocating the entity then the next gene is processed.

4.10. Experiments and Results

This section describes the experiments carried out in order to assess the performance of the metaheuristics described above and reports on the results obtained in these experiments. The goals were to tune the approaches to produce the best results possible with these methods and to identify those components of each metaheuristic that can be used to design a hybrid approach.

4.10.1. The Initialisation Heuristics

The first set of experiments compared the quality of solutions (in terms of fitness and diversity) generated by each of the initialisation heuristics described in section 4.5.1. The experiments consisted of generating 50 solutions with each of the heuristics for three of the test instances described in section 2.5. The results are reported in tables 4.1 to 4.3.

Initialisation Heuristics	**Total Penalty *F(x)***			***V(p)***
	maximum	**average**	**minimum**	
AllocateBestAll	1817.10	1817.10	1817.10	0.0
AllocateRnd-Rnd	9686.72	8892.05	8246.58	82.71
AllocateRnd-BestRnd	6294.67	4639.40	2966.13	60.26
AllocateWgt-BestRnd	8479.87	8269.83	8097.73	5.50
AllocatePrty-BestRnd	5583.31	4284.81	2789.26	36.38
AllocateCsrt-BestRnd	3713.60	2521.13	1717.52	44.43

Table 4.1. Performance of the initialisation heuristics on the test instance nott1.

Initialisation Heuristics	**Total Penalty *F(x)***			***V(p)***
	maximum	**average**	**minimum**	
AllocateBestAll	6070.47	6070.47	6070.47	0.0
AllocateRnd-Rnd	6637.38	6235.83	5854.21	71.76
AllocateRnd-BestRnd	6418.23	6100.41	5852.30	67.98
AllocateWgt-BestRnd	7335.64	6917.46	6581.47	20.44
AllocatePrty-BestRnd	5614.65	5221.08	4986.64	20.89
AllocateCsrt-BestRnd	5735.85	5453.26	5210.24	57.03

Table 4.2. Performance of the initialisation heuristics on the test instance trent1.

Of course, the greedy heuristic **AllocateBestAll** always generates the same solution which can be used as a reference to assess the quality of the solutions generated by the other heuristics. As may be expected, the heuristic **AllocateRnd-Rnd** produces sets of solutions with the highest diversity but also with low quality. The heuristic **AllocateWgt-BestRnd** generates solutions with low quality and also low diversity. This gives an indication that in this problem, guiding the initialisation of solutions by space utilisation appears to be inadequate perhaps due to the existence of additional constraints. Therefore, although the problem studied here can be seen as a variant of the knapsack problem, it would probably not be wise to use initialisation heuristics that have been proposed for knapsack problems to generate solutions for the academic space allocation problem since those heuristics are mainly based on the optimisation of space.

Initialisation Heuristics	**Total Penalty *F(x)***			***V(p)***
	maximum	**average**	**minimum**	
AllocateBestAll	1974.03	1974.03	1974.03	0.0
AllocateRnd-Rnd	7079.90	6449.81	5596.82	80.55
AllocateRnd-BestRnd	2264.54	1470.26	857.18	32.17
AllocateWgt-BestRnd	8112.36	8041.62	8112.36	4.38
AllocatePrty-BestRnd	2989.34	2054.71	1473.20	22.78
AllocateCsrt-BestRnd	2189.87	1395.62	931.04	31.62

Table 4.3. Performance of the initialisation heuristics on the test instance wolver1.

The heuristic **AllocatePrty-BestRnd** generates solutions with higher quality but the population diversity *V(p)* is still low. The heuristics **AllocateRnd-BestRnd** and **AllocateCsrt-BestRnd** appear to be the ones that provide the best compromise between quality and diversity in the set of generated solutions. Comparing these two heuristics, it can be observed that **AllocateRnd-BestRnd** produces solutions with

higher diversity and competitive fitness while **AllocateCsrt-BestRnd** obtains sets of solutions with lower diversity but better quality. Comparing the sets of solutions generated by each of the proposed initialisation heuristics permits the choice of the appropriate strategy to generate initial solutions when assessing the performance of the metaheuristics investigated in this thesis. In the rest of the experiments in this chapter, the heuristic **AllocateRnd-BestRnd** is used to initialise solutions. The reason for this selection is that with this strategy, solutions with a wider range of fitness values can help to better assess the performance of metaheuristics instead of using mostly very high quality initial solutions like those generated by the **AllocateCsrt-BestRnd** heuristic.

4.10.2. The Neighbourhood Exploration Heuristics

The next set of experiments was carried out to compare the various neighbourhood exploration strategies described in section 4.5.3. Different versions of the three metaheuristics that use neighbourhood search (iterative improvement, simulated annealing and tabu search) were implemented using the various neighbourhood exploration strategies as shown in table 4.4. The same neighbourhood exploration heuristic was used for the three moves in each variant, i.e. **Rnd-Rnd** in table 4.4 means that this strategy was used in the three moves *relocate*, *swap* and *interchange*. No combinations between different heuristics of the three moves were used in these experiments.

Metaheuristics	**Neighbourhood Exploration Heuristics**		
	Rnd-Rnd	**Rnd-BestRnd**	**Pnty-BestRnd**
Iterative Improvement	IIRnd-Rnd	IIRnd-BestRnd	IIPnty-BestRnd
Simulated Annealing	SARnd-Rnd	SARnd-BestRnd	SAPnty-BestRnd
Tabu Search	TSRnd-Rnd	TSRnd-BestRnd	TSPnty-BestRnd

Table 4.4. Variants of the three approaches using neighbourhood search.

The algorithm parameters used in these experiments were as described next (some of the parameters were tuned according to the size of the problem instance). For the simulated annealing algorithm the arithmetic cooling schedule was used with initial temperature = 1000, decrement step = 200 and decrement interval = $n/2$. For the tabu search algorithm, tenure = $2n$. The termination condition in all runs was a maximum of 5000 iterations. Each metaheuristic variant was tested 20 times with

each data set and the best results obtained by each variant are presented in tables 4.5 to 4.7. The aim here was to assess the effect of the different neighbourhood exploration heuristics on the performance of the three metaheuristics. Therefore, each table compares the performance between the variants of the same metaheuristic on the three problems. Each table shows the best solution, the execution time in seconds needed to complete the run and the iteration at which the best solution was obtained.

From the results presented in tables 4.5 to 4.7 it can be observed that the variants with the **Rnd-Rnd** and the **Rnd-BestRnd** heuristics are comparable in terms of the solution quality and execution time in most of the cases. On the other hand, the variants with the **Pnty-BestRnd** heuristic produce competitive results in terms of solution quality in some cases but the execution time is the longest in most of the cases too. Although there is not clear evidence that the **Rnd-BestRnd** strategy is the best, it appears from the results presented here that this heuristic for neighbourhood exploration benefits the performance of the three metaheuristics tested here since good quality solutions are obtained in short execution time and also the best solutions are found in the earliest iterations in most of the cases.

Problem Instance	Metric	IIRnd-Rnd	IIRnd-BestRnd	IIPnty-BestRnd
nott1	total penalty *F(x)*	2227.19	774.22	1733.17
	execution time (s)	39	31	57
	iteration best	4905	2924	4957
trent1	total penalty *F(x)*	2712.43	4440.12	5914.62
	execution time (s)	30	33	66
	iteration best	4939	2730	121
wolver1	total penalty *F(x)*	717.23	634.19	1164.02
	execution time (s)	25	20	37
	iteration best	4309	1465	234

Table 4.5. Results for the iterative improvement metaheuristic variants.

Problem Instance	Metric	SARnd-Rnd	SARnd-BestRnd	SAPnty-BestRnd
nott1	total penalty *F(x)*	4591.96	839.50	1371.96
	execution time (s)	34	33	83
	iteration best	87	4522	4543
trent1	total penalty *F(x)*	3558.76	4646.73	5144.22
	execution time (s)	28	29	76
	iteration best	4898	3490	2052
wolver1	total penalty *F(x)*	1391.87	1627.55	1110.38
	execution time (s)	16	20	28
	iteration best	54	1433	4123

Table 4.6. Results for the simulated annealing metaheuristic variants.

Problem Instance	Metric	**TSRnd-Rnd**	**TSRnd-BestRnd**	**TSPnty-BestRnd**
nott1	total penalty *F(x)*	2111.15	735.37	1626.76
	execution time (s)	54	37	46
	iteration best	4719	3424	4637
trent1	total penalty *F(x)*	3214.61	3903.82	3728.87
	execution time (s)	38	57	73
	iteration best	4938	4658	4833
wolver1	total penalty *F(x)*	1867.14	1431.77	1726.65
	execution time (s)	26	20	34
	iteration best	5000	635	4129

Table 4.7. Results for the tabu search metaheuristic variants.

4.10.3. Comparing the Four Metaheuristics

After selecting the initialisation and neighbourhood exploration heuristics as described in the previous sections, experiments were carried out to compare the performance of the four metaheuristics: iterative improvement, simulated annealing, tabu search and the genetic algorithm. For the first three algorithms, the parameters were set as described in the previous section and the **Rnd-BestRnd** variants were used in these experiments. The parameters for the genetic algorithm were set as follows: population size = 20, tournament size = 3, crossover probability = 80% and mutation probability = 5%. Each algorithm was executed 20 times with each problem instance and the best results in terms of solution quality are presented here. The termination condition for the single-solution algorithms (iterative improvement, simulated annealing and tabu search) was a maximum of 10000 iterations while for the genetic algorithm the maximum number of generations was set to 1000.

The results obtained are presented in table 4.8. For each of the test problems, a reference solution exists and its corresponding quality is also given in table 4.8. This reference solution is a manually constructed allocation that was obtained from the space officers in the universities that provided us with the test data sets. The quality of this reference solution is shown here for comparison with the quality of the solutions generated by the four algorithms tested here.

Problem Instance	**Metric**	**Iterative Improvement**	**Simulated Annealing**	**Tabu Search**	**Genetic Algorithm**
nott1	total penalty *F(x)*	754.45	849.62	772.28	2145.21
reference = 599.56	execution time (s)	60	58	57	221

	iteration best	4220	3700	4957	812
trent1 reference = 3873.51	total penalty *F(x)*	4341.77	4385.99	3924.03	7924.10
	execution time (s)	59	60	66	237
	iteration best	6840	9940	9480	901
wolver1 reference = 1141.01	total penalty *F(x)*	634.19	1217.81	634.19	1312.01
	execution time (s)	39	44	45	178
	iteration best	1020	1024	1300	620

Table 4.8. The best solutions obtained by the four approaches in the three test instances. The quality of a reference (manually constructed) solution is also shown for comparison.

4.10.4. Further Discussion of Results

From the results shown in table 4.8 it can be observed that the best results in terms of the solution quality and execution time are produced by the iterative improvement and the tabu search algorithms in the three test problems. The simulated annealing algorithm produces good results but which are slightly inferior to those obtained with iterative improvement and tabu search. Overall, the genetic algorithm seems to be the worst performer in terms of the solution quality and execution time. However, it is interesting to note that the genetic algorithm seems to be competitive in terms of solution quality for the problem wolver1 but is well outperformed in problems nott1 and trent1. That is, it seems that the genetic algorithm is capable of finding competitive solutions for the less constrained problem (wolver1). This gives an indication of the importance of the additional constraints that exist in the academic space allocation problem. Even when the genetic operators were reasonably tuned to deal with these constraints, still the recombination of solutions appears to be a difficult issue in this problem.

The time required for manually constructing an allocation varies from weeks to months according to space officers. It is observed that the metaheuristic approaches implemented here offer a promising alternative for automating the academic space allocation process in a shorter time. From the approaches investigated here, iterative improvement and tabu search appear to be the ones that are able to produce the best results but still do not match the quality of the manually constructed allocation when the problem is highly constrained (nott1 and trent1). Again, for the less constrained problem (wolver1) these two methods are able to produce solutions that are better than the reference solution measured with the fitness function used in this thesis. Constructing a completely new allocation is not a frequently needed task, but the

experts spend days, even months, on it, while the heuristic methods implemented here produce competitive initial solutions in seconds or minutes.

4.11. Summary and Final Remarks

This chapter presented an initial investigation into the application of metaheuristics for searching good solutions to the academic space allocation problem. A direct solution representation and associated data structures based on linked lists were used to store the information about the instance being solved and the allocation or solution. The use of these two components was beneficial in three ways. They permitted the implementation of faster solution evaluation routines. It was also easier to design the local search and genetic operators. In addition, these data structures can be easily updated if the features of the problem instance change, i.e. number of entities and rooms, constraints, etc.

Various initialisation heuristics were designed and compared in terms of the quality and diversity of the set of generated solutions. Having different strategies to generate initial allocations permits the production of sets of solutions with various quality and diversity values that help to better analyse the performance of the metaheuristics investigated in this thesis. Two of the initialisation heuristics generate sets of solutions with a good compromise between solution quality and population diversity. The heuristic finally chosen to generate initial solutions for the rest of the experiments was **AllocateRnd-BestRnd**, which selects one entity at random and then evaluates the suitabiliy of a set of rooms to allocate the entity and chooses the best of these rooms.

Heuristics for neighbourhood exploration with various degrees of intensification were compared with respect to their effect on the performance of the local search based metaheuristics (iterative improvement, simulated annealing and tabu search). The neighbourhhod exploration strategy that obtained the best results is the one in which the search of the move of the selected type (*relocate*, *swap* or *interchange*) is a trade-off between random and exhaustive search (**Rnd-BestRnd**).

This chapter proposed implementations of four well-known approaches: iterative improvement, simulated annealing, tabu search and a genetic algorithm and

compared their performance on some test instances of the space allocation problem. The iterative improvement algorithm is a simple non-trivial method used as a reference to compare the performance of other more elaborate approaches. In the simulated annealing method, several cooling schedules were compared. The best results were obtained with the arithmetic and geometric schedules with reheating. For the tabu search method, two matrices were proposed to implement the short-term and long-term memory components. These matrices maintain pools of genes (parts of solutions) that are used in the intensification and diversification strategies. For the genetic algorithm, several recombination operators were implemented. The best results were obtained with the heuristic non-uniform operator which was designed specifically for the space allocation problem in order to avoid the excessive violation of hard constraints. The simple mutation operator implemented in this thesis changes the assigned room (maintaining feasibility) of an entity selected at random.

Overall, after comparing the four metaheuristics, iterative imporvement and tabu search are the best performers, simulated annealing produces good results and the genetic algorithm is the worst performer mainly because of the highly constrained nature of the problem. Since no similar previous work has been reported in the literature, this investigation is a useful reference not only for the work presented in the following chapters but also for other researchers and practitioners interested in the application of metaheuristics to solve the space allocation problem in academic institutions.

Chapter 5. Hybrid Metaheuristic Approaches

5.1. Introduction

This chapter describes hybrid metaheuristics that were designed by combining components of the algorithms investigated in the previous chapter and adding some additional features described here. By preliminary experiments, it was possible to identify suitable sets of parameters that produced good performance on the approaches tested in chapter four and also to identify those components that seemed to contribute the most to their best performance. Two hybrid metaheuristics are proposed here. The first is a single-solution method that incorporates various features such as local search heuristics, adaptive cooling schedules, short-term memory, long-term memory and mutation operators. The second hybrid approach proposed here is a population-based variant of the first one. Both approaches make an automatic selection of the parameter settings according to the size of the problem instance and surpass the best performance of the metaheuristics implemented in the previous chapter.

In chapter two we noted that in the space allocation problem, like in many other optimisation problems, it is often desirable to obtain a set of high quality candidate solutions so that the decision-makers can select the best among them. However, it may also be the case that only one high quality solution is required. One particular feature of the hybrid population-based metaheuristic described later in this chapter is that by controlling a common cooling schedule for the whole population, it is possible to adapt the cooling schedule to favour either the generation of one high quality solution in short time or a set of high quality solutions at the expense of more computation time. The experiments and corresponding results presented in this chapter show that these hybrid approaches produce competitive solutions for the academic space allocation problem. The research work described in this chapter is included in the papers [Bur2001b], [Bur2001c] and [Bur2001d] (see the appendix on page 199).

5.2. A Single-Solution Hybrid Metaheuristic

Preliminary experiments revealed that some of the components of the metaheuristics tested in chapter four were beneficial when incorporated into a hybrid approach. For example, the aggressive exploration of the iterative improvement algorithm permitted us to construct solutions of reasonable quality in a relatively short computation time compared with the other techniques. Also, the oscillating effect in the acceptance function in simulated annealing and the memory structures in tabu search had a considerable contribution to the good performance of those algorithms. The mutation operator in the genetic algorithm was the operation that permitted us to better explore the solution space by adding diversity to the population without introducing too many problems of infeasibility. The pseudocode for the proposed single-solution hybrid metaheuristic is shown in figure 5.1.

Step 1. Generate an initial current solution x.
Step 2. Execute heuristic for parameters setting.
***** Heuristic Iterative Improvement Phase *****
Step 3. For iterations = 1 to *IterationsII* do
    Step 3.1. Generate a candidate solution x' using the modified $\mathbf{H_{LS}}$ heuristic that incorporates the *intensification and diversification strategies* using the memory components M_T and M_A.
    Step 3.2. If fitness(x') > fitness(x) then $x = x'$.
Step 4. Copy current solution to the best solution so far, i.e. $x^* = x$.
***** Simulated Annealing with Reheating Phase *****
Step 5. Set *AcceptanceProbability* = *InitialAcceptance*.
Step 6. Generate a candidate solution x' using the modified $\mathbf{H_{LS}}$ heuristic that incorporates the *intensification and diversification strategies* using the memory components M_T and M_A.
Step 7. If a feasible move was found then calculate ΔF = fitness(x') - fitness(x).
Step 8. If $\Delta F > 0$ then $x = x'$ and if fitness(x') - fitness(x^*) > 0 then update the best so far, $x^* = x'$.
Step 9. If $\Delta F \leq 0$ then if *AcceptanceProbability* > random [0,1] then $x = x'$.
Step 10. Update the *AcceptanceProbability* according to the geometric cooling schedule.
Step 11. If no feasible move was found then increment *FailedMoveAttempts*.
Step 12. If *FailedMoveAttempts* > *MaxFailedAttempts* implement the *Heavy Mutation Operator* to disturb the current solution x.
Step 13. If stopping condition satisfied finish, otherwise go to Step 6.

Figure 5.1. The single-solution hybrid metaheuristic incorporates elements from various methods.

The hybrid approach consists of the components listed below:

- **Heuristic Neighbourhood Search.** Selects the neighbourhood to be explored and in consequence the moves to try while attempting to improve the current solution.

- **Heuristic Iterative Improvement.** Initialises the solution and achieves a certain level of quality in the initial allocation.

- **Simulated Annealing with Reheating.** Improves the solution produced by the heuristic iterative improvement algorithm and avoids being trapped in poor local optima by exploring different areas of the solution space by using an oscillation strategy driven by the acceptance probability.

- **Heavy Mutation Operator.** Modifies the current solution by unallocating some entities from the current solution and encourages a better exploration of the solution space.

- **Heuristic Parameters Setting.** Selects the algorithm parameters according to the problem characteristics. This heuristic might not produce the optimal parameter values for each problem, but will find a good set of parameters in general.

5.2.1. The Hybrid Components

Heuristic Neighbourhood Search

The strategy used to explore the neighbourhood of the current solution in the hybrid approach was the $\mathbf{H_{LS}}$ heuristic of figure 4.3 extended with the incorporation of the tabu and attractive matrices described in chapter four, i.e. the neighbourhood exploration in the hybrid algorithms is done in the same way as in the tabu search algorithm of section 4.8.

Heuristic Iterative Improvement

After generating an initial solution, the iterative improvement algorithm described in chapter four is executed for *IterationsII*. The purpose of this component is to quickly improve the initial solution by using the heuristic neighbourhood search component. Given the improved solution (not necessarily local optima) produced by this component, a further exploration of the solution space is accomplished in the subsequent phases of the single-solution hybrid metaheuristic.

Simulated Annealing with Reheating

The simulated annealing phase takes the improved feasible current solution obtained from the previous phase and uses the heuristic neighbourhood search component to search the solution space and attempt to find a better solution. This simulated

annealing phase uses a cooling schedule that is simpler than the one used in the implementation of chapter 4. It is a simple geometric cooling schedule (see section 3.5.10) that sets the *AcceptanceProbability* parameter to the value of *InitialAcceptance* and decrements it after a number of iterations. When the *AcceptanceProbability* is below a certain minimum, the cooling schedule maintains this value while the search process attempts to find improvements in the best solution so far. If, after a number of iterations *ReheatInterval*, no improvement is achieved in the best solution so far, the parameter *AcceptanceProbability* is again set to *InitialAcceptance*, i.e. the process is reheated.

Heavy Mutation Operator

A mutation operator was designed to disrupt the current solution and explore other areas of the solution space after a number of failed attempts to find a feasible move. The disruption consists of removing from their assigned room, those allocated entities that contribute the most to the total penalty. This operation releases the space assigned to those entities so that new possibilities of allocating them can be explored. This heavy mutation operator works as follows. A maximum of *RemoveLimit* entities to be unallocated is determined according to the size of the problem instance. The allocated entities are sorted in decreasing order of their associated penalty, i.e. the violation degree of the soft constraints associated to each of them. Then, starting from the most penalised one, entities are unallocated up to the maximum *RemoveLimit*. Once the current allocation is disrupted in this way, the simulated annealing component will reallocate the unallocated entities because the neighbourhood exploration heuristic will select the *allocate* move until all entities are allocated again as described in section 4.5.2. The purpose of this heavy mutation operator is to modify the current allocation after the algorithm gets stuck but this modification is directed so that only bad parts of the solution (penalised entitities) are disturbed.

Heuristic Parameters Setting

This component selects the algorithm parameters according to problem instance being solved. The parameters for the simulated annealing component are set as follows. The maximum acceptance probability *InitialAcceptance* is set to a value

between 95% and 100%. The decrement factor α in the geometric cooling schedule is set to a value between 0.97 and 0.99. The number of iterations after which *AcceptanceProbability* is reduced is set to a value between 1 and 3. Once *AcceptanceProbability* temperature has been reduced to 0.001 or below (the process is cooled), it is reset to the value of *InitialAcceptance* if after *ReheatInterval* = $10 \cdot n$ iterations no further improvement has been achieved in the best solution so far. The number of iterations for the iterative improvement phase is set to *IterationsII* = $5 \cdot n$. The value for the maximum number of failed move attempts is set to *MaxFailedAttempts* = $n/10$.

5.3. On the Performance of the Single-Solution Hybrid

In this section the performance of the proposed hybrid approach is assessed and compared against the three single solution metaheuristics implemented in chapter 3: iterative improvement, simulated annealing and tabu search. The experiments carried out for this purpose are described next followed by a presentation and discussion of the results obtained. The genetic algorithm was not considered here because of the poor performance shown in section 4.10.3.

5.3.1. Experimental Settings

Three problem instances: nott1, trent1 and wolver1 were used for the experiments. For each of these test problems, 20 initial solutions were generated using the **AllocateRnd-BestRnd** heuristic described in section 4.5.1. Then, each algorithm was executed with each of these 20 initial solutions. Preliminary experiments were carried out to determine, for each algorithm, the execution time after which no further improvements on the best solution so far were observed. Then, the termination condition was set to an amount of execution time large enough to allow the four algorithms to achieve their best performance in each test problem. This execution time for problems nott1, trent1 and wolver1 was set to 300, 120 and 15 seconds respectively.

5.3.2. Results and Discussion

Table 5.1 below shows for each test instance, the quality of the reference solution and the results obtained in the experiments described above. Similarly to the results from the experiments carried out in the previous chapter, the iterative improvement and the tabu search algorithms produce very competitive results while the simulated annealing implementation exhibits comparable performance only in the nott1 instance. However, note that the hybrid metaheuristic outperforms the other three algorithms and it is also capable of finding better solutions than the reference allocations for the three test instances. It also appears that the performance of the hybrid metaheuristic is more robust than the other three algorithms with respect to the quality of the solutions produced in different runs as reflected by the values for the worst and average fitness.

The contribution of the space misuse and violation of soft constraints to the total penalty in the solutions obtained is presented in figure 5.2. This permits to have a closer look at the improvements achieved using the single-solution hybrid metaheuristic over the solutions produced with the other three algorithms and over the reference solution. Each bar in the graphs represents the average space misuse and the average soft constraint violation for each set of 20 solutions obtained by each algorithm in the experiments described above.

Problem Instance	**Total Penalty F*(x)***	**Iterative Improvement**	**Simulated Annealing**	**Tabu Search**	**Single-Solution Hybrid Metaheuristic**
nott1 reference = 599.56	worst	887.65	806.81	844.63	674.49
	average	716.79	703.14	698.77	592.24
	best	568.36	548.52	546.67	527.15
	execution time (s)	300	300	300	300
trent1 reference = 3873.51	worst	4531.50	4671.72	4302.54	3838.03
	average	4303.11	4435.04	3960.90	3676.36
	best	3968.48	4162.94	3572.19	3526.27
	execution time (s)	120	120	120	120
wolver1 reference = 1141.01	worst	920.20	1935.64	872.15	714.05
	average	716.70	1583.05	717.47	642.17
	best	634.19	1142.16	634.19	634.19
	execution time (s)	15	15	15	15

Table 5.1. Quality of the solutions obtained by the four single-solution approaches in the three tests problems. The quality of the reference (manually constructed) solution is shown for comparison.

It can be observed from figure 5.2 that, regarding space utilisation, it is apparent that all the solutions obtained with the four algorithms are comparable with the

reference solution. The difference between the performance of the single-solution hybrid metaheuristic and the other three approaches appears to be mainly in the satisfaction of soft constraints. That is, the single-solution hybrid metaheuristic obtains solutions of better quality because it is capable of finding solutions with less violation of soft constraints than the solutions produced by the other three algorithms. By comparing the solutions obtained with the single-solution hybrid metaheuristic to the reference allocations, it can be observed that in all problems the hybrid approach is capable of finding solutions with better space utilisation which contributed to produce solutions with lower total fitness overall. However, for the problem nott1, none of the algorithms is capable of finding better solutions than the reference one with respect to the satisfaction of soft constraints. This gives an indication of the particular difficulty of this problem instance for which none of the algorithms implemented so far has been capable of finding solutions that are better than the manually constructed allocation in terms of the satisfaction of soft constraints.

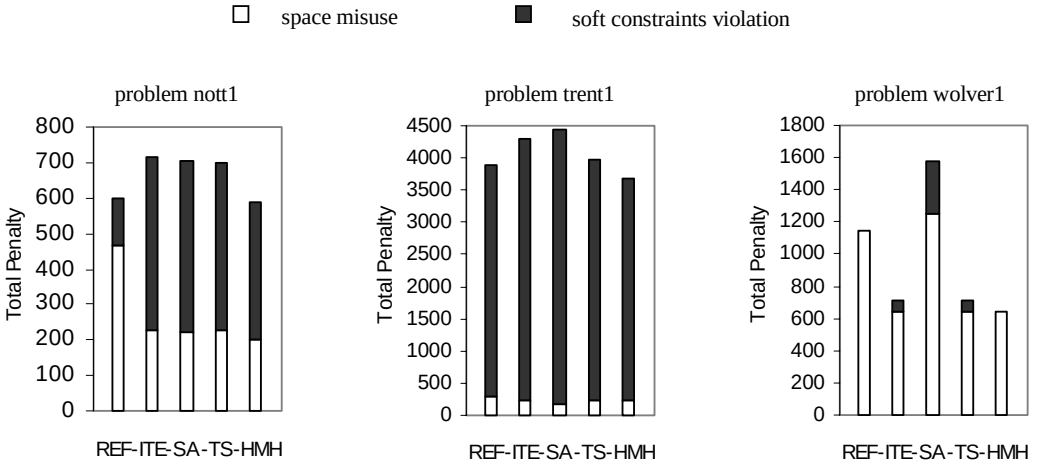


Figure 5.2. Contribution of space misuse and soft constraints violation to the total penalty. For each problem, the reference solution (REF) and average solutions obtained with the iterative improvement (ITE), simulated annealing (SA), tabu search (TS) and hybrid metaheuristic (HMH) are shown.

5.3.3. Further Comparison with Previous Results

The results presented and discussed above show that the single-solution hybrid metaheuristic produces the best solutions for the three test instances. The aim of the experiments described above was to assess the ability of each algorithm on finding good solutions after considerable execution time. This is the reason why the best

solutions obtained in these experiments using the three 'pure' metaheuristics (iterative improvement, simulated annealing and tabu search) are better than those produced by the same algorithms in chapter 4 (table 4.8, see section 4.10.3). Therefore, to further assess the performance of the hybrid approach proposed in this chapter, this hybrid algorithm was executed using the same initial solutions and termination condition (10000 iterations) of the experiments in section 4.10.3. The results are presented in table 5.2. In this table, the values for the three 'pure' methods are those given in table 4.8.

Problem Instance	Metric	Iterative Improvement	Simulated Annealing	Tabu Search	Single-Solution Hybrid Metaheuristic
nott1 reference = 599.56	total penalty *F(x)* execution time (s) iteration best	754.45 60 4220	849.62 58 3700	772.28 57 4957	715.42 54 812
trent1 reference = 3873.51	total penalty *F(x)* execution time (s) iteration best	4341.77 59 6840	4385.99 60 9940	3924.03 66 9480	3803.14 71 4193
wolver1 reference = 1141.01	total penalty *F(x)* execution time (s) iteration best	634.19 39 1020	1217.81 44 1024	634.19 45 1300	634.19 31 843

Table 5.2. Quality of the solutions obtained by the four single-solution approaches in the three tests problems when the number of iterations is set to 10000.

It is confirmed with the results presented in table 5.2 that even with a limited number of iterations, the single-solution hybrid metaheuristic obtains better solutions that the other three approaches. For the three test instances, the hybrid approach generates better solutions and the best solution is achieved in a shorter number of iterations. With respect to the total execution time required for the 10000 iterations, it can be observed that the time spent by the hybrid approach is very similar to the time spent by the other three algorithms. . From the results presented and discussed here, it is clear that the single-solution hybrid metaheuristic is the algorithm that produce the best solutions so far.

5.4. A Population-Based Hybrid Metaheuristic

In this section we show how the single-solution hybrid metaheuristic described in the previous section was extended towards a population-based approach. A population of individuals is initialised and then it is subjected to further improvement using the heuristic iterative improvement component described in the previous section. This

iterative improvement phase is executed for a number of *IterationsII* as in the single-solution approach. Then, the simulated annealing with reheating phase also described above is applied to each of the individuals in the improved population. The feature of the approach presented here is that instead of having a cooling schedule for each individual (this would be like a parallel implementation of the single-solution hybrid metaheuristic), a common cooling schedule is set for the whole population. The way in which the parameters for the common cooling schedule are set is described below. Since in the simulated annealing phase, inferior solutions may be accepted with some probability, two populations are maintained. The current population consists of the current solution for each individual in the population and the best population consists of the best solution found by each individual during the search process so that a set of best solutions can be presented at the end of the algorithm. The pseudocode for this population-based approach is presented in figure 5.3.

Step 1. Generate the initial current population.
Step 2. Execute heuristic for parameters setting.
***** Heuristic Iterative Improvement Phase *****
Step 3. For each individual x_i in the current population Do
    Step 3.1. Generate candidate solution x_i' from x_i using the modified $\mathbf{H_{LS}}$ heuristic that incorporates the *intensification and diversification strategies* using the memory components M_T and M_A.
    Step 3.2. If fitness(x_i') > fitness(x_i) then $x_i = x_i$'.
Step 4. If stopping condition (usually a maximum of *IterationsII* iterations) for the heuristic iterative improvement phase is met then go to Step 5, otherwise go to Step 3.
Step 5. Copy the current population to the best population, i.e. $x_i^* = x_i$ for $i = 1,...p$.
***** Simulated Annealing with Reheating Phase *****
Step 6. Set global *AcceptanceProbability = InitialAcceptance*.
Step 7. For each individual x_i in the current population Do
    Step 7.1. Generate candidate solution x_i' from x_i using the modified $\mathbf{H_{LS}}$ heuristic that incorporates the *intensification and diversification strategies* using the memory components M_T and M_A.
    Step 7.2. If a feasible move was found then calculate ΔF = fitness(x_i') - fitness(x_i) otherwise increment failed move attempts(i).
    Step 7.3. If $\Delta F > 0$ then Do
        Step 7.3.1. Update the current solution for the individual, $x_i = x_i$'.
        Step 7.3.2. If fitness(x_i') - fitness(x_i^*) > 0 then update the best solution for the individual, $x_i^* = x_i$'.
    Step 7.4. If $\Delta F \leq 0$ then if *AcceptanceProbability* > random [0,1] then $x_i = x_i$'.
    Step 7.5. If the *AcceptanceProbability* equals zero (process is cooled) and *FailedMoveAttempts*(i) > *MaxFailedAttempts* then implement the *Heavy Mutation Operator* to disturb the current solution x_i.
***** Common Cooling Schedule Update *****
Step 8. Update *AcceptanceProbability* according to common cooling schedule.
Step 9. If stopping condition satisfied finish, otherwise go to Step 7.

Figure 5.3. The population-based hybrid metaheuristic uses a common cooling schedule to control the simulated annealing phase for the whole population of individuals.

To summarise, the population-based hybrid metaheuristic incorporates a population of individuals that cooperate during the search by using the common neighbourhood search strategy and memory structures. Also, the annealing process for the whole population is driven by a common cooling schedule in which the control of the acceptance probability is distributed over all individuals in the population. The various features of the proposed algorithm are further described in the following subsections.

5.4.1. The Shared Memory Structures

Instead of maintaining a single solution, a set of individuals are evolved in the extended algorithm. Therefore, in order to take advantage of the collective searching process, the memory structures containing tabu and attractive genes (matrixes M_T *and* M_A) are shared among all individuals in the population. In this way, the heuristic $\mathbf{H_{LS}}$ for neighbourhood exploration can be seen as a cooperative mechanism by which the good and bad parts of solutions encountered by the various members of the population are stored so that a more effective search can be performed collectively.

Then, the neighbourhood search in the population-based hybrid metaheuristic is performed as in the previous chapter by using the heuristic $\mathbf{H_{LS}}$ with the same memory structures and diversification and intensification mechanisms. Referring to the pseudocode in figure 5.3, x_i represents the current solution for the i^{th} individual in the population, x_i* represents the best solution found so far by the i^{th} individual and the shared memory structures are updated accordingly each time a candidate solution x_i' is generated for the i^{th} individual. Experiments were carried out to assess the contribution of the shared memory structures to the performance of the extended algorithm and the results obtained are presented later in this chapter.

5.4.2. The Common Cooling Schedule

The other feature which is characteristic of the population-based hybrid metaheuristic is that a common cooling schedule is used to control the evolution of the whole population. This strategy of using a common cooling schedule for the whole population makes it possible to have a set of co-operating individuals that react differently to the annealing process. The way in which the common annealing

process is controlled permits the algorithm to find one high quality solution in a short computation time or a set of good solutions provided more computation time is available. This section describes how the common annealing schedule (step 8 in figure 5.3) operates upon the population.

The *AcceptanceProbability* is decreased (process is cooled) after *IntervalCounter* iterations (complete executions of step 7 in figure 5.3) as *AcceptanceProbability* = *AcceptanceProbability*$\cdot\alpha$ where α takes values between 0.97 and 0.99 as in the single-solution hybrid metaheuristic. A counter $ReheatCounter_i$ is maintained for each individual i in the population and it is incremented in one each time the candidate solution x_i' does not improve upon the current solution x_i and the *AcceptanceProbability* equals zero. There is a global counter *GlobalReheatCounter* that is set to the highest $ReheatCounter_i$ of all individuals each time the step 7.5 in figure 5.3 is processed. This means that as soon as one of the individuals cannot be improved for *ReheatInterval* iterations, the common *AcceptanceProbability* is raised again. The effect of this common annealing strategy is that while one (maybe more) individual is stuck during the search, the others may not be yet. Then, by switching to the random phase of the simulated annealing algorithm (*AcceptanceProbability* above 0.001) the exploration of the search space can continue. It may appear that waiting for all the individuals to achieve the most improvement possible before raising the global acceptance probability makes more sense. However, our experiments showed that when this was done, few individuals in the population were likely to achieve further improvement after getting stuck in a possible local optima. On the other hand, using the strategy proposed above permitted more individuals to explore other areas of the search space and more improvements were obtained which allowed the algorithm to produce better results overall.

5.5. On the Performance of the Population-Based Hybrid

5.5.1. Experiments and Results

Since in the previous section it was observed that the single-solution hybrid metaheuristic obatined the best results among all the single-solution algorithms, the first set of experiments in this section seeks to compare the performance of the

single-solution hybrid metaheuristic and the population-based variant. Experiments were also carried out to assess the contribution of the shared memory structures and the heavy mutation operator on the performance of the population-based algorithm. For each test problem, the same initial population of $p = 20$ individuals used for the single-solution variant was also taken as the initial population for the population-based hybrid metaheuristic. The overall computing time assigned to each algorithm was the same. That is, while each of the 20 runs (one run for each individual) of the single-solution hybrid metaheuristic was given a certain execution time t_{run} according to the test instance, the execution time for one run of the population-based approach was set to $20 \cdot t_{run}$. Another run of the population-based approach without the shared memory structures and without the heavy mutation operator but using the same initial population was executed. This experiment was repeated 10 times. That is, 200 solutions were produced in total with each of the theree algorithms compared.

Problem Instance	**Total Penalty** ***F(x)***	**Single-solution hybrid metaheuristic**	**Population-based hybrid metaheuristic**	**Population-based hybrid metaheuristic'**
nott1 reference = 599.56	best-average	576.15	619.02	681.69
	average	592.24	633.10	668.61
	minimum	527.15	575.51	641.38
	std. dev.	47.21	44.67	67.33
	diversity V(p)	32.85	61.96	62.58
	execution time (s)	300	6000	6000
trent1 reference = 3873.51	best-average	3614.85	3787.56	4279.78
	average	3676.36	3817.34	4319.19
	minimum	3526.27	3669.97	4238.67
	std. dev.	120.54	115.90	69.88
	diversity V(p)	30.72	80.65	80.94
	execution time (s)	120	2400	2400
wolver1 reference = 1141.01	best-average	639.94	664.12	681.63
	average	642.17	677.85	690.73
	minimum	634.19	634.25	634.19
	std. dev.	61.23	53.16	74.02
	diversity V(p)	28.41	45.31	44.63
	execution time (s)	15	300	300

Table 5.3. Quality and diversity of the final population obtained by the single-solution hybrid metaheuristic and the population-based variant on the three tests problems. Population-based hybrid metaheuristic' refers to the modified algorithm when the shared memory structures and the mutation operator are not implemented. The quality of the reference solution is also shown for comparison.

Table 5.3 shows the results of these experiments. For each algorithm and each test instance, this table reports the following: the minimum penalty (the best of the 200 obtained solutions), average penalty (average of all 200 solutions), best-average penalty (the best value selected from the averages of the 10 repetitions), the standard

deviation (measured for all 200 solutions), the diversity of solutions (for all 200 obtained solutions) measured as described in section 2.4.3 and the execution time in seconds. The values in the third column are the results obtained by the single-solution hybrid metaheuristic apoproach in section 5.3 (see table 5.1). The fourth column shows the results obtained by the population-based hybrid metaheuristic (complete version) while the last column shows the results obtained by this approach when no shared memory structures are used during the neighbourhood search and no mutation operator was implemented.

It can be observed that the population-based algorithm (the complete version) produces solutions that are very competitive with those obtained by the single-solution approach for the three test problems. In particular, note that the best solutions found by both algorithms are of similar quality. It appears that in terms of the quality of solutions, the results produced by the single-solution approach are better that those obtained with the population-based variant. That is, the average and best-average values obtained with the single-solution method are better than those produced with the population-based variant in the three test cases and the standard deviations are very similar. However, an interesting observation can be made by looking at the results obtained with respect to the diversity of solutions. It is clear that the population-based algorithm produces more diverse sets of solutions for the three test instances. In other words, although the sets of solutions obtained with the single-solution approach seem to be of better quality, the diversity values obtained (around 30%) suggest that all the 20 solutions are in fact very similar in structure. On the other hand, the population-based variant produces sets of solutions of slightly lower quality but which are more diverse in structure. As discussed above, this can be particularly important in some scenarios where a set of solutions that actually represent very different allocations are required so that the decision-makers can choose the most appropriate. These results on the diversity of solutions motivated a further investigation of this aspect in the next chapter. The interest on this arises from the fact that obtaining a set of diverse solutions is an important goal in areas such as multicriteria decision-making and multiobjective optimisation.

From the results presented and discussed above, it appears that the population-based variants achieve solutions that are not only competitive with those produced by

the single-solution method in terms of the solution quality, but also the diversity of the population is clearly higher. It is also noted that when the shared memory structures and mutation operator are eliminated from the population-based approach, the performance of this algorithm is worsened as reflected by the results shown in the last column of table 5.3, although the diversity of the obtained populations is still high.

5.5.2. Variants of the Population-Based Hybrid

So far, the single-solution hybrid metaheuristic has produced the best solutions in two of the three test instances. The population-based approach generated solutions of slightly less quality. The aim of this section is to further investigate the performance of this population-based approach and present a variant of it that seems to outperform the best results produced by the single-solution hybrid metaheuristic. In the previous section, the termination criterion for the experiments was a fixed computation time. An insight into the behaviour of the population-based algorithm is observed when the termination criterion is a maximum number of iterations without improvement (idle iterations) on the best solution achieved by each individual. In order to assess the effect of the strategy selected to control the evolution of the population in the population-based approach, more experiments were carried out using a maximum number of iterations without improvement over the best solutions so far as the termination criterion in the iterative improvement and the simulated annealing phases (steps 4 and 9 respectively in figure 5.3). This permits us to produce a set of solutions of uniform quality or one high quality solution with the rest of the population being considerably less fit. Suppose that the termination condition is a number of iterations without improvement upon the best solution, i.e. for the i^{th} individual, the counter $NoImprovesCounter_i$ is incremented each time the candidate solution x_i' does not improve upon the best solution x_i*. Obviously, some individuals would reach this condition before others. If the algorithm is stopped after the first individual reaches this condition, one high quality solution is obtained after a relatively short computation time. But if the algorithm is stopped after all individuals have reached the above condition, a set of solutions of uniform high quality will be obtained at the expense of more computation time.

Two versions of the population-based hybrid metaheuristic were implemented: the population-based hybrid metaheuristic-single and the population-based hybrid metaheuristic-multiple (referred to as PMHS and PMHM respectively in the results presented below). The termination condition for the iterative improvement and simulated annealing phases was set to $5 \cdot n$ and $2 \cdot ReheatInterval$ (i.e. $20 \cdot n$) iterations without improvement respectively. In the PMHS approach, these two phases are terminated when the first individual in the population reaches the corresponding termination condition. In the PMHM approach, these phases are terminated until all individuals in the population reach the termination condition. The single-solution approach of section 5.2 was also implemented using $2 \cdot ReheatInterval$ idle iterations as the termination condition (step 13 in figure 5.1). As before, 20 individuals were generated and the same initial population was used for each of the three algorithms. Ten repetitions of the experiment were executed for each algorithm and each test instance. The results obtained in these experiments are presented in table 5.4.

Problem instance	**Total Penalty** ***F(x)***	**Population-based hybrid metaheuristic single strategy PMHS**	**Population-based hybrid metaheuristic multiple strategy PMHM**	**Single-solution hybrid metaheuristic**
nott1 reference = 599.56	best-average	1001.76	780.90	835.89
	average	825.18	698.75	780.27
	minimum	664.19	619.21	647.61
	std. dev.	102.26	50.67	89.19
	diversity V(p)	67.44	61.25	32.77
	time (s)	526	2150	620
trent1 reference = 3873.51	best-average	4166.59	3892.93	4260.69
	average	3937.63	3789.43	4056.22
	minimum	3711.75	3580.10	3909.87
	std. dev.	155.66	85.36	112.75
	diversity V(p)	82.56	80.79	39.51
	time (s)	890	2220	720
wolver1 reference = 1141.01	best-average	834.59	905.27	638.09
	average	725.71	735.71	634.58
	minimum	637.22	638.36	634.19
	std. dev.	65.30	88.11	1.20
	diversity V(p)	47.82	46.95	41.73
	time (s)	225	300	210

Table 5.4. Solutions obtained by the single-solution hybrid metaheuristic over 20 runs and the population-based variants with a population of 20 individuals when a number of idle iterations is used as termination criterion.

Several observations can be made from the results summarised in table 5.4. Both population-based variants seem capable of finding solutions of higher quality than those obtained with the single-solution approach with the exception on the wolver1

test instance were all 20 solutions are of high quality as reflected by the low standard deviation value. It is clear that the population of solutions produced by the PMHM algorithm are the best for the trent1 and nott1 test problems. For these instances, this variant obtains a population of high quality solutions while the PMHS approach obtains populations in which an outstanding high quality solution can be identified with the rest of the population being noticeably less fit, which is also reflected by the values of the standard deviation. It is observed that the single-solution variant is capable of producing high quality solutions for the three test instances but the variation between the results over the runs is also considerable. With respect to the computation time spent in these runs, as was expected, the PMHS variant finds a good quality solution quickly while the PMHM variant requires more execution time to achieve a set of high quality solutions. The execution time required by the single-solution hybrid metaheuristic is the lowest in the wolver1 and trent1 test problems but not in the nott1 instance where the best computation time is that of the PMHS approach. An additional observation is that as before, the diversity of the populations produced by the single-solution approach is the lowest while both population-based variants produce sets of solutions that are very different in their structure. This aspect is further investigated in chapter 6 where a multiobjective approach is adopted.

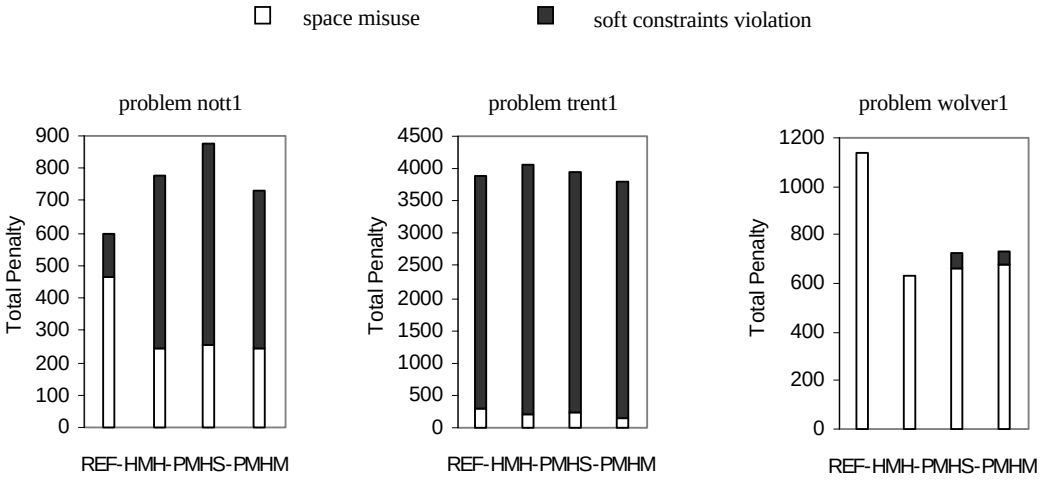


Figure 5.4. Contribution of the space misuse and soft constraints violation to the total penalty. For each test instance, the reference allocation (REF) and average solutions obtained with the single-solution iterative hybrid metaheuristic (HMH), the population-based hybrid metaheuristic single (PMHS) and the population-based hybrid metaheuristic multiple (PMHM) varianst are presented.

Figure 5.4 shows the contribution of space misuse and violation of soft constraints to the total penalty with respect to the average quality in the populations produced by each of the algorithms compared in this section. As before, the reference solution for each test problem is also shown for comparison. In the wolver1 instance, the single-solution approach finds solutions that are the best even than the reference solution and with no soft constraint violations. For the trent1 problem, the three algortihms are comparable. The PMHM variant and the single-solution approaches obtain solutions with slightly better space utilisation than in the reference solution. Finally, it is also observed that for the nott1 instance, none of the algorithms match the manually constructed solution with respect to the satisfaction of soft constraints, although all the solutions found are better than this reference solution on the space utilisation. Then, it is confirmed that the nott1 test instance seems to be particularly difficult to solve due to the high number of constraints that should be satisfied in this problem.

5.6. Summary and Final Remarks

In this chapter, competitive hybrid metaheuristic approaches for the space allocation problem were described and tested on some test instances. Van Veldhuizen and Lamont expressed that *"the selection of an appropriate solution technique must follow after a detailed examination of the problem to solve has been accomplished to integrate both problem and algorithm domains"* (Van Veldhuizen and Lamont, 2000). The approaches presented here were designed by a combination of the best features of several algorithms and a certain amount of knowledge about the problem domain. As a result, improved solutions have been produced with these hybrid algorithms over those generated with the 'pure' approaches investigated in chapter 4.

The single-solution approach described in section 5.2 is a hybrid that incorporates elements from the various techniques investigated in chapter 3: iterative improvement, simulated annealing, tabu search and genetic algorithms. The hybrid algorithm clearly outperforms the other techniques in the experiments carried out in this thesis. In the population-based approach described in section 5.4, the combination of adaptive cooling schedules in simulated annealing, population-based techniques and shared memory structures is proposed as an effective technique to

tackle the space allocation problem. The population-based technique produces very competitive results when compared to the single-solution hybrid but still the latter obtains the best solutions. However, with respect to the population diversity, the population-based variant produces much better results than those obtained with the single-solution approach. In addition, it was shown that when the shared memory structures and the mutation operator are not present in the population-based algorithm, the performance of this technique deteriorates considerably.

The population-based metaheuristic was modified in order to produce one single high quality solution in a short amount of time (PMHS algorithm) or a population of high-quality allocations provided more computation time is available (PMHM algorithm). The two variants of the population-based technique and the single-solution hybrid were again compared in section 5.5.2. The advantage of having a population of solutions is evident when the cooling schedule is controlled over a maximum number of iterations with no improvement (idle iterations). Under this condition, the PMHM algorithm was capable of producing sets of solutions with better quality and which are more diverse than those obtained with the single-solution approach for two of the test instances. The techniques proposed in this chapter seek to combine the best features from the metaheuristics studied in chapter 4 so that better results can be obtained for the problem studied in this thesis. If a diverse set of high quality solutions is required, then the population-based approaches are more appropriate but if the required non-similarity between allocations is low, then the single-solution hybrid metaheuristic is the most appropriate approach.

As with other combinatorial optimisation problems, in the real instances of the academic space allocation problem it is usually desirable to present a set of high quality solutions so that a human administrator can decide which allocation will be finally implemented (Burke and Varley, 1998). In such situations, two possible ways of achieving this are suggested here: reinitiate the single-solution hybrid metaheuristic to find several solutions, or use the population-based approaches. It is shown that the population-based techniques described here are capable of finding sets of high quality solutions. Given the considerable non-similarity between the solutions obtained (population diversity), it is clear that these solutions represent

very different allocations, which is valuable in some scenarions where one solution has to be chosen by the decision-makers. This observations motivated the investigation presented in chapter 6 on the multiobjective nature of this problem.

Chapter 6. Multiobjective Approaches

6.1. Introduction

In the previous chapters, the space allocation problem has been approached as a single-objective optimisation problem. The single goal has been the minimisation of the total penalty *F(x)* (eq. 2.7), i.e. the sum of space misuse and violation of soft constraints. This chapter investigates the space allocation problem from a multiobjective perspective based on the concepts of Pareto optimisation (Rosenthal, 1985; Steuer, 1986). We consider the multiple objectives separately and use the concept of dominance to assign fitness to solutions. Instead of combining all the criteria into a single scalar value, the solution fitness is represented by a *k*-dimensional vector containing all the *k* criteria. A solution *x* is said to be non-dominated with respect to a set of solutions if there is no other solution *x'* in that set that is as good as *x* in all the criteria and better in at least one of them. The Pareto optimal front is the set of non-dominated solutions with respect to the whole solution space. The aim in Pareto optimisation is to find the Pareto optimal front or a set of non-dominated solutions that constitutes a good approximation to that front.

Two main issues are investigated in this chapter. First, the hybrid metaheuristics developed in chapter 5 are adapted to approach the space allocation problem from a multiobjective perspective. Then, an investigation of the influence that different fitness evaluation methods have on the performance of some multiobjective optimisation algorithms is carried out. Since non-dominated solutions represent the goal, the dominance relation is frequently used to establish preference between solutions in Pareto optimisation. It has been argued that using aggregating functions to evaluate the solution fitness in multiobjective optimisers is not adequate (Deb, 2001). Recently, relaxed forms of the dominance relation have been proposed in the literature for improving the performance of multiobjective optimisers (Kokolo et al., 2001). It is shown in this chapter that the method used to evaluate the fitness of candidate solutions during the search affects the performance of the algorithms tested here and it appears that the dominance relation is not always the best method to use, in particular if the search space is highly constrained. The research work presented in

this chapter is included in the papers [Bur2002] and [Bur2003] (see the appendix on page 199).

6.2. A Brief Review of Multiobjective Optimisation

6.2.1. Multiple Criteria Decision-Making

In multiobjective combinatorial optimisation problems, various criteria exist to evaluate the quality of the solution and there is an objective (minimisation or maximisation) attached to each of these criteria. It is commonly the case that some of the criteria are in conflict, i.e. an improvement in one of them can only be achieved at the expense of worsening another. Moreover, some of the criteria may be incommensurable, i.e. the units used to measure the compliance with each of the criteria are not comparable at all.

The first decision that has to be made when dealing with a multiobjective optimisation problem is how to combine the search and the decision-making processes. This can be done in one of three ways (Steuer, 1986; Goicoechea et al., 1982):

- **Decision-making and then search.** Also known as the *a priori* approach because the preferences for each of the objectives have to be set by the decision-makers and then, one or various solutions satisfying these preferences have to be found.
- **Search and then decision-making.** This is also known as the *a posteriori* approach because various solutions have to be found and then, the decision-makers select the most adequate. All the solutions presented to the decision-makers should normally represent a trade-off between the various objectives.
- **Interactive search and decision-making.** In this approach the decision-makers intervene during the search in order to guide it towards promising solutions by adjusting the preferences in the process.

Another important decision in multiobjective optimisation is how to deal with the multiple objectives. At present, three methods can be identified in the literature (Coello Coello, 2000; Coello Coello et al., 2002):

- **Combine the objectives.** This is one of the classical methods to evaluate the solution fitness in multiobjective optimisation. It refers to converting the multiobjective problem into a single-objective one by combining the various criteria into a single scalar value. The most common way of doing this is by setting weights to each criterion and then adding them all together using an aggregating function. This is the approach used in previous chapters in this thesis.

- **Alternating the objectives.** This approach refers to optimising one criterion at a time while imposing constraints on the others. The difficulty here is establishing the ordering in which the criteria should be optimised since this can have an effect on the success of the search.

- **Optimising all objectives simultaneously (Pareto optimisation).** In this method, a vector containing all the objective values represents the solution fitness and the concept of dominance is used to establish preference between solutions.

Commonly, in the first two methods, preferences are established a priori (decision-making and then search) while in Pareto optimisation, no preferences are considered or are available before the search (search and then decision-making).

6.2.2. Pareto Optimisation

Formally, the dominance relation is described as follows (Dasgupta et al., 1999):

Suppose we have two distinct vectors $V = (v_1,v_2,\ldots,v_k)$ and $U = (u_1,u_2,\ldots u_k)$ containing the objective values of two solutions for a k-objective minimisation problem, then:

- V strictly dominates U if $v_i < u_i$, for $i = 1,2,..,k$.
- V loosely dominates U if $v_i \leq u_i$, for $i = 1,2,..,k$ and $v_i < u_i$, for at least one i.
- V and U are incomparable if neither V (strictly or loosely) dominates U nor U (strictly or loosely) dominates V.

Other researchers refer to *strict dominance* and *loose dominance* as *dominance* and *weak dominance* respectively (Zitzler, 1999). Minimisation is considered here because of the problem tackled in this thesis, but the above definitions are altered in

the obvious way for the case of maximisation problems. It is important to note that using strict or loose dominance can have an effect on how the search is performed. This is because if a solution is strictly dominated then it is outperformed by the other solution in all criteria while if the solution is loosely dominated it means that it is outperformed in some of the criteria but as good as the other solution in at least one of them. Then, finding a new solution that strictly dominates the current one may be more difficult than finding a solution that loosely dominates it. This is particularly true in some combinatorial problems in which the connectedness of the search space is such that some solutions are more difficult to reach from the current one. In such cases, using loose dominance may enable more solutions to be reached (Ehrgott and Klamroth, 1997).

In this thesis, strict dominance is used to distinguish a dominated solution from a non-dominated one, i.e. only solutions that are strictly dominated are rejected. This means that solutions that are loosely dominated are also considered because of the interest in obtaining diversity in the solution space. In the rest of this document, *strict dominance* is referred to as dominance.

A solution x is said to be non-dominated with respect to a set of solutions S if there is no other solution in S that dominates x. The Pareto-optimal front in multiobjective optimisation is the set of all non-dominated solutions in the whole solution space (Coello Coello et al., 2002; Deb, 2001; Steuer, 1986). When there is no knowledge of the localization of the Pareto-optimal set, the set found should be referred to as the obtained non-dominated set or the known Pareto front. In the test instances of the problem tackled in this thesis, there is no knowledge about the localization or shape of the Pareto-optimal front.

The appeal of Pareto optimisation derives from the fact that in most multiobjective optimisation problems there is no *single-best* solution or global optima and it is also very difficult to establish preferences among the criteria before the search process is carried out. Even when this is possible, it may be that these preferences change and having a set of solutions eases the decision-making process. One of the conditions that must be satisfied for a problem to be considered to be *truly* multiobjective is that the criteria are in conflict. Two objectives are in conflict if the

complete satisfaction of one of them prevents the complete satisfaction of the other. If any improvement in one of the objectives induces a detriment on the other, then the objectives are said to be strictly conflicting (Bagchi, 1999). It has been argued by some researchers that even if the conflicting nature of the criteria is not proved, Pareto-based metaheuristics would be able to find the ideal solution that is the best in all criteria (Fonseca and Fleming, 1995).

Since in Pareto optimisation the final outcome must be a set of non-dominated solutions, another important aspect to consider is how to evaluate the quality of the obtained non-dominated front. This is a multiple criteria problem on its own because several aspects have to be considered to determine how good the obtained front is. Among these aspects there are the following (Zitzler, 1999; Deb, 2001):

- The number of non-dominated solutions obtained.
- The closeness between the obtained front and the Pareto optimal front (if known).
- The coverage of the front, i.e. the spread and distribution of the non-dominated solutions.

Several methods have been proposed to evaluate the quality of the obtained non-dominated front in Pareto optimisation and therefore, assess the performance of multiobjective optimisers (Fonseca and Fleming, 1996; Van Veldhuizen and Lamont 2000b; Knowles and Corne, 2002). Since the Pareto optimal front is defined with respect to the objective space, is it common that most of the metrics proposed are also defined with respect to the objective space. One aspect that is frequently overlooked is the diversity of the obtained front with respect to the solution space. In fact, when researchers report on the quality of the obtained non-dominated sets it is very rare for information to be provided regarding the diversity in the solution space. This is extremely important because although the obtained non-dominated solutions may be well spread and distributed over the front in the objective space, it may be that the solutions are structurally very similar between them. Considering diversity in the solution space when assessing the quality of the obtained front becomes even more important in real-world multiobjective combinatorial optimisation problems (like the one tackled in this thesis) because this type of similarity directly relates to how different the solution structures are.

Large multiobjective combinatorial optimisation problems are particularly difficult to tackle. One reason for this is that the size of the search space grows exponentially as the problem size increases. Also, theoretical understanding of the solution space is lacking and as a consequence, in many problems of this type, there is no notion of the localization and shape of the Pareto optimal front (Ulungu and Teghem, 1994).

6.2.3. Metaheuristics for Multiobjective Optimisation

This section provides an overview of some proposed techniques for Pareto optimisation but no attempt is made to present an exhaustive survey of the field. This brief review is limited to multiobjective metaheuristics, in particular to evolutionary algorithms and approaches based on local search, and does not cover classical techniques because they are not relevant to the work reported in this thesis. The classical methods (also called traditional methods in the literature) include weighting approaches, goal programming, constraint methods, the Tchebycheff method and others. For reviews on classical techniques for multiobjective optimisation refer to (Steuer, 1986; Belton et al., 2002; Goicoechea et al., 1982; Miettinen, 2001).

In recent years, metaheuristics have received considerable attention in the area of multiobjective optimisation. Several surveys on the application of metaheuristics to multiobjective optimisation are available in the literature (Coello Coello, 1999; Coello Coello, 1999a; Van Veldhuizen and Lamont, 2000; Ehrgott and Gandibleux, 2000; Jones et al., 2001). Also, there are several studies that focus on measuring and comparing the performance of different algorithms for multiobjective optimisation (Zitzler and Thiele, 1998; Zitzler et al., 2000; Van Veldhuizen and Lamont, 2000b; Zydallis et al., 2001; Tan et al., 2001; Purshouse and Fleming, 2001).

Multiobjective Evolutionary Algorithms

A number of multiobjective evolutionary algorithms have been proposed in recent years and the increasing interest in these methods has motivated the extension of evolutionary algorithms (originally proposed for single-objective optimisation) to multiobjective variants. See (Coello Coello, 2001) for a brief tutorial on this topic. Some of these algorithms are briefly described next.

Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985). This is perhaps the first genetic algorithm that used dominance for evaluating and selecting individuals. In each generation, a group of individuals is selected according to one of the *k* objectives in the problem until *k* groups are formed. That is, each group of individuals excels in one of the *k* criteria. Then the *k* groups are shuffled together and the genetic operators are applied to produce the new population.

Multiobjective Genetic Algorithm (MOGA) (Fonseca and Fleming, 1993). In this algorithm each individual is assigned a rank according to the number of individuals in the population by which it is dominated, i.e. all non-dominated solutions are assigned rank 1. The fitness is assigned to each individual using an interpolation between the best and the worst rank. A scheme for niche formation is used in which fitness in the objective domain is shared among non-dominated individuals in order to maintain a uniform distribution of individuals over the trade-off surface. The fitness of all individuals in the same rank is averaged and this value is assigned to all of them. A more recent version of this algorithm is described and compared against other methods in (Purshouse and Fleming, 2001).

Niche Pareto Genetic Algorithm (NPGA) (Horn et al., 1994). The selection of individuals is carried out using a tournament scheme based on the concept of dominance. The two individuals competing for selection are compared against a subset of the population and the one that is non-dominated (assuming the other is dominated) is selected for reproduction. If both competitors are dominated or non-dominated, a sharing scheme based on the size of the niche (equivalence class sharing) is used to break the tie. The improved version of this algorithm, called NPGA-2 is described in (Erickson et al., 2001).

Non-dominated Sorting Genetic Algorithm (NSGA) (Srinivas and Deb, 1995). This algorithm also classifies individuals according to dominance in a ranking scheme similar to the one used in (Fonseca and Fleming, 1993). However, a dummy fitness value proportional to the population size is determined for each dominance class. Fitness sharing within the same class is also implemented to help maintain a well-distributed population over the trade-off front. Once the whole population is classified, a stochastic remainder proportionate selection scheme is used to ensure

that the individuals in the first front get more copies for reproduction than the rest of the population. Updated versions of this algorithm incorporating elitism are described in (Deb, 2001).

Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler and Thiele, 1999). This algorithm was proposed as an approach that incorporates several of the desirable features of other multiobjective evolutionary algorithms. The three features common to other approaches and put together here are: the use of dominance to evaluate and select solutions, the use of additional populations to store non-dominated solutions and the use of a niching or clustering scheme. The particular feature in this approach is that the non-dominated individuals in the external population are used to determine the fitness of individuals in the current population and also participate in the selection process for reproduction. In addition, a niche method based on Pareto dominance is proposed which does not require any measure of distance between individuals as in other clustering techniques. The improved version of this technique, called SPEA2 is described in (Zitzler et al., 2001).

Pareto-Archived Evolutionary Strategy (PAES) (Knowles and Corne, 2000). This algorithm starts with one randomly initialised solution and then, one candidate solution is generated in each iteration by means of mutations. An external archive (of limited size) is maintained to collect non-dominated solutions. An adaptive grid that divides the objective space is used to evaluate how much crowded the region (in which each solution lies) is. The candidate solution is discarded if it is dominated by the current solution or any other solution in the external archive. The candidate solution is added to the archive and becomes the current solution if it dominates the current solution. If none of them dominates the other, the decision as to which solution becomes the current solution and whether to add or not the candidate solution to the archive is made based on the crowding mechanism. Other variants of this algorithm with population sizes greater that one, were also proposed (Knowles, 2001).

Other Multiobjective Evolutionary Algorithms. The algorithms above are just a sample of the vast number of methods proposed in the literature in recent years. Other approaches include the multiobjective messy genetic algorithm (MOMGA) I

and II (Van Veldhuizen and Lamont, 2000) and the Pareto converging genetic algorithm (PCGA) (Kumar and Rockett, 2002). Another multiobjective genetic algorithm was proposed in (Murata et al., 1996; Murata et al., 1996b). Subsequent variants of this algorithm were presented in (Ishibuchi et al., 1997; Ishibuchi and Murata, 1998; Murata et al., 2000; Murata et al., 2001; Ishibuchi et al., 2002; Ishibuchi et al., 2002a). In the last two years, many other extensions of evolutionary algorithms for multiobjective optimisation have been proposed. For example, variants of micro-genetic algorithms, cellular genetic algorithms, particle swarm optimisation methods, agent-based algorithms and others can be found in proceedings of recent conferences in this area (EMO 2001, EMO 2003, CEC 2002, GECCO 2002, GECCO 2003, PPSN VII).

Other Multiobjective Metaheuristics

Another class of metaheuristics for Pareto optimisation are those that explicitly use local search or neighbourhood exploration (instead of genetic operators) to drive the search or as an important component of the process (hybrid approaches). Several multiobjective metaheuristics using local search have been put forward in the literature. Some of these multiobjective metaheuristics are briefly described below.

Simulated Annealing for Multiobjective Optimisation (Serafini, 1992). This was perhaps the first extension of simulated annealing for multiobjective optimisation reported in the literature. The proposed idea was to modify the acceptance criteria of candidate solutions in the original algorithm. Various alternative criteria were investigated in order to increase the probability of accepting non-dominated solutions. A special rule given by the combination of several criteria was proposed in order to concentrate the search almost exclusively on the non-dominated solutions.

Multiobjective Tabu Search (MOTS) (Hansen, 1997). This algorithm is a population-based extension of the tabu search metaheuristic that uses a set of weights to guide the search towards the Pareto frontier. Each solution maintains its own tabu list and the weights are adjusted in order to keep the solutions away from their neighbours and therefore, attempt to cover the whole trade-off surface.

Pareto Simulated Annealing (PSA) (Czyzak and Jaskiewicz, 1998). This is a population-based extension of simulated annealing proposed for multiobjective combinatorial optimisation problems. The population of solutions explore their neighbourhood in a similar manner to classical simulated annealing, but weights for each objective are tuned in each iteration in order to assure a tendency to cover the trade-off surface. The weights for each solution are adjusted in order to increase the probability of moving away from the closest neighbourhood in a similar way as in the multiobjective tabu search algorithm (Hansen, 1997). From simulated annealing, this hybrid metaheuristic borrows the idea of neighbourhood search, probabilistic acceptance of candidate solutions and the dependence of this acceptance from a temperature parameter. From genetic algorithms, the approach incorporates the idea of using a sample population of interacting solutions.

Multiobjective Simulated Annealing (MOSA) (Ulungu et al., 1999). This approach is another extension of simulated annealing in which a weighted aggregating function is used to evaluate the fitness of solutions to attempt approximating the various regions of the trade-off surface. The algorithm works with only one current solution but maintains a population with the non-dominated solutions found during the search.

Evolutionary Local Search Algorithm (ELSA) (Menczer et al., 2000). This is an evolutionary algorithm that uses local selection as the main component in order to minimise the interaction between the individuals in the population. The idea behind this approach is that a population of competing individuals can search the space in a parallel fashion. This algorithm does not use recombination and the only operator to generate new solutions is mutation. The authors stressed that the major strengths of this algorithm are its potential to be implemented in parallel and that it maintains the diversity of the population in a way similar to fitness sharing but more efficiently.

Memetic PAES (M-PAES) (Knowles and Corne, 2000b). This is a memetic variant originated from the PAES method. This memetic algorithm incorporates a population and a crossover operator but uses the same selection mechanism as PAES. Two archives are used, one is the global archive of non-dominated solutions and another serves as the comparison set in the local search phase. The second archive is emptied

after each local search and filled again with solutions from the global archive. The authors reported that this memetic version outperformed the original algorithm on test instances of the multiobjective knapsack problem.

Genetic Local Search (GLS) (Jaszkiewicz, 2002). This algorithm is a hybrid between genetic algorithms and local search in which a weighted aggregating function is generated at random in each iteration. This function is used to select the solutions that will be recombined to form the offspring and to guide the local optimisation of this offspring.

Simulated Annealing for Multiobjective Optimisation (Suppapitnarm et al., 2000). This is another extension of simulated annealing in which one temperature is associated to each objective in the problem. The algorithm uses only one solution and the annealing process adjusts each temperature independently according to the performance of the solution in each criterion during the search. An archive is used to store all the non-dominated solutions visited.

Other Multiobjective Metaheuristics Using Local Search. Many other approaches have been proposed and investigated in the literature. For example, the tabu search variant of (Baykasoglu et al., 1999) maintains a single solution but additional lists of non-dominated solutions found during the search are kept in order to seed and guide the search. Another tabu search approach using weights adaptation was proposed specifically for the bi-objective knapsack problem in (Gandibleux and Freville, 2000). Other multiobjective variants of ant colony optimisation, hybrids between tabu search and evolutionary algorithms and other implementations of multiobjective genetic local search can be found in proceedings of recent conferences (EMO 2001, EMO 2003, CEC 2002, GECCO 2002, GECCO 2003, PPSN VII).

6.3. Conflicting Objectives in Space Allocation

Using the dominance relation when dealing with a multiobjective optimisation problem makes sense only if the objectives are partially or totally conflicting. If the objectives are uncorrelated or reinforce each other, it is often adequate to combine all of them into a single scalar value and approach the problem as a single-objective one. More than two objectives could be considered in the space allocation problem as

described in chapter 2. In fact, it can be argued that this problem is an eight-objective optimisation problem, i.e. the satisfaction of each of the six types of constraints listed in section 2.4.1 plus the minimisation of space wastage and space overuse (eq. 2.10).

Sets of experiments were carried out in order to investigate the conflicting nature of the objectives in the space allocation problem. For the test problems nott1, trent1 and wolver1 described in section 2.5, eight sets of ten runs were executed using the single-solution hybrid metaheuristic described in section 5.2. In each set of ten runs, one of the eight objectives was subject to optimisation, i.e. only the value of that objective was used to assign fitness to solutions while the value of the other seven objectives were traced to observe their response. Since in each set of runs one of the objectives is subject to optimisation, it is possible to calculate the correlation between that objective and the others. A positive correlation is an indication that the two objectives are reinforcing each other or moving together, i.e. improvements in one objective are associated to improvements in the other. A negative correlation is an indication of the conflict between two objectives, i.e. improvements in one objective are associated with detriments in the other. A correlation value near to zero is an indication that the two objectives being unrelated or not affecting each other.

		Objective being traced							
		ws	**os**	**ai**	**af**	**at**	**tg**	**sh**	**gp**
Objective being optimised	**ws**	---	0.98	0.04	-0.04	-0.15	-0.70	-0.50	-0.40
	os	0.99	---	-0.35	-0.23	-0.61	-0.50	0.55	0.48
	ai	-0.21	0.34	---	0.18	0.88	0.24	0.02	0.02
	af	-0.48	0.02	0.06	---	0.04	0.06	0.28	0.03
	at	-0.82	-0.74	0.08	0.07	---	0.34	-0.01	0.53
	tg	-0.69	-0.69	0.30	0.05	0.30	---	0.60	0.60
	sh	-0.83	-0.83	0.06	0.01	-0.01	0.06	---	0.08
	gp	-0.24	-0.50	-0.04	0.02	0.72	0.77	0.54	---

Table 6.1. Correlation between objectives for the nott1 test instance.

The correlation values obtained in each set of ten runs were averaged for each pair of objectives. Results are presented in table 6.1 for the nott1 test problem. Each row corresponds to the objective being subject to optimisation and the columns in that row contain the correlation with each of the other (traced) seven objectives. The corresponding abbreviation for each objective is as follows: **ws** is *wasted space*, **os** is *overused space*, **ai** is *allocated in*, **af** is *away from*, **at** is *adjacent to*, **tg** is *together*

with, **sh** is *not sharing* and **gp** is *grouped with*. The negative correlation values corresponding to pairs of conflicting objectives are highlighted in table 6.1.

It can be observed that there is a high positive correlation between the minimisation of space wastage (**ws**) and the minimisation of space overuse (**os**). It appears then that these two objectives reinforce each other or cooperate strongly. On the other hand, it can be noted that in most of the cases, the correlation values between these two objectives and those corresponding to the satisfaction of soft constraints are negative or very near to zero. Only the minimisation of space overuse (**os**) has a relatively high positive correlation with the satisfaction of not sharing (**sh**) constraints and the satisfaction of grouped with (**gp**) constraints. It seems that the minimisation of space misuse is in conflict with the satisfaction of soft constraints in general. With respect to the correlation values between the six objectives associated to the satisfaction of constraints, it is observed that most of the values are positive and near to zero. Only two (very low) negative values were obtained corresponding to the correlations between **at** and **sh**. It appears that in general, the satisfaction of one type of soft constraints is not in conflict with the satisfaction of another type of soft constraint.Similar observations were made in the results obtained for the other two test instances. These results permit us to conclude that, at least on the test instances used in this thesis, not all the eight objectives are conflicting. We then grouped the eight objectives into two conflicting objectives: the minimisation of space misuse and the minimisation of soft constraint violation. It should be noted that the conflicting nature of the objectives will depend very much on the constraints that exist in each particular problem instance and therefore, an analysis similar to the one described here would be appropriate in order to illustrate the multiobjective nature of the problem.

In order to confirm that the two objectives considered here are conflicting, the experiments described next were carried out to observe the behaviour of each objective while the other was subject to optimisation. Two sets of ten runs were executed for each test instance (nott1, trent1 and wolver1) and each run was executed for a fixed number of iterations (20000, 10000 and 5000 respectively). In each set of runs, only one of the objectives was subject to optimisation (i.e. considered for evaluation of the solution quality) while the values of the other objective were

monitored during the search. For clarity, only two of each set of ten runs are shown in figures 6.1 to 6.3, but similar results (discussed below) were obtained in all runs.

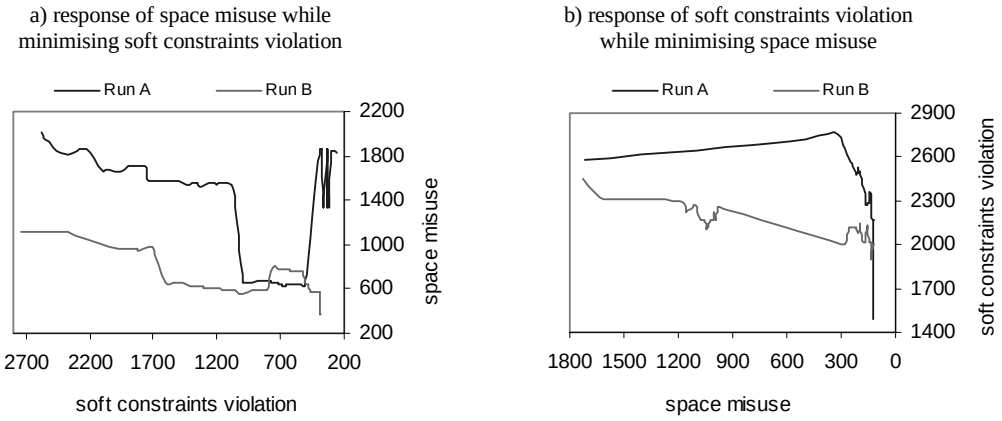


Figure 6.1. Response of one of the objectives while minimising the other using the single-solution hybrid metaheuristic on the nott1 instance.

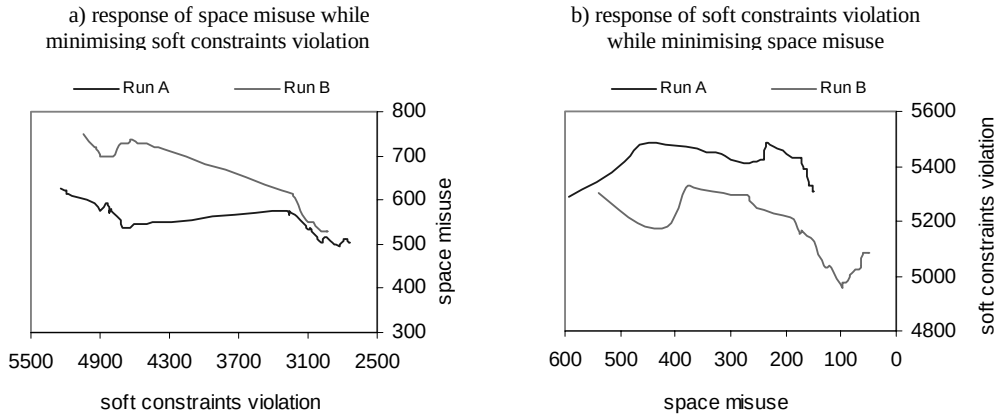


Figure 6.2. Response of one of the objectives while minimising the other using the single-solution hybrid metaheuristic on the trent1 instance.

The graphs presented in figures 6.1 to 6.3 show, to some extent, the conflicting nature of the two objectives in the space allocation problem: the minimisation of space misuse and the minimisation of soft constraints violation. For example, in figure 6.1.a it is observed that in both runs the space utilisation has to be worsened (space misuse increases in the graph) at some stages during the optimisation of the soft constraints satisfaction. Similarly, figure 6.1.b shows that the violation of soft constraints has to be increased if the space misuse is to be optimised. Note also that this behaviour can occur in an unpredictable way. While in the two runs in figure

6.1.a the conflict appears to be accentuated towards the end of the run, in figure 6.1.b the conflict between the two objectives occurs at different stages in each run. Moreover, if their corresponding graphs are compared, it is also apparent that the conflicting performance of the objectives is different for the three test problems. It should be noted that in the case of the wolver1 test instance, the particular shape of the graphs presented in figure 6.3.a can be explained because the problem has a small number of soft constraints which are satsified very easily at the begining of the search.

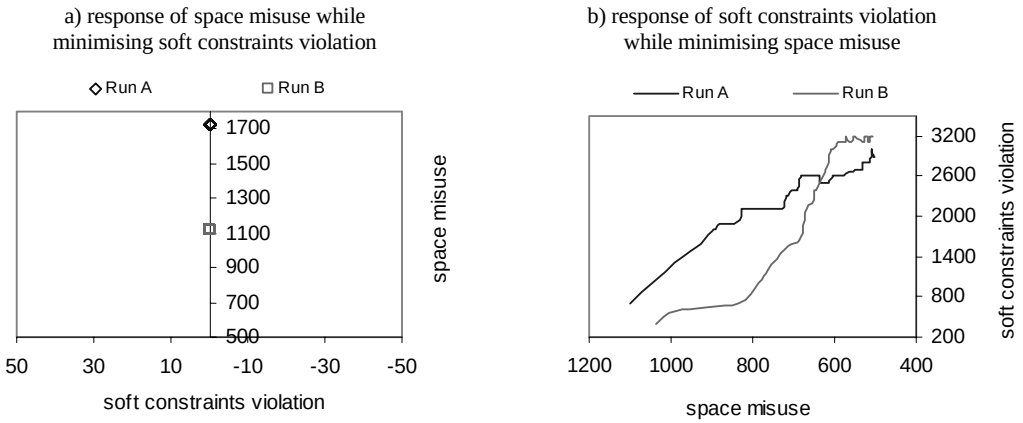


Figure 6.3. Response of one of the objectives while minimising the other using the single-solution hybrid metaheuristic on the wolver1 instance.

6.4. Pareto Optimisation of Space Allocation

6.4.1. Adapating the Hybrid Algorithms

This section assess the suitability of the hybrid metaheuristics presented in the previous chapter for the Pareto optimisation of the space allocation problem. The two algorithms were slightly modifed in order to apply them to the space allocation problem treated as a two-objective optimisation problem. A mechanism to archive non-dominated solutions found during the search was added. Solutions visited during the search can be considered for updating this external archive. Since both the single-solution and the population-based algorithms employ the $\mathbf{H}_{\mathbf{LS}}$ neighbourhood search heuristic of section 4.5.2, candidate solutions are generated which may replace the current solution if they are considered to be better than the existing one. Every time a candidate solution is generated, the dominance relation is used to decide if the new solution replaces the current solution or not. The external archive is not used for this

purpose, i.e. it is enough for the candidate solution to dominate the existing current solution in order to replace it. However, every candidate solution is considered for updating the archive of non-dominated solutions because even if the current solution is not replaced by the new one, the candidate solution may dominate some of the solutions in the archive. The purpose here is to investigate if these adapted versions of the algorithms (which perform well on the single-objective case) are capable of producing good results on the two-objective version of the space allocation problem.

6.4.2. Experiments and Results

In these experiments only the nott1 and trent1 test problems were used. Two reasons exist for this. On one hand the wolver1 test instance has been consistently the easiest to solve by the algorithms tested so far and, on the other hand, only few soft constraints exist in that instance so that it becomes almost a single-objective problem (as shown in figure 6.3). The experiments here consisted of applying the single-solution hybrid metaheuristic and the two versions of the population-based hybrid metaheuristic (PMHS and PMHM) to the test problems. Ten runs of each algorithm were executed on each test instance. The termination condition in each run was a number of idle iterations equal to $2 \cdot ReheatInterval$ as in the experiments of the previous chapter. Figure 6.4 shows the offline non-dominated populations (i.e. the non-dominated solutions collected after the ten runs) found by each algorithm.

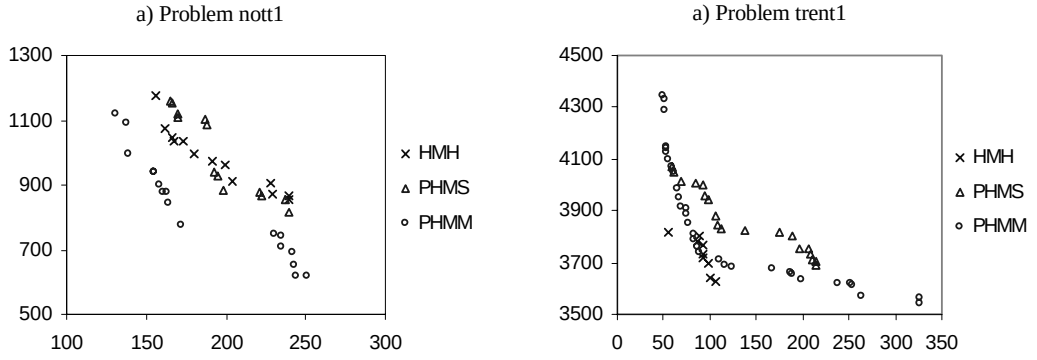


Figure 6.4. Non-dominated solutions obtained by the single-solution hybrid metaheuristic (HMH) and the two variants of the population-based hybrid metaheuristic (PHMS and PHMM) on the nott1 and the trent1 test instances.

Although the three algorithms are capable of producing non-dominated solutions, it is clear from figure 6.4 that for the nott1 test instance, the PMHM

algorithm multiple produces the best results since the solutions found by this approach dominate all the solutions found by the two other approaches. In the case of the test instance trent1, the solutions obtained with the single-solution hybrid metaheuristic dominate all solutions produced by the PMHS algorithm and some of the ones produced by the population-based hybrid metaheuristic multiple. However it is clear that in terms of the distribution and spread of the solutions, the results produced by the single-solution hybrid metaheuristic are not competitive. Similar experiments were carried out with the nott1b and nott1c tests instances and the same observations were made. From these results, it was clear that among these three methods, the PMHM algorithm obtains the best sets of non-dominated solutions overall. Since this approach is slightly different (dominance-based fitness evaluation and archive of non-dominated solutions added) from the one described in chapter five, in the rest of this chapter this modified version is referred to as the population-based hybrid annealing algorithm (PBAA).

6.5. The Influence of the Fitness Evaluation Method

6.5.1. Assigning Fitness to Solutions in Pareto Optimisation

In Pareto optimisation we usually wish to establish the way in which the various objectives will be handled in order to assign fitness to candidate solutions during the search and therefore, decide which solutions will survive and which ones will be discarded. Three ways of doing this are investigated here: an aggregating function, the dominance relation and a relaxed form of the dominance relation. With aggregating functions, the two objective values are combined into a single scalar value as shown in section 2.4.2 (eq. 2.7). With this method, the solution with the smaller value of $F(x)$ is preferred or considered to be better. In Pareto dominance, the solution fitness is represented using a two-dimensional vector containing the values of the two objectives ($F1(x),F2(x)$) and preference between solutions is established as described in section 6.2.2. The relaxed dominance method is described in the next section.

6.5.2. Relaxed Pareto Dominance

Relaxed forms of Pareto dominance have been proposed by researchers as a means to improve the performance of multiobjective optimisers. For example, Kokolo et al. suggested the use of α-dominance for dealing with what they call dominance resistant solutions, i.e. solutions that are fairly inferior quantitatively but other solutions that dominate them are scarcely found (Kokolo et al., 2001). This variant of dominance establishes lower and upper bounds for trade-off between the objectives. In α-dominance, small detriments in one of the objectives are considered to be acceptable if this leads to an attractive improvement in the other objective.

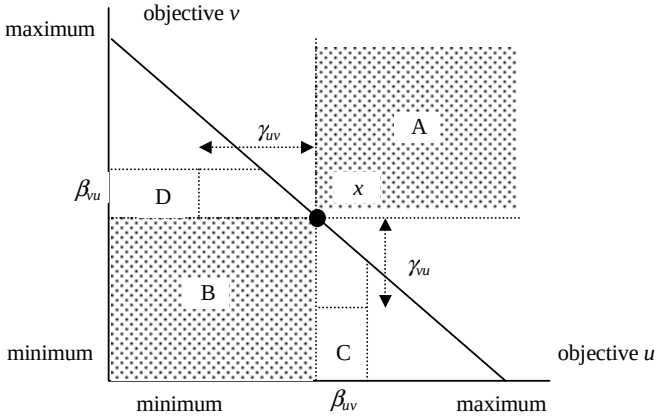


Figure 6.5. Three fitness evaluation methods: aggregating function, dominance relation and α-dominance (relaxed dominance) in a two-objective minimisation problem. Solutions in region A dominate *x*. Solutions in regions B, C and D α-dominate *x*. Solutions above the sloping line have a better aggregated value than *x*.

Figure 6.5 illustrates the concept of α-dominance for a two-objective minimisation problem and it also compares it to the other two evaluation methods considered here: dominance and aggregation of objectives. Solutions in regions B, C and D all α-dominate solution *x*. Then, in region C for example, β_{uv} represents the maximum detriment permitted in objective *u* given the minimum improvement γ_{vu} in objective *v*. In region D, β_{vu} and γ_{uv} are defined in a similar way. Solution *x* is dominated by all solutions in region B while solution *x* dominates all solutions in region A. When using the aggregation of objective values, a line that splits the objective space into two regions can be drawn. All the solutions above the line are considered to be worse than *x* and all solutions below the line are considered better

that x. A line at 45 degrees of inclination is used here according to equal weight values for the two objectives but different slopes will reflect different preferences.

In α-dominance, given an optimisation problem with k objectives, the relation between β_{vu} and γ_{uv} for each pair of objectives $u \neq v$ represents the relation between the detriment permitted in the objective v and the improvement obtained in the objective u. For the formal definition of α-dominance see (Kokolo et al., 2001). A similar form of relaxed dominance called ε-dominance was recently suggested by Laumanns et al. to implement better archiving strategies that permit us to overcome the difficulty of multiobjective evolutionary algorithms to converge towards the Pareto-optimal set and maintain a wide diversity in the population at the same time (Laumanns et al., 2002). In some sense, the relaxed forms of dominance (α-dominance and ε-dominance) are similar to establishing preferences among the objectives using weights in an aggregating function. In both cases, a detriment in one or more of the objectives is permitted in an attempt to widen the search by accepting not only dominating solutions. The different perspectives in viewing candidate solutions affects the way in which surviving solutions are selected. An algorithm may find it difficult to discover feasible solutions that dominate the current one(s). This is particularly true in highly constrained combinatorial optimisation problems like the one presented here. Then, by accepting α-dominating (or ε-dominating) solutions or solutions for which the aggregated value is better, it is possible to provide the algorithm with a wider view of the potential ways to approach the Pareto-optimal front.

The relaxed form of dominance implemented here for the two-objective space allocation problem follows the same principle as α-dominance and ε-dominance but it is slightly different. Let x be the current solution and x' be a candidate solution with fitness vectors given by $V = (v_1,v_2,\ldots,v_k)$ and $U = (u_1,u_2,\ldots,u_k)$ respectively. If the first objective in the candidate solution is better than in the current solution, i.e. $u_1 < v_1$, the corresponding *gain* or improvement proportion is $gain = (v_1 - u_1) / v_1$. The candidate solution x' is considered to be better than the current solution x if the detriment proportion in the other objective is at most *gain*, i.e. if $u_2 < v_2 \cdot (1+gain)$. This calculation is modified in the obvious way in the case $u_i < v_i$ for $i = 1,2,\ldots,k$.

6.5.3. Multiobjective Algorithms Tested

Justification

The two algorithms used in this investigation are: the population-based hybrid annealing algorithm of section 6.4 and the (1+1)-Pareto archived evolutionary strategy proposed in (Knowles and Corne, 2000). It was observed in preliminary experiments that when applying the population-based hybrid annealing algorithm to the two-objective space allocation problem, better non-dominated fronts were produced if the aggregation of objectives or the relaxed concept of dominance was used instead of the dominance relation to assign fitness to solutions during the search. In order to investigate whether this behaviour is due to the search strategy used by the algorithm or due to the problem domain, a multiobjective optimiser that has been well-studied in the specialised literature was also implemented and tested. The (1+1)-Pareto archived evolutionary strategy is a modern multiobjective optimisation technique that is simple to implement, it has been tested across a range of problems and it is considered to be competitive with other modern multiobjective evolutionary algorithms (Knowles, 2001; Tan et al., 2001).

The two approaches above are alike in the sense that both evolve solutions based on self-adaptation, i.e. the current solution is modified by mutation or local search and no recombination is used. Algorithms like these are often referred to as trajectory-based methods because the candidate solution is somehow similar to the existing one. The population-based hybrid annealing algorithm has been tested on various instances of the space allocation problem in previous chapters while the (1+1)-Pareto archived evolutionary strategy is an approach that has been applied to many other multiobjective optimisation problems but not to the one tackled in this thesis. Then, by using these two algorithms in this study, the effect of the fitness evaluation method can be further investigated without bias due to the algorithm design. Also, previous experience has shown that the recombination of solutions in this highly constrained problem almost always produces infeasible solutions (see chapter 4). Since both algorithms use local search as the main operator to generate candidate solutions, they show good performance when applied to the highly constrained two-objective space allocation problem. A brief description of the (1+1)-Pareto archived evolutionary strategy is given below.

The (1+1)-Pareto Archived Evolutionary Strategy

This algorithm starts with one initial solution and in each iteration, one candidate solution is generated by means of mutations. An external archive (of limited size) is maintained to collect non-dominated solutions. An adaptive grid that divides the objective space is used to evaluate how crowded the region in which each solution lies is. The candidate solution is discarded if it is dominated by the current solution or any other solution in the external archive. The candidate solution is added to the archive and becomes the new current solution if it dominates the old current solution. If none of them dominates the other, the decision on which solution becomes the current solution and whether to add or not the candidate solution to the archive is made based on the crowding mechanism, see (Knowles and Corne, 2000) for a detailed description. For the problem domain considered here, when a mutated solution is infeasible, successive mutations are tried until a feasible solution is generated. This is a very fast operation and it worked well in this implementation.

6.5.4. Experimental Settings

The nott1, nott1b and trent1 test instances described in section 2.5 were used in these experiments. For each test instance and each fitness evaluation method (aggregation of objectives, dominance and relaxed dominance) ten repetitions of the experiments (as described next) were executed. An initial population of size 20 was generated as described above. The population-based hybrid annealing algorithm was executed for *eval* solutions evaluations. Since the Pareto archived evolutionary strategy evolves a single solution, one run of the algorithm corresponds to 20 executions for *eval/20* solution evaluations, one with each of the 20 initial solutions. That is, the same initial population was used in each set of runs comparing the three evaluation methods in the two algorithms, i.e. 10 different populations were generated and in total 90 runs were executed for each algorithm.

For the population-based hybrid annealing algorithm, the parameters were set as follows: $\alpha = 0.95$, $IntervalCounter = n$ and $ReheatCounter = 10 \cdot n$ (see figure 5.3). The number of maximum solution evaluations *eval* was set to 100000, 80000 and 50000 for the nott1, nott1b, and trent1 test instances respectively. The number of non-dominated solutions in the external archive was limited to 30 in both algorithms

although in some cases fewer solutions were obtained in the final set. In the rest of this chapter, the population-based hybrid annealing algorithm and the (1+1)-Pareto archived evolutionary strategy are referred to as PBAA and PAES respectively.

6.5.5. The Offline Non-dominated Sets

For each set of ten runs corresponding to the same triplet (algorithm, problem,fitness evaluation method) the offline non-dominated sets were collected and these are presented in figures 6.6 to 6.8. It is observed from figure 6.6 that for the nott1 problem, the non-dominated sets obtained with both algorithms using the relaxed dominance and the aggregating function are better than those sets produced using the standard dominance relation. For both algorithms, the relaxed dominance clearly produces better results than the dominance relation. Also for both algorithms, a considerable section of the front obtained using the relaxed dominance is dominated by the front obtained using the aggregating function with the exception of a few solutions at the top end of these fronts. That is, using the aggregating function seems to benefit the performance of the algorithms in finding more solutions with low violation of soft constraints (small values of *F2(x)*) but none of the solutions obtained have values of space misuse (*F1(x)*) as low as some of the solutions obtained using the relaxed dominance relation.

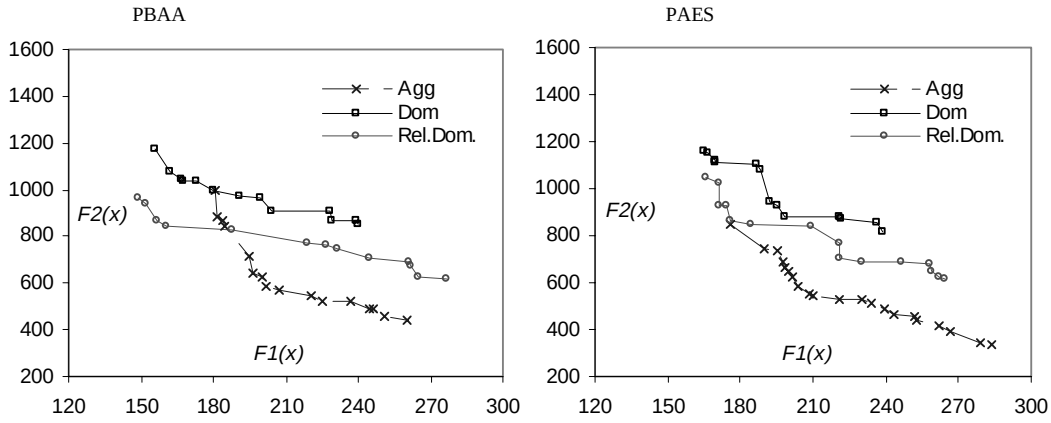


Figure 6.6. Offline non-dominated sets obtained by the PBAA and PAES algorithms with each evaluation method for the test instance nott1.

For the problem nott1b, figure 6.7 shows that the non-dominated sets obtained using the standard dominance and the relaxed dominance are comparable in the case of the two algorithms. That is, none of these two fitness evaluation methods appears

to clearly outperform the other. With PBAA some of the solutions obtained using dominance have better space utilisation while with the PAES many solutions obtained using relaxed dominance are better with respect to the satisfaction of soft constraints. It is noticeable that for both algorithms, none of the solutions obtained using the aggregating function is dominated by solutions produced with the other two fitness evaluation methods. However, as in the nott1 problem, using the aggregating function produces solutions that excell with respect to the minimisation of soft constraint violation (*F2(x)*) but solutions with very low values of space misuse (*F1(x)*) are not found.

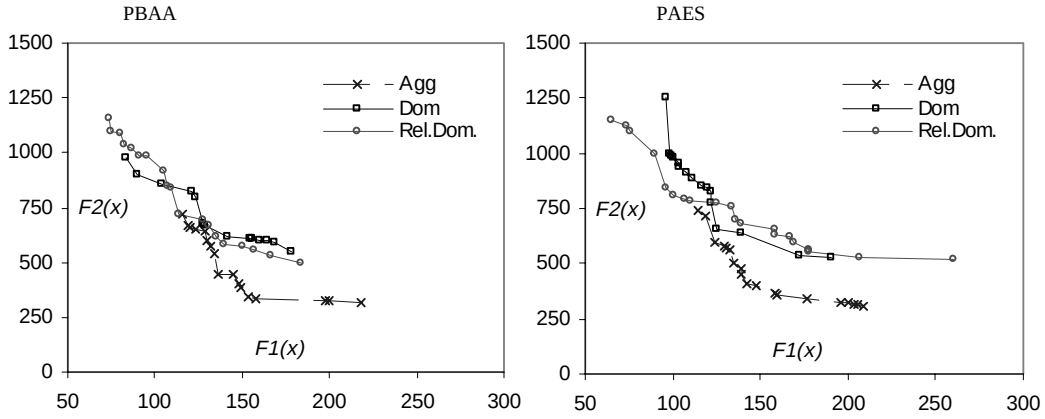


Figure 6.7. Offline non-dominated sets obtained by the PBAA and PAES algorithms with each evaluation method for the test instance nott1b.

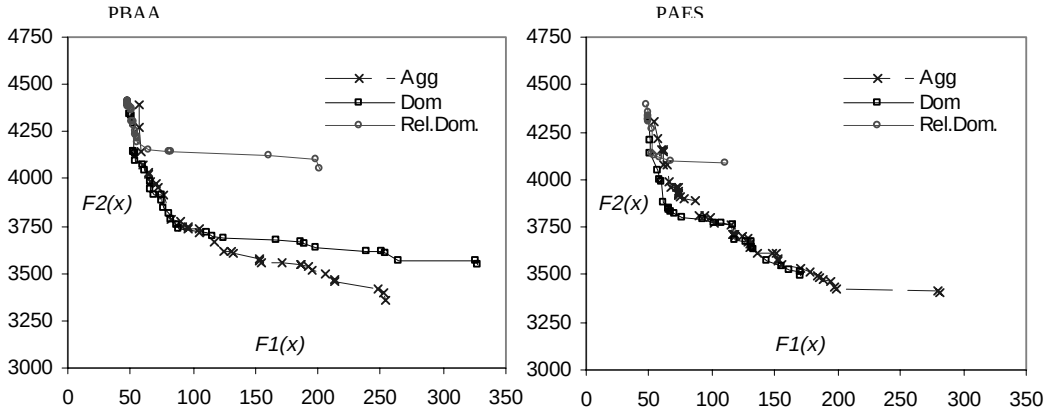


Figure 6.8. Offline non-dominated sets obtained by the PBAA and PAES algorithms with each evaluation method for the test instance trent1.

Figure 6.8 shows that for the trent1 problem, the comparison between the non-dominated sets obtained using the standard dominance and the aggregating function is very tight. In the case of PBAA the aggregating method outperforms the dominance relation with respect to the solutions in the bottom half of the front, i.e. solutions with low values of soft constraint violation. But in the case of the PAES, using the dominance relation generates a few solutions that dominate a section in the middle of the front produced with the aggregating method. Note that the results obtained using the relaxed dominance are very poor for both algorithms. Only a few solutions in the top end of the front produced with the relaxed form of dominance are competitive with those produced by the two other evaluation methods. It seems that when the relaxed dominance is used in problem trent1, both algorithms have difficulty in finding solutions with low values of soft constraints violation (*F2(x)*). One of the reasons for this behaviour might be the levels established for the relation between improvement in one of the objectives and the corresponding detriment in the other. This is further investigated later in this chapter.

6.5.6. The Online Non-dominated Sets

With respect to the online performance, the non-dominated populations obtained in the runs using the same algorithm on the same test instance but with the three different fitness evaluation methods were compared by using the metric proposed by (Zitzler et al., 2000). This metric was selected because it directly compares the quality of two non-dominated sets, it is not required to know the Pareto optimal front and it is simple to compute. Various other metrics are described in (Knowles and Corne, 2002). The metric by Zitzler et al. is described by the equation 6.1, where *A*, *B* are sets of non-dominated vectors.

$$C(A,B) = \frac{|\{b \in B;\ \exists\ a \in A : a \preceq b\}|}{|B|} \qquad (6.1)$$

A value of *C(A,B)* = 1 indicates that all solutions in set *B* are dominated by at least one solution in set A while a value of *C(A,B)* = 0 indicates that no solution in set *B* is dominated by a solution in set *A*. Ten values of *C(dom,agg)*, *C(agg,dom)*, *C(dom,reldom)*, *C(reldom,dom)*, *C(agg,reldom)* and *C(reldom,agg)* were computed

and averaged for each set of runs comparing the three fitness evaluation methods using the same algorithm and test problem. These results are shown in table 6.2.

By observing the comparison between the aggregating function results and the two other evaluation methods, it can be said that in general the aggregating function helps both algorithms to obtain the best results or at least it is as competitive as the relaxed dominance. Only for the PBAA method on the trent1 instance, the average coverage *C(dom,agg)* is slightly better than the average coverage *C(agg,dom)*. When comparing the results obtained with the stardard dominance and relaxed dominance, it is clear that for the nott1 instance the relaxed dominance is better. In the case of the nott1b instance, both strategies appear to be comparable along the ten runs. However, as mentioned above, in the trent1 instance the performance of both algortihms when using the relaxed dominance is very poor and beaten clearly by the standard dominance too. The following section presents and discusses results in terms of the population diversity.

	PBAA			**PAES**		
	nott1	nott1b	trent1	nott1	nott1b	trent1
C(dom,agg)	0.13	0	0.39	0.08	0.17	0.28
C(agg,dom)	0.99	0.76	0.23	0.96	0.81	0.32
C(dom,reldom)	0	0.58	0.94	0	0.51	0.92
C(reldom,dom)	1	0.49	0.14	0.98	0.64	0.21
C(agg,reldom)	0.65	0.54	0.97	0.77	0.62	0.96
C(reldom,agg)	0.41	0.43	0.10	0.26	0.35	0.17

Table 6.2. Comparing the online performance of each algorithm using the three evaluation methods, where *agg* = aggregating function, *dom* = dominance relation and *reldom* = relaxed dominance.

6.5.7. Results on Diversity

Table 6.3 shows the results with respect to the diversity *V(p)* (see section 2.4.3) of the non-dominated sets obtained in the experiments described above. For each set of 10 runs corresponding to the same triplet (algorithm, test problem, fitness evaluation method), the values of *V(p)* were averaged and these are shown as the online results in table 6.3. The values of *V(p)* were also computed for the offline populations collected after each set of ten runs and these are shown as the offline results in the same table.

It can be observed that with respect to the online performance, both algorithms obtain non-dominated sets with very similar diversities for the three fitness evaluation methods in the three test problems. In all cases, the relaxed dominance helps both algorithms to achieve slighlty more diverse populations but the difference with the other methods is almost insignificant. In the case of the offline non-dominated sets, although the results obtained with the three fitness evaluation methods are still very similar, greater differences between the diversity values obtained can be observed. For example, the aggregating function benefits PBAA in problems nott1 and nott1b and PAES in problem nott1b. The relaxed dominance method favors PBAA in the trent1 problem and PAES in the nott1 problem. The standard dominance relation helps PAES to obtain a slightly more diverse offline population in problem trent1. In general, it can be said from these results, that none of the three fitness evaluation methods seems to be clearly more beneficial than the others with respect to the population diversity that the two algorithms achieve. However, some improvement in the diversity of the obtained solutions can be noted when using the relaxed dominance and the aggregating function.

		PBAA			**PAES**		
		nott1	nott1b	trent1	nott1	nott1b	trent1
online	agregating	71.3	75.7	81.9	71.2	75.7	82.9
	dominance	72.1	76.9	81.5	73.6	75.9	81.8
(average)	relaxed dominance	72.5	78.2	84.4	73.8	77.5	83.6
offline	agregating	32.2	53.8	32.0	28.1	48.8	30.5
	dominance	27.0	39.1	32.8	29.7	30.5	33.6
	relaxed dominance	26.3	34.6	40.0	32.5	32.3	23.5

Table 6.3. Results on diversity for the online and offline non-dominated sets obtained with each algorithm when using the three different fitness evaluation methods.

In the next section, further experiments are carried out in order to investigate the reasons why the relaxed dominance appears to adversely affect the performance of both algorithms in the trent1 instance as noted in section 6.5.5.

6.5.8. Compromise Between Objectives in Relaxed Dominance

As described above, in the relaxed dominance relation used here, the detriment proportion acceptable in one of the objective values cannot be greater than the *gain* or improvement proportion obtained in the other objective value. If improvements

for one of the objectives are more difficult to achieve than for the other, then the above compromise may not be as beneficial as thought. This appears to be the case in the trent1 problem instance as revealed in the experiments and results presented next.

Given the results obtained with the relaxed dominance as evaluation method in the trent1 problem, it was decided to carry out more experiments with different levels of compromise between the two objectives. Consider the current and candidate solutions x and x' with fitness vectors $V = (v_1,v_2)$ and $U = (u_1,u_2)$ respectively. Four levels of trade-off between the two objectives were considered as described below.

Relaxed Dominance. In this case the compromise is set as described in section 6.5.2. In the three cases below *gain* is calculated as before.

Relaxed Dominance Variant A. Now, a greater detriment proportion is permitted in *F1(x)* given an improvement in *F2(x)*. That is, when $u_2 < v_2$ then x' is considered better than x if $u_1 < v_1 \cdot (1+10 \cdot gain)$. When $u_1 < v_1$, the detriment permited in v_2 is as before.

Relaxed Dominance Variant B. In this case, a greater detriment proportion is permited in *F2(x)* given an improvement in *F1(x)*. That is, when $u_1 < v_1$ then x' is considered better than x if $u_2 < v_2 \cdot (1+10 \cdot \text{gain})$. When $u_2 < v_2$, the detriment permited in v_1 is as before.

Relaxed Dominance Variant C. Now, the detriment proportion permited in *F2(x)* given an improvement in *F1(x)* is less than in the previous case. That is, when $u_1 < v_1$ then x' is considered better than x if $u_2 < v_2 \cdot (1+5 \cdot gain)$. When $u_2 < v_2$, the detriment permited in v_1 is as before.

The variant A refers to the case in which an improvement in the satisfaction of soft constraints (*F2(x)*) is more desirable and therefore more detriment in space misuse (*F1(x)*) is permitted. The other two variants reflect the case in which the improvement in space misuse (*F1(x)*) is considered more attractive and the detriment permited in the soft constraints satisfaction (*F2(x)*) is greater. Sets of runs were executed as described in section 6.5.4 but using only the above four variants of relaxed dominance relation on the trent1 instance. The results (offline non-dominated sets) of these experiments are presented in figure 6.9.

It is clear that the level of compromise between the objectives has an influence on the performance of both algorithms when solving the trent1 instance. Among the levels of compromise considered here, the best results are obtained when greater detriments in the satisfaction of soft constraints (*F2(x)*) are allowed given an improvement in the space misuse (*F1(x)*). This can be interpreted in two ways. It may be that improvements in *F1(x)* are difficult to achieve so they are highly welcomed no matter what the detriment caused in *F2(x)*. The other possibility is that improvements in *F2(x)* are the ones that are difficult to achieve so that this objective is permitted to deteriorate sometimes in order to find improvements later in the search. In order to find out which of the above possibilities is ocurring here, counters were maintained for the number of times in which the combination of improvement in one objective and detriment in the other led to the candidate solution being considered to be better. The results given next correspond to the relaxed dominance variant B (the one producing better results above).

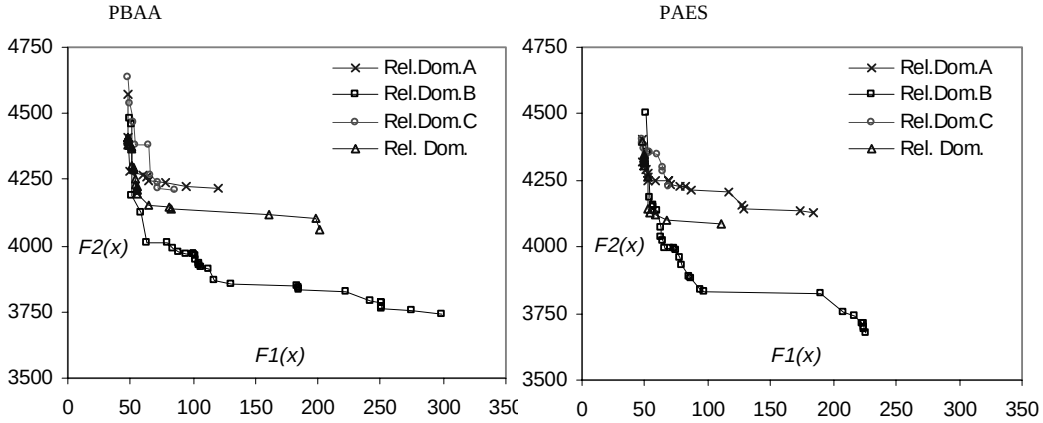


Figure 6.9. Offline non-dominated sets obtained by the PBAA and PAES algorithms using the four variants of the relaxed dominance relation for the test instance trent1.

In the case of PBAA, out of the total number of times in which an improvement in at least one of the objectives was achieved, 70% of these times *F1(x)* was improved and 32% of these times *F2(x)* was improved. The sum is greater than 100% since sometimes both objectives are improved. Out of the number of times in which *F1(x)* was improved, in 30% of these the detriment in *F2(x)* was acceptable and the new solution considered better than the current one. Out of the number of times in which *F2(x)* was improved, in 35% of these the detriment in *F1(x)* was acceptable and the new solution considered to be better than the current one. For PAES the

results are as follows: out of the total number of times in which an improvement in at least one of the objectives was achieved, 61% of the times *F1(x)* was improved and 41% of the times *F2(x)* was improved. Out of the number of times in which *F1(x)* was improved, in 43% of these the detriment in *F2(x)* was acceptable and the new solution considered to be better than the current one. Out of the number of times in which *F2(x)* was improved, in 35% of these the detriment in *F1(x)* was acceptable and the new solution considered to be better than the current one.

The above results suggest that, for the trent1 instance, finding candidate solutions with lower values of soft constraint violation (*F2(x)*) than the current solution is more difficult in general. Then, it seems that by relaxing the acceptance of solutions with higher values of *F2(x)* in the trent1 problem, the algorithms are provided with a wider view and these solutions may lead to better ones later on in the search. Finally, figure 6.10 compares, for the trent1 instance, the offline non-dominated sets obtained with the relaxed dominance variant B and the other two evaluation methods, standard dominance and aggregating function.

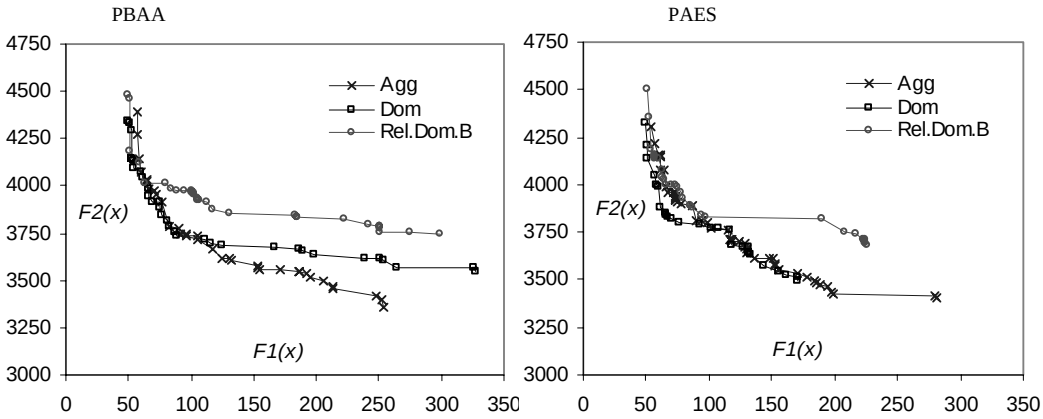


Figure 6.10. Offline non-dominated solutions obtained by the each algorithm with each evaluation method for the test instance trent1.

Although the non-dominated sets obtained with both algorithms, using the relaxed dominance variant B, are much better that the ones obtained with the original relaxed dominance (shown in figure 6.8), still the two other fitness evaluation methods help to obtain better results in both algorithms. In the next section, more results are presented in an attempt to investigate the effect of the fitness evaluation method used on the evolution of the objective values.

6.5.9. The Evolution of Objective Values

To investigate the effect of the fitness evaluation method on the evolution of the objectives, the values of *F1(x)*, *F2(x)* and *F(x)* for each individual in the best populations of PBAA (see figure 5.3) were recorded. The same was done for the current solution in PAES. Only a sample of the results are presented here, but the graphs shown below are typical of the observations made in all the runs of the experiments for both algorithms and the three test problems. Figures 6.11 to 6.13 show for the nott1 instance and the PBAA, the evolution of *F1(x)*, *F2(x)* and *F(x)* for one individual when each of the evaluation methods was used.

As expected, the values of *F1(x)* or *F2(x)* when using the aggregating function are sometimes worsened in favor of improving the aggregated value but frequently that detriment is temporal and the previous value is recovered or improved later on in the process. Similar observations can be made when the relaxed dominance is used to evaluate solution fitness. This, of course, cannot happen when using the standard dominance relation since the candidate solution is accepted only if at least one of the objectives is improved without worsening the other.

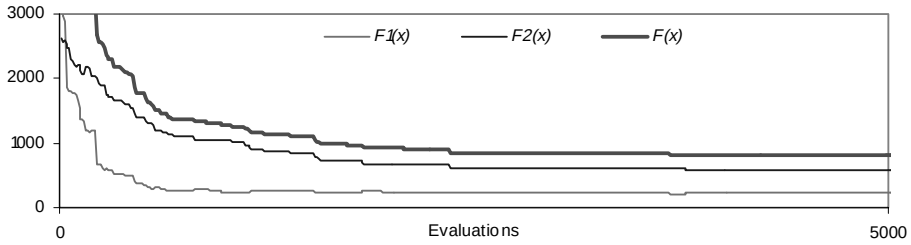


Figure 6.11. Evolution of the objective values for one individual in the best population of PBAA during a typical run using the aggregating function.

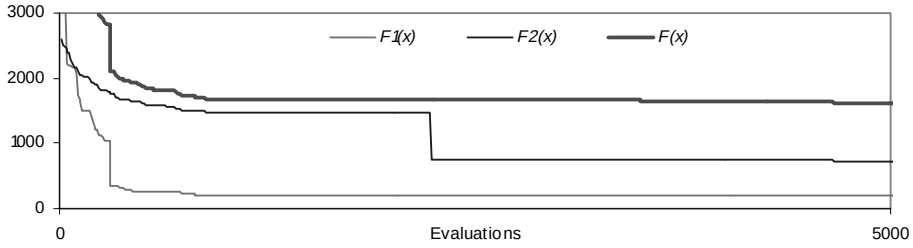


Figure 6.12. Evolution of the objective values for one individual in the best population of PBAA during a typical run using the standard dominance.

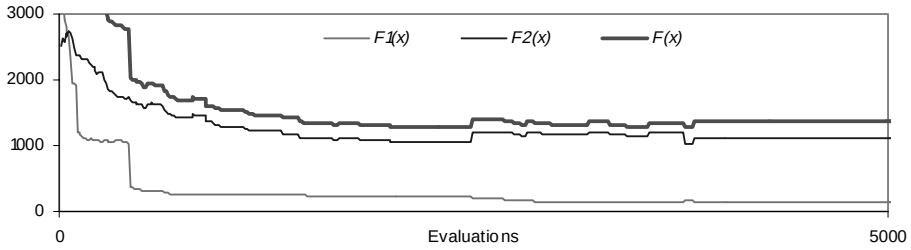


Figure 6.13. Evolution of the objective values for one individual in the best population of PBAA during a typical run using the relaxed dominance.

6.5.10. Further Discussion of Results

There is an increasing interest by researchers in various fields in the application of metaheuristics to multiobjective optimisation problems. Most of the published research on this subject has been focused on the development of new algorithms or extending existing single-objective ones towards multiobjective approaches. As we noted in section 6.2.3, a considerable number of papers report on the comparison between multiobjective optimisers on test and real-world problems. However, it is also fundamental to investigate the reasons why metaheuristics for multiobjective optimisation succeed or fail in certain problem domains to achieve a better understanding of their functioning in order to design more effective and efficient approaches. Although some research has been published on the effect that some strategies have on the performance of some metaheuristics for multiobjective optimisation (Deb, 1999; Laumanns et al., 2001), we believe that more research on this subject is required.

The research presented here aims to be a contribution to the better understanding of the mechanisms and conditions that influence the performance of multiobjective optimisers. The subject of study here has been the effect of the method used to assign fitness to solutions (and therefore select surviving ones) on the performance of some algorithms for Pareto optimisation. The fitness evaluation methods considered here were: the aggregation of objectives, the dominance relation and a relaxed form of this dominance relation. Arguments can be found in the literature in favor and against the use of aggregating functions or the use of dominance within metaheuristics for Pareto optimisation. For example:

- Some researchers have expressed that Pareto-based evolutionary algorithms are more suitable for multiobjective optimisation than local search methods using aggregation of objectives (Coello Coello et al., 2002; Deb, 2001) while other researchers have shown that approaches that use local search and aggregating functions are suitable for dealing with various multiobjective optimisation problems (Jaszkiewicz, 2002; Czyak and Jaszkiewicz, 1998; Gandibleux and Freville, 2000; Menczer et al. 2000; Ulungu et al. 1999).
- Knowles proposed and evaluated several approaches (single-point and population-based) for multiobjective optimisation based on a form of local search: mutation operators and using the dominance relation to evaluate solutions (Knowles, 2001).
- Jaszkiewicz expressed that "…*Pareto ranking is not well suited for hybridization with local search*" and found that weighted linear functions had better ability than Tchebycheff functions in finding potential non-dominated solutions within a genetic local search algorithm (Jaszkiewicz, 2002, page 54).
- Knowles et al. suggested that using the dominance relation can be beneficial even in single-objective optimisation for reducing the number of local optima (Knowles et al., 2001).
- Kokolo et al. illustrated the difficulty that approaches using dominance selection may exhibit in finding Pareto optimal solutions and suggested the use of relaxed dominance (α-dominance) instead (Kokolo et al., 2001).
- Laumanns et al. used ε-dominance (similar to α-dominance) to implement better archiving strategies that overcome the difficulty of multiobjective evolutionary algorithms to converge towards the optimal Pareto front and maintain a wide diversity in the population at the same time (Laumanns et al., 2002).
- The use of subcost guided search was proposed by Wright to deal with compound-objective timetabling problems (Wright, 2001). An improvement of a subcost (objective) is preferred even if the overall cost or solution fitness is not improved at all or it is worsened. The hope is that the detriment suffered will be repaired later in the process since the improvement in one aspect of the solution (the subcost) enables a kind of guided diversification towards promising areas of the solution space.

The above points show the various opinions (some of these conflicting) that researchers have expressed when referring to the fitness evaluation method used when implementing algorithms for multiobjective optimisation. In relation to this, the investigation carried out in this chapter shows the reader that, although an approximation to the Pareto optimal set is the aim, the (standard) dominance relation is not always the best method to assign fitness to solutions. The results from the experiments described here suggest that the performance of the multiobjective metaheuristics investigated is very much influenced by the method used to evaluate the fitness of solutions during the search process. The problem tackled here is a highly constrained combinatorial optimisation problem and the existence of constraints seems to be a reason for the difference observed in the performance of both algorithms when using different fitness evaluation methods. It is apparent that if it is more difficul to achieve improvements in one of the objectives (*F2(x)* here) than in the other (*F1(x)* here). Then, a compromise that allows detriments in the objectives should be made so that the algorithms are provided with better mechanisms to explore other areas of the solution space. In terms of both online and offline performance, the inferiority of the dominance evaluation method is evident. Between the aggregating function and the relaxed dominance it seems that the first one helps us to achieve better values on the minimisation of soft constraints violation (*F2(x)*) while with respect to the minimisation of space misuse (*F1(x)*) the relaxed dominance benefits the most. It also appears that the relaxed dominance evaluation method helps to achieve a better coverage of the intended compromise surface. However, the distance between the obtained non-dominated fronts and the intended compromise surface is shorter when using the aggregating method. In terms of diversity in the solution space for the obtained sets, the three methods seem to be competitive but a small inferiority of the dominance relation can be observed.

6.6. Summary and Final Remarks

This chapter presented an investigation into the space allocation problem from a multiobjective perspective. First, experiments were carried out to compute the correlation between the various criteria (six soft constrainst types, space wastage and space overuse) in order to determine if they were in conflict or not (Wright and

Marett, 1996). It was shown that not all the criteria are in conflict and therefore, they were grouped into two conflicting objectives: the minimisation of soft constraints and the minimisation of space misuse. Additional experiments were carried out to confirm the conflicting nature of these two objectives in the test problems used here. It was observed that, in general, while optimising one of these two objectives the other one suffered considerable detriment.

Given the two conflicting objectives, the hybrid metaheuristics developed in chapter 5 were adapted for the Pareto optimisation of the academic space allocation problem. These approaches were modified in order to collect a set of non-dominated solutions to be presented at the end of the search. It was observed by experimentation that the population-based hybrid annealing algorithm produced the best results overall. During the experiments, it was noted that this technique is capable of obtaining sets of non-dominated allocations that are also highly diverse. It can be suggested that this is because, instead of recombination, the algorithm is based on self-adaptation operators to evolve solutions. The cooperation among individuals within the population is encouraged by a mechanism to share information during the evolution process. Although it was shown that this population-based hybrid is an effective technique for the Pareto optimisation of the space allocation problem, experiments with other benchmark problems, like those proposed in (Deb, 1999; Zitzler, 1999; Knowles and Corne, 2000; Ulungu and Teghem, 1994), are required to validate the effectiveness of this approach in other problem domains.

An investigation was also carried out in this chapter to assess the influence that different methods of assigning fitness to solutions have on the performance of multiobjective optimisers. We questioned the circumstances (problem domain and search strategy) under which the dominance relation is the best alternative to identify improvement during the search. As shown in section 6.5, sometimes it is more beneficial to use the combination of objectives (aggregating functions) or relaxed forms of dominance (that allow detriment of objective values) for assessing solutions during the search in Pareto optimisation. An interesting future research direction is the evaluation of solution fitness using different strategies within the population. For example, some solutions in the population can be evaluated using dominance while others using an aggregated function and others using relaxed dominance.

Chapter 7. Hybrid Evolutionary Metaheuristics Based on Cooperative Local Search

7.1. Introduction

As noted in the literature review of chapter 3 (section 3.5.14), the hybridisation of metaheuristics has proven to be very successful in many applications. Among the hybrid approaches reported in the literature, some common ideas have been comprehensively explored while other alternatives appear less frequently. This chapter focuses on the concept of cooperative local search and proposes this method for extending a range of single-solution local search algorithms to hybrid evolutionary metaheuristics. Instead of incorporating local search into a population-based approach, a scheme that promotes the cooperation between various local searchers by sharing the information gained during the search is proposed.

At a high level of abstraction there are two ways in which the hybridisation of population-based algorithms (such as genetic algorithms) with local search-based techniques (such as simulated annealing or tabu search) can be achieved. One is adding local search components that 'help' the population-based method by providing it with 'intensification' mechanisms (Reeves, 1996b). The second way is to consider a population of local searchers and a powerful cooperation mechanism that allows then to 'help' each other. The first approach has received considerable attention and the hybrids obtained are commonly known as memetic algorithms, genetic local search, hybrid genetic algorithms and other names (Moscato, 1999; Moscato and Cotta, 2003). It has been shown that by adding intensification local search techniques to the explorative capability of genetic algorithms better results can be produced in many optimisation problems. See for example (Fox, 1993; Reeves, 1996; Reeves, 1996b; Glover et al., 1995).

The hypotheis presented here is that the second method of 'keeping' local search as the driving mechanism and 'helping' it when required to perform a better exploration can be effective in those combinatorial optimisation problems in which the recombination of solutions is not straightforward. This includes problems such as space allocation, scheduling, timetabling, grouping and other constrained problems

that require special attention when recombining solutions. In these cases, specific solution encodings, recombinative operators, reparing methods or unfeasibility penalty schemes have to be designed (Michalewicz, 1999). On the other hand, good local search heuristics can be (relatively) easily implemented for many of the problems mentioned above (Aarts and Lenstra, 1999). Then, by having a population of local searchers that share the information obtained during the search, a form of recombination can be achieved and the performance of the local search mechanism can be improved. In order to illustrate this form of hybridisation, a range of single-solution local search algorithms are extended towards hybrid evolutionary approaches by adding a population and a mechanism that promotes cooperation between the members of the population during the search. Experiments are carrried out to compare the performance of the original and the extended variants of the algorithms when applied to test instances of the space allocation problem. The main goal of this chapter is to propose and evaluate some ideas for hybridising metaheuristics particularly for problems where several high quality and diverse solutions are required and the design of recombinative operators requires extra effort. The research presented in this paper is included in the paper [Bur2003b] (see the appendix on page 199).

7.2. Hybridising Recombinative and Local Search Methods

The hybridisation of recombinative approaches and local search techniques has been extensively studied and, in particular, the integration of simulated annealing, tabu search and genetic algorithms has received considerable attention. See for example (Abboud et al., 1998; Chen and Lin, 2000; Fox, 1993; Glover et al., 1995). The hybrid metaheuristics proposed in the previous chapters are also examples of this type of hybridisation. Moreover, the incorporation of local search heuristics, specialised recombination/mutation operators and other ‘helpers’ specifically designed to exploit the knowledge of the problem domain into genetic algorithms has led to the development of so-called memetic algorithms (Moscato 1989, Moscato, 1999; Moscato and Cotta, 2003). The name memetic algorithms is a relatively recent terminology that attempts to include all algorithms that fit the description given above but other names for this class of methods include genetic local search, hybrid

genetic algorithms and others. See for example (Burke and Smith, 1999; Burke and Newall, 1999; Burke and Smith, 2000; Burke et al., 2001; Falkenauer, 1996; Ishibuchi et al., 1997; Jaszkiewicz, 2002; Reeves, 1996; Reeves, 1996b).

One of the most common strategies used by researchers and practitioners to design memetic algorithms is to add 'helpers' to an evolutionary algorithm (commonly a genetic algorithm). That is, the structure of the evolutionary algorithm based on the concepts of generations, recombination, selection and mutation is maintained and the knowledge of the problem domain is added to 'help' to achieve a better performance. This strategy is illustrated in figure 7.1.

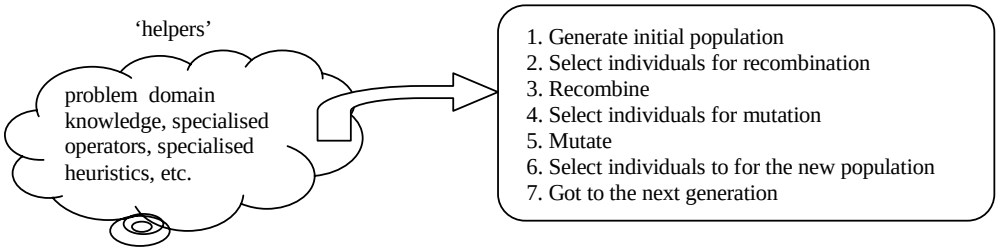


Figure 7.1. Common strategy for designing memetic algorithms.

The design of specialised recombination/mutation operators is not straightforward for some combinatorial problems such as scheduling, timetabling, rostering and related problems (Aickelin and Dowsland, 2000; Burke et al., 1995; Erben, 2001; Falkenauer, 1994). Also, dealing with highly constrained problems adds additional difficulties to the application of recombinative techniques (Coello Coello, 2000; Kellerer and Pferschy, 1999; Thiel and Voss, 1999). It must be said that despite these difficulties, many successful applications of recombinative techniques to these and other problems exist. See for example (Aickelin and Dowsland, 2000; Brizuela et al., 2001; Burke et al., 1995; Chambers, 2001; Chu and Beasley, 1997; Chu and Beasley, 1998; Falkenauer, 1994; Falkenauer, 1996). However, it can also be noted that not many hybrids based on the opposite philosophy have been investigated. That is, given a local search method to incorporate 'helpers' perhaps inspired from population-based methods that improve the explorative ability of the algorithm. In fact, very efficient single-solution local search heuristics have been developed for some combinatorial problems and their

possible extension to population-based approaches deserves attention. One way in which this concept can be implemented is by, what in this chapter is called, cooperative local search which will be described in the next section.

7.3. Cooperative Local Search

The goal here is to describe algorithms that were implemented following the idea of enriching local search methods with elements of recombinative approaches. In cooperative local search, there is a population of local searchers and each of them can be thought of as an explorer. Each explorer is associated with a particular solution. Several explorers can be made to cooperate by sharing the information that each of them obtains or learns during the search. This cooperation can be achieved, for example, by sharing promising parts of good discovered solutions. But also sharing 'bad experiences' among the population can prevent some explorers from being trapped in areas of poor solutions. This information sharing can be implemented by recombinative operators or by keeping track of good and bad moves. Moreover, these periods of cooperation are not necessarily driven by the principle of generations as in genetic algorithms. That is, each explorer searches the given solution space on its own and the cooperation occurs whenever it is required. It may be that some explorers achieve better results than others resulting in asynchronous converging times. Then those explorers that cannot achieve further improvement ask for the cooperation of others. This concept of cooperative local search is illustrated in figure 7.2.

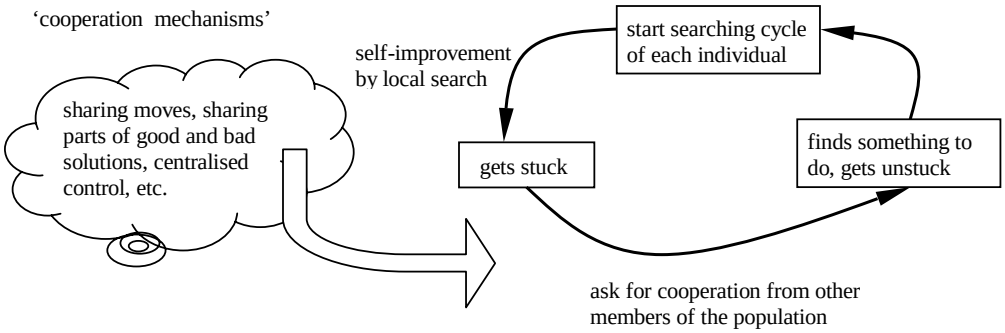


Figure 7.2. The *cooperative local search* scheme where each individual carries out its own local search. When an individual gets *stuck* it asks for the cooperation of the population in order to find something to do to get *unstuck* and continue the search from another position in the solution space. The results achieved by each individual may be different at different times and this encourages diversity within the population.

Each explorer can have its own intensification and diversification mechanisms and some degree of cooperation during the intensification phases could be permitted (for example, by means of a common control scheme). However, the central idea is to allow each explorer to do its own search and implement the cooperative phases when required. Using the terminology presented by Preux and Talbi (section 3.5.14) the cooperation can be synchronous or asynchronous, the explorers can use the same (homogeneous) or different (heterogeneous) search strategies and can search the same of different solution spaces (global, partial or functional). Similar concepts to the ones illustrated in figure 7.3 were proposed by Salman et al. in their implementation of a cooperative team of heuristics to solve a variant of the multiple knapsack problem (Salman et al., 2002).

7.4. Hybrid Evolutionary Metaheuristics

7.4.1. Relation to Previous Work

The application of various metaheuristic approaches, including genetic algorithms, to solve the space allocation problem has been investigated earlier in this thesis. It was shown in chapter 4, that despite designing specialised genetic operators to deal with the existing constraints, the genetic algorithm was outperformed by the other three local search techniques implemented: iterative improvement, simulated annealing and tabu search. Then in chapter 5, components from various metaheuristics were incorporated into one single-solution hybrid approach and it produced very good results. Also in that chapter, the single-solution hybrid approach was extended towards two population-based variants using the concepts of cooperative local search presented above. One population-based approach obtained a high quality solution (with the rest of the population being substantially less fit) in a short computation time while the other approach generated a set of high quality solutions at the expense of longer computation time. In this chapter, the cooperation mechanism described in section 5.4 has been enhanced as detailed next in section 7.4.2 and is used to extend a range of single-solution approaches to hybrid evolutionary variants.

7.4.2. The Cooperation Mechanism

As discussed in section 7.3, the design of a hybrid evolutionary approach based on the idea of cooperative local search could be implemented in several ways. The way in which this was done for the space allocation problem domain using the single-solution algorithms presented in chapter four is illustrated next. The cooperation mechanism implemented here attempts to promote the idea that individual explorers should share information during the search and it differs from the one in chapter five on the heavy mutation operator. The two matrices M_A and M_T (described in section 4.8.1) are shared among all individuals in the population in order to store the tabu and attractive moves explored by all individuals in a shared memory scheme. That is, this strategy can be regarded as a way of storing parts of attractive solutions in M_A and parts of unattractive solutions in M_T (genetic material in recombinative algorithms terminology). The information stored in the two matrices is used in the cooperative local search scheme in two ways:

Information sharing. Each explorer performs the neighbourhood exploration but the matrices are updated by all individuals in the population so that the whole population contributes to the tabu and attractive moves stored in M_T and M_A. When a single-solution explorer cannot get a feasible solution from the neighbourhood search heuristic $\mathbf{H_{LS}}$ (see section 4.5.2), i.e. when the *cooperation mechanism* is invoked, a heuristic is used to modify the solution using the information stored in M_A. This heuristic goes through each row i in the matrix and explores the most attractive allocations for that entity. That is, it starts with the cell having the highest value and continues to the one with the lowest value and makes the allocation entity to a room that is suitable (keeps the solution feasible) and is different from the one in the current solution. The changes are made even if the solution is worsened and in order to avoid a potential high disruption a maximum of $n/20$ changes are implemented in this way.

Heavy mutation. A mutation operator that 'heavily' disrupts the current solution is implemented as follows. Those entities that are penalised the most (are involved in the violation of soft constraints or in the misuse of space) are removed from their assigned rooms. Then, the allocation of each of these (now unallocated) entities to various rooms is attempted. For each entity, the rooms from the first to the last one

are evaluated for a feasible allocation with the exception of those allocations marked as tabu in the matrix M_T. The degree of disturbance carried out by the mutation operator is controlled by setting the maximum number of penalised entities that will be unallocated (many can be penalised). A maximum of *n/5* entities are permitted to be unallocated here. The purpose of this 'heavy' disturbance is to encourage each explorer to search a (hopefully) very different area of the solution space.

7.4.3. Extending the Single-Solution Approaches

Given a single-solution explorer (local searcher) LS_{SS} that takes the current solution *x* and attempts to find a better next solution *x'*, a hybrid evolutionary approach LS_{PB} based on cooperative local search can be designed. The three single-solution local search metaheuristics described in chapter 4 (iterative improvement, simulated annealing and tabu search) were extended to hybrid evolutionary algorithms as described in the pseudocode given in figure 7.3 in order to illustrate the idea of cooperative local search.

Extended Population-based Approach LS_{PB}

Step 1. Generate the initial current population.
Step 2. Archive the current population as the best population so far.
Step 3. Do
  Step 3.1. Do population self-improvement (intensification) updating the best population so far, i.e. each individual in the population executes the single-solution local search approach LS_{SS} using the *information sharing* mechanism and attempts to improve its own solution iteratively. This phase continues until no further self-improvement is possible, i.e. it terminates when none of the individuals in the population can improvement its current solution.
  Step 3.2. Do random variation of the population (diversification), i.e. since all individuals appear to be 'stuck', all of them are disturbed using the heuristic *heavy mutation* operator.
Step 4. Until the termination criterion is satisfied.

Figure 7.3. Hybrid evolutionary scheme based on *cooperative local search.*

The first phase (step 1) corresponds to the construction of a population of explorers each one associated to an initial solution. In the intensification phase (step 3.1) each explorer aims to achieve self-improvement using the *information sharing* mechanism. In the diversification phase (step 3.2), each explorer randomly modifies its current solution using the *heavy mutation* operator. The best solution found by each explorer is maintained in the best population so far. This population serves as an

archive that keeps the best solution visited by each explorer in the population. Note that although the improvement rate of some of the explorers could be better than others, each explorer has its own solution and none is permitted to contribute more than one solution to the best population so far. This has been decided for two reasons: 1) to encourage diversity in the population by avoiding one or more explorers to dominate the search, and 2) to assess the effect of the cooperation mechanism in the experiments presented later.

The detailed pseudocode for each hybrid evolutionary approach is not included here. However, note that the modification consists of replacing LS_{SS} by each single-solution technique in the pseudocode shown in figure 7.3 above. Then, the algorithm variants implemented here are the following: the iterative improvement algorithm of section 4.6 and its population-based variant (II_{SS} and II_{PB}); the simulated annealing algorithm of section 4.7 and its population-based variant (SA_{SS} and SA_{PB}) and the tabu search algorithm of section 4.8 and its population-based variant (TS_{SS} and TS_{PB}).

7.5. On the Performance of the Extended Approaches

7.5.1. Experimental Settings

Several sets of experiments were carried out in order to assess the validity of the concepts presented and described in the previous sections. The main issue was to evaluate whether it is beneficial or not to extend a single-solution technique towards a population-based approach as proposed above. The experiments were designed to compare the performance of the population-based variant against the performance of the corresponding single-solution technique for finding a set of high quality allocations which are also diverse with respect to the solution space. A fair way to do that is to execute each method for an equal computation time. The number of solution evaluations or neighbourhood move explorations is another possibility for comparison but because the population-based approaches spend extra time using the cooperation mechanism this could be unfair for the single-solution methods. Given a short computation time, the single-solution approaches quickly achieve improvement but they get stuck relatively early too while the population-based approaches can

take more time (relatively to the single-solution variants) to reach high quality solutions. With this in mind, experiments were carried out to find the computation time t_{ind} for which the single-solution approaches achieved no further improvement for a considerable number of iterations.

Given an initial population of size p, the single-solution approach was applied for t_{ind} computation time to each of the solutions in this population and the best solution at the end of each run was archived, i.e. p solutions are obtained. Then, the corresponding population-based approach was applied to the same initial population for $p \cdot t_{ind}$ computation time. This process was repeated ten times for each of the problem instances used here: nott1, nott1b, nott1c and trent1 (described in section 2.5). For each set of p solutions obtained, the best, average and worst solution qualities were recorded and these values were averaged for each set of ten repetitions. In order to further compare the performance of each population-based variant against its corresponding single-solution algorithm, experiments were carried out using small and large populations with low and high diversity for each test instance as shown in table 7.1 below. The results obtained from the experiments described here are presented and discussed in the following subsections.

	$p = 20$		$p = 5$	
	$65\% < V_{ip} > 90\%$	$20\% < V_{ip} < 40\%$	$65\% < V_{ip} > 90\%$	$20\% < V_{ip} < 40\%$
nott1 , $t_{ind} = 120$	nott1A	nott1A2	nott1B	nott1B2
nott1b , $t_{ind} = 60$	nott1bA	nott1bA2	nott1bB	nott1bB2
nott1c , $t_{ind} = 30$	nott1cA	nott1cA2	nott1cB	nott1cB2
trent1 , $t_{ind} = 70$	trent1A	trent1A2	trent1B	trent1B2

Table 7.1. Initial populations of different sizes and diversity values for the four test problems.

7.5.2. Results on the Fitness of Solutions

In this section, the single-solution approaches and corresponding population-based variants are compared with respect to the fitness of the solutions obtained. Each of the graphs in figures 7.4 to 7.7 summarises all the results obtained using the various initialised populations for one of the test instances. In each pair of bars in these graphs, the left bar refers to the results produced by the population-based variant of one algorithm, the right bar refers to the results obtained by the corresponding single-solution approach and a line is drawn between the averaged solution fitness obtained so that the comparison is clearer.

From figures 7.4 to 7.7 it is apparent that the solutions obtained by the population-based variants are better than those produced with the single-solution approaches. It can be observed that the best, average and worst solution qualities are better for the extended algorithms in most cases of each test instance. This is clear for the nott1 and nott1c test instances as shown in figures 7.4 and 7.6 respectively. In the results for the nott1b instance shown in figure 7.5, the extended simulated annealing algorithm is outperformed by the single-solution approach when the initial population is small and the diversity is low (nott1bB2). Also, in some cases the worst solution found by the population-based variant of one algorithm has a lower quality than the one found by the corresponding single-solution approach. This is true for the simulated annealing algorithm on the nott1bA, nott1bA2, nott1bB and trent1A2 cases and the tabu search algorithm on the nott1bB and nott1bB2 cases.

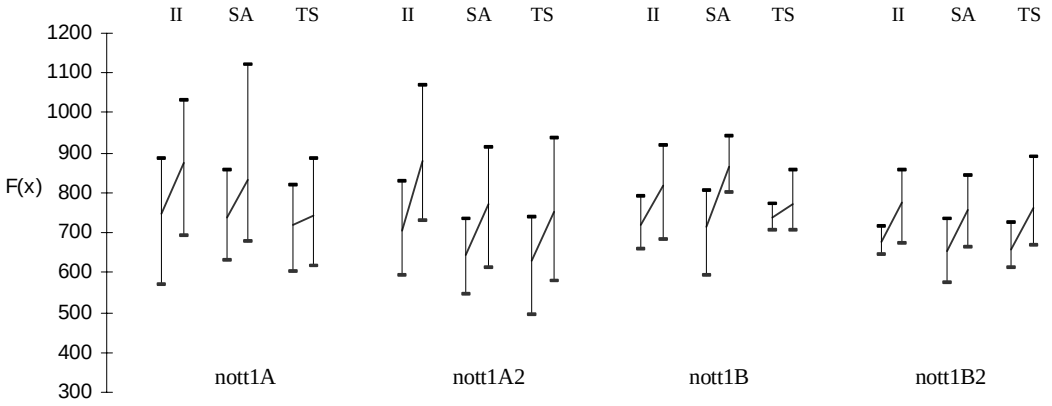


Figure 7.4. Results obtained by the hybrid evolutionary approaches for the problem nott1.

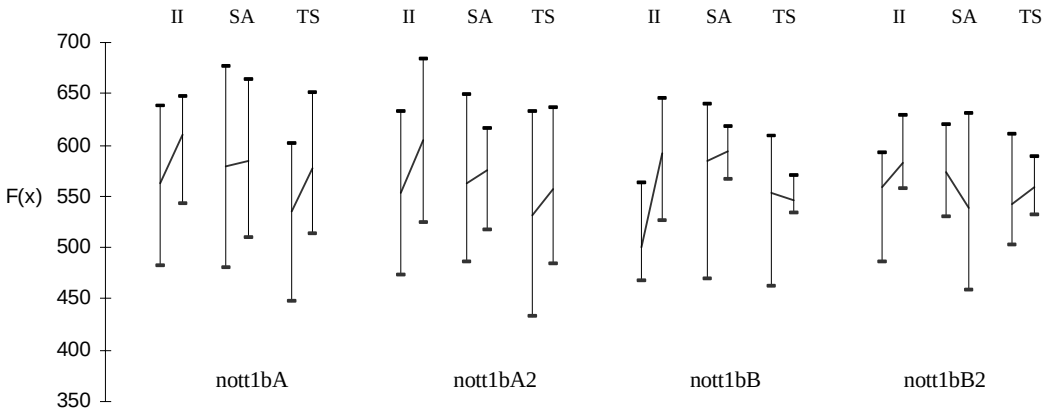


Figure 7.5. Results obtained by the hybrid evolutionary approaches for the problem nott1b.

Another important observation in these graphs is that in some cases, even the worst solution produced by the extended algorithm outperforms (or at least matches) the quality of the best solution found by the corresponding single-solution approach. This is true for the iterative improvement algorithm on most cases of the nott1c and trent1 problems, the simulated annealing algorithm on the nott1cA and trent1B cases and the tabu search algorithm on nott1cB, nott1cB2 and trent1A2 cases.

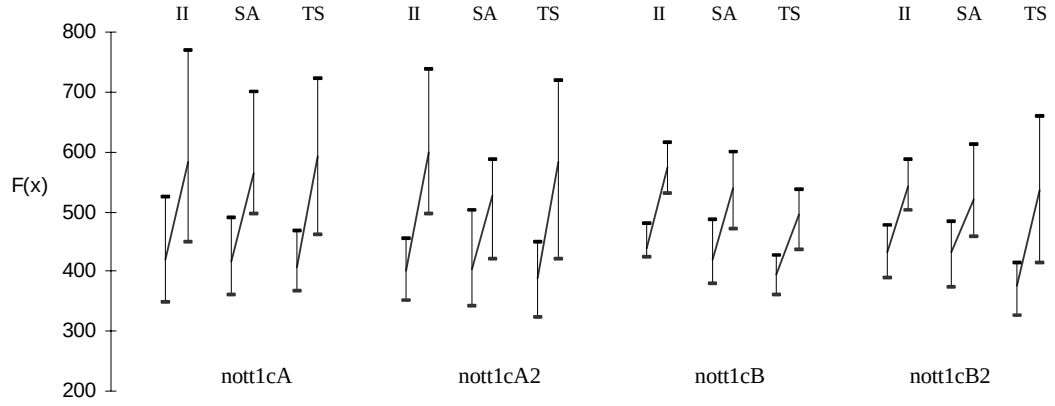


Figure 7.6. Results obtained by the hybrid evolutionary approaches for the problem nott1c.

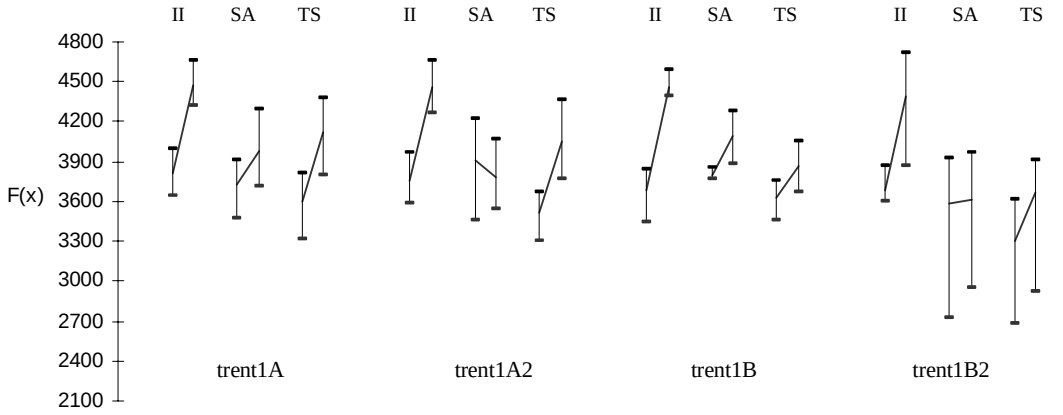


Figure 7.7. Results obtained by the hybrid evolutionary approaches for the problem trent1.

The size of each bar gives an indication of the difference in quality between the worst and best solutions found by each algorithm variant and it is observed that in general this difference appears to be smaller for the population-based approaches compared with the corresponding single-solution algorithms. The exception to the above seems to be on the nott1b test instance for which the bars corresponding to the extended approaches are larger than those of the single-solution variants in most of

the cases. An interesting result observed from the figures 7.4 to 7.7 is that in some cases the extended variant of a less sophisticated algorithm outperforms the single-solution variant of another more elaborate technique. For example, in figure 7.4 it can be seen that the extended variant of the iterative improvement algorithm clearly outperforms the single-solution variant of the simulated annealing algorithm on the nott1B case. The next subsection presents and discusses the results obtained in terms of the population diversity.

7.5.3. Results on the Diversity of Solutions

Tables 7.2 and 7.3 show the diversity for the initial population (indicated V_{ip}) and the diversity of the set of solutions produced (indicated V_{fp}) by each algorithm variant on the experiments described above. Each value corresponds to the averaged (over the ten runs) percentages of the population diversity (see section 2.4.3).

		V_{fp} values obtained by the single-solution and population-based variants of each algorithm					
Test Case	V_{ip}	II_{SS}	II_{PB}	SA_{SS}	SA_{PB}	TS_{SS}	TS_{PB}
P1A	75	57.5	59.9	58.3	59.5	57.6	55.9
P1B	90	69.7	73.7	71.6	71.5	71.6	70.4
P2A	78	68.3	72.0	70.7	71.8	68.2	67.9
P2B	95	88.2	87.5	87.0	85.8	88.9	83.1
P3A	65	37.9	46.6	39.4	47.0	37.6	45.4
P3B	86	50.5	58.5	52.9	59.8	49.4	50.7
nott1A	84	68.9	74.0	72.8	74.0	71.7	77.3
nott1B	95	85.4	84.6	85.4	87.2	86.0	88.0

Table 7.2. Average diversity in the final population when the diversity of the initial population is high.

		V_{fp} values obtained by the single-solution and population-based variants of each algorithm					
Test Case	V_{ip}	II_{SS}	II_{PB}	SA_{SS}	SA_{PB}	TS_{SS}	TS_{PB}
P1A2	28	35.4	59.5	44.2	57.1	34.4	55.9
P1B2	32	48.0	71.9	38.5	73.2	46.6	70.8
P2A2	39	48.9	72.8	41.0	71.9	48.9	69.0
P2B2	34	56.2	87.0	57.7	88.4	55.3	55.0
P3A2	31	25.9	47.5	37.1	44.0	25.8	43.8
P3B2	26	29.7	58.7	41.5	58.2	39.7	56.6
nott1A2	23	40.9	73.7	42.4	72.7	51.8	76.5
nott1B2	40	37.0	87.0	54.9	85.5	44.7	87.5

Table 7.3. Average diversity in the final population when the diversity of the initial population is low.

Table 7.2 shows the results obtained when the initial population has a high diversity while table 7.3 shows the results obtained when the initial population has a low diversity. When comparing the results obtained with the two variants of each algorithm in each problem, the best diversity (highest value) is underlined to make the comparison clearer. In those cases in which the initial population is highly diverse, it is observed in table 7.2 that the population-based variants are capable of obtaining better results than the corresponding single-solution approaches in many cases. In the rest of the cases in table 7.2, the diversities of the populations produced by the extended approaches are very competitive with those of the single-solution variants. On the other hand, if the initial population has a low diversity (table 7.3), the extended approach is capable of improve upon the diversity of the initial population in some cases and although the single-solution variants also achieve a certain improvement in this respect, the diversities obtained by the extended approaches are far better in almost all cases.

7.5.4. On the Rate of Improvement

From the results presented and discussed above it is clear that the population-based variants are capable of finding a high quality and diverse set of solutions regardless of the diversity (low or high) and size (small or larger) of the initial population. This section reports on the performance of the single-solution and extended methods with respect to the computation time required to achieve the results reported above. Figure 7.8 shows typical runs for the single-solution and extended approaches over the computation time for the problem case trent1B in which the population size equals 5 and the initial population is highly diverse. These graphs show the quality of the best, average and worst individuals in the population at each time during the run. Note that since the population size is 5, the processing time shown for the extended approaches is five times longer ($5 \cdot t_{ind}$) than the processing time shown for the single-solution methods. However, as explained in section 7.5.1 the total time spent by each variant to process the whole population is the same because in each run, the single-solution method was applied to each individual in the initial population. Only the graphs of typical runs for one problem case are shown here, but similar results were observed for all problems in runs with different population sizes and different initial diversities.

Figure 7.8.b shows that the single-solution variant of iterative improvement (II_{SS}) achieves its best performance very quickly in slightly less than 20 seconds. For the population-based variant of the same algorithm (II_{PB}) comparable high quality solutions are found after 100 seconds, although further improvement and the best average are achieved after 250 seconds (figure 7.8.a). In other words, it takes about 275 seconds for the extended approach to find the best values for the best, average and worst statistics in this population of 5 solutions. For the single-solution variant it takes about 20 seconds to achieve its best statistics for each of the individuals in the population.

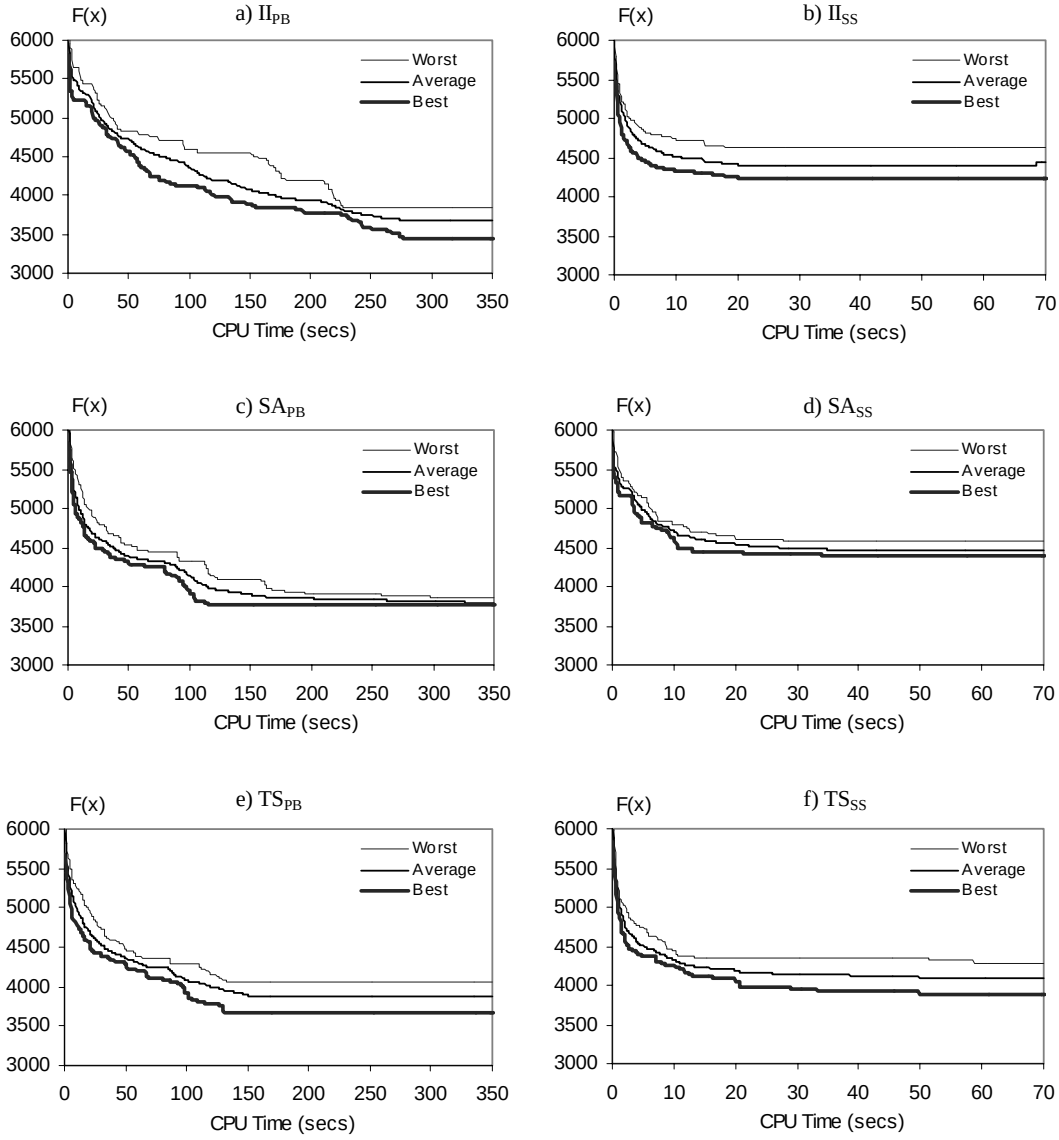


Figure 7.8. Rate of improvement over the computation execution time on the problem trent1B for each algorithm. The worst, average and best solutions for a typical run are show here.

As expected, the single-solution variant is less computationally expensive but no further improvements can be achieved. Although the population-based approach takes more time to produce a set of high quality solutions, even the worst solution found here outperforms the best result obtained with the single-solution method. Moreover, after a long processing time II_{PB} still improves the average quality of the population while II_{SS} does not produce any better result. Assuming that we need to obtain only one good solution and that we do not have much computation time available, then II_{SS} is a perfectly acceptable approach. However, if 5 good solutions are required in order to carry out comparisons and select the most appropriate one, then it will take about 20 x 5 = 100 seconds for the II_{SS} to achieve this (by re-starting the algorithm). At this time II_{PB} has already achieved a much better best solution and the average is as good (if not better) than the one produced by II_{SS}. However, it is clear that it is possible for II_{PB} to further improve the quality of the population after this computation time. Similar observations can be made for the variants of the simulated annealing and tabu search algorithms.

It could be argued that the execution times used here present an advantage for the population-based approaches, therefore experiments were carried out to run the single-solution variants for the same time as the corresponding population-based approach. For example, for problem trent1B, II_{SS} was executed for 350 seconds ($5 \cdot t_{ind}$) for each individual in the population. That is, the total time spent to obtain the set of 5 solutions was 1750 secs. These results were compared with those obtained by II_{PB} after an execution time equal to 350 seconds (the same as before). In any of the cases, the single-solution variant outperformed the extended approach even with this advantage of longer execution time. The same was observed for the three algorithms in all test problems, i.e. none of the single-solution methods achieved further improvements after longer execution times.

7.6. The Best Results for All Test Instances

A single-solution hybrid metaheuristic and population-based variants of that approach were presented and tested in chapter 5. Those population-based methods incorporated a cooperation mechanism similar to one described in section 7.4.2. to share information within the population during the search. However, that mechanism

is less elaborate and effective than the one proposed in this chapter. The focus in chapter five was on implementing a common annealing schedule to control the evolution of the whole population. This section compares the performance of the hybrid evolutionary algorithms presented in this paper and the PMHM approach described in chapter five. In addition, the single-solution hybrid metaheuristic presented in chapter five is also extended as proposed here using the scheme of figure 7.3. That is, this extended variant MH_{PB} differs from the PMHM approach with respect to the cooperation mechanism, which was enhanced in this chapter by incorporating a more 'intelligent' heavy mutation operator.

Two goals were pursued in the experiments carried out here. First, to further assess the performance of the various hybrid evolutionary metaheuristics developed in this thesis and second, to report on the best produced results for all the test instances of the space allocation problem described in section 2.5. Ten repetitions of the experiments (as described next) were carried out. For each test instance, a population of 20 initial solutions were generated using the **Allocate-Rnd-BestRnd** heuristic described and tested in chapter 4. Then, each of the population-based hybrid algorithms was applied to that initial population for $p \cdot t_{ind}$ computation time. That is, ten different initial populations were generated for each test instance and in each repetition, the same initial solutions were used for all the algorithms. After collecting all the solutions obtained by each hybrid evolutionary algorithm on each test instance, the overall best and average solutions are reported in table 7.4. The best results among all the algorithms for each of these test instances are indicated in bold. Also as a reference, table 7.4 shows the quality of the manually constructed solution for each test instance.

The first observation that can be made is that the wolver1 test problem is easily solved by all the algorithms and the solutions produced are far better than the manually contructed allocation. It is also observed that for this problem, the average solution quality obtained by the PMHM and the MH_{PB} algorithms is the same as the best solution found, i.e. these algorithms are capable of finding the best solution in all the runs. As it was noted in previous chapters, this test problem seems easy to solve because of the low number of constraints that it contains.

	nott1 $p \cdot t_{ind} = 2400$		**nott1a** $p \cdot t_{ind} = 1600$		**nott1b** $p \cdot t_{ind} = 1200$		**nott1c** $p \cdot t_{ind} = 600$		**trent1** $p \cdot t_{ind} = 1400$		**wolver1** $p \cdot t_{ind} = 500$	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver	Best	Aver	Best	Aver
Manual	599.56	-----	592.22	-----	538.44	-----	337.04	-----	3873.56	-----	1141.0	-----
II_{PB}	568.13	728.42	574.53	731.61	468.47	544.02	348.27	424.69	3439.12	3736.22	**634.19**	821.41
SA_{PB}	543.78	687.07	575.76	704.53	470.72	575.38	342.55	418.73	2724.47	3756.46	**634.19**	697.24
TS_{PB}	491.25	680.14	558.40	684.79	432.69	547.76	323.82	391.46	2682.98	3510.64	**634.19**	669.58
PMHM	525.93	647.74	540.65	693.56	458.06	505.84	334.91	378.54	3217.40	3618.78	**634.19**	634.15
MH_{PB}	**482.21**	621.56	**521.91**	648.05	**417.16**	479.50	**315.41**	392.16	**2531.41**	3104.01	**634.19**	634.13

Table 7.4. Comparing the performance of the hybrid evolutionary metaheuristics on the test instances of the space allocation problem.

For the rest of the test problems, it can be observed from table 7.4 that the new population-based variant of the hybrid metaheuristic (MH_{PB}) outperforms the previous extended version (PMHM). That is, the enhanced cooperation mechanism proposed in this chapter permits this hybrid evolutionary algorithm to produce even better results. With the exception of test instance nott1a, the population-based tabu search approach also produces better results than those obtained with the PMHM algorithm. The II_{PB} and the SA_{PB} approaches produce competitive results overall but are clearly outperformed by the other three algorithms. Only on test instance nott1c, the population-based variants of the iterative improvement and simulated annealing algortihms do not match the quality of the manually constructed solution. In the rest of the cases, all algorithms are capable of finding allocations with higher quality than the reference solution. Table 7.4 shows that, in all the test problems used in this thesis, the best solutions are also produced with the MH_{PB} algorithm, i.e. the hybrid evolutionary approach obtained from extending the single-solution hybrid metaheuristic presented in chapter 5.

7.7. Summary and Final Remarks

This chapter has reported results from a range of experiments on extending four single-solution techniques: iterative improvement, simulated annealing, tabu search and a hybrid algorithm, towards population-based approaches in order to illustrate the concept of *cooperative local search* that was proposed here. The cooperation mechanism implemented consists of adding an information sharing scheme and a heavy mutation operator that allows individuals in the population to share good and

bad parts of solutions during the evolution process. This cooperating local search scheme can be seen as an alternative to the design of elaborate recombination or repairing operators for highly constrained optimisation problems. Also, since each individual in the population uses mainly local search, no specific mechanism is required to maintain diversity (in the solution space) within the population. This way of approaching hybridisation seems to be a good alternative for improving upon the performance of other single-solution metaheuristics when a set of solutions is required. Other alternatives as discussed throughout the chapter are: designing a sophisticated version of the algorithm, fine-tuning the parameters, designing specialised heuristics and operators, hybridising using other schemes, etc.

From the experiments carried out here, it is clear that the performance of the extended versions of the four metaheuristics, when solving the set of tests instances of the space allocation problem, is better than the performance of the corresponding single-solution algorithm. It also appears that population size and diversity in the initial population does not decrease the effectiveness of the extended variants. This is an attractive feature of the scheme proposed here since other population-based approaches such as genetic algorithms usually require larger populations in order to operate or they tend to converge prematurely unless mechanisms to maintian diversity are added (Horn, 1997). Note that the implementations described here are relatively simple and not a lot of parameter tuning was necessary. However, it would be important to evaluate the effect on the sensitivity to parameter tuning of the population-based variants with respect to the original single-solution methods, but this is left for future work.

The main purpose of this chapter was to propose and illustrate the concept of cooperative local search towards the design of hybrid evolutionary metaheuristics. In addition, this chapter also justifies the effectiveness of the method by presenting the best available results on a set of test instances of the space allocation problem. It is shown that the best results overall are produced by the hybrid evolutionary metaheuristic MH_{PB} and that very competitive results are obtained with the TS_{PB} approach. The research and results presented here summarise the work carried out by the author over the last few years on the application of metaheuristics to the solution of the space allocation problems in academic institutions.

Chapter 8. Conclusions and Future Work

8.1. Conclusions

In order to draw some conclusions from the investigation presented in this thesis, it is important to consider the aims and scope that were established when this research programme was started. The overriding aim was to carry out an investigation on the suitability of applying metaheuristic techniques to tackle the space allocation problem in academic institutions. The complete construction of allocations was considered here. That is, we were concerned with allocating a set of entities into the available room space so that the space misuse and the satisfaction of soft constraints are minimised. The emphasis was in obtaining a set of high quality (i.e. not necessarily optimal) allocations that are also structurally non-similar (i.e. diverse with respect to the solution space) so that the decision-makers can select the most appropriate solution. Since very few publications in the literature have approached the space allocation problem, an additional aim here was to give a detailed description and appropriate formulation of this problem.

8.1.1. Description and Formulation of the Problem

The overall space allocation process in UK universities was well described in (Burke and Varley, 1998). The present thesis focused on the construction of allocations and this problem was described and formulated here. A metric to measure the non-similarity between allocations was proposed. This is a meaningful metric for decision-makers because it directly reflects how different the allocations are between them. Also, test data sets were prepared from real data provided by some UK universities (available from http://www.cs.nott.ac.uk/~jds/research/spacedata.html). All this work, will help researchers and practitioners to obtain a better understanding of this problem for future research in this area.

8.1.2. Design of Basic Operators

Flexible data structures based on linked lists were proposed to represent the problem instance being solved and the allocation or solution. Using this representation was beneficial in three aspects. First, the characteristics of the problem instance and the

allocation can be easily updated. Second, fast solution evaluation routines can be implemented. And third, considering the highly constrained nature of the problem, the flexibility of the data structures assisted the implementation of the local search and genetic operators. A number of heuristics for initialising solutions were designed and the best are the **AllocateRnd-BestRnd** and the **AllocateCsrt-BestRnd** heuristics, which generate sets of solutions with a good compromise between quality and diversity. Three neighbourhood structures were designed: *relocate*, *swap* and *interchange*. The heuristic ($\mathbf{H_{LS}}$) designed to choose the neighbourhood to explore takes into consideration the current status of the allocation and the history of the search. Several heuristics were designed to explore the neighbourhood structures. It was found that the best strategy (**Rnd-BestRnd**) is to choose one of the elements of the move (entity or room) at random and then to explore a subset to choose the second element of the move (entity or room). Various genetic operators were implemented including two for the recombination of solutions that were specifically designed for the space allocation problem. Even with these tailored operators, maintaining the feasibility of allocations while recombining solutions proved to be a difficult task in the space alloction problem.

8.1.3. Suitability of Metaheuristics

To the best knowledge of the author, this thesis presents the first investigation on the application of metaheuristic techniques to the space allocation problem in academic institutions. It was shown that metaheuristics can produce good solutions in much shorter time than required when constructing allocations manually. Four well-known metaheuristics were implemented in the first step of this research: iterative improvement, simulated annealing, tabu search and a genetic algorithm. The methods were reasonable adapted to the problem and benchmark results were provided. Tabu search and iterative improvement performed the best, simulated annealing produced acceptable results while the genetic algorithm exhibited a poor performance. The strong intensification feature of iterative improvement and tabu search and the memory structures for genes collection in the latter, helped these two algorithms to produce the best results among the four metaheuristics. The difficulty of recombining solutions and maintaining feasibility in this problem, contributed to the failure of the genetic algorithm which performed well only in the less constrained test instance.

8.1.4. The Hybrid Algorithms Proposed

The single-solution hybrid metaheuristic designed in this thesis, surpassed the performance of the other four methods previously implemented. Although this hybrid produced solutions of better quality than the manually constructed allocation in the test problems, this is because the obtained allocations have less space misused than the reference solutions but the satisfaction of soft constraints is higher. This confirms the difficulty of solving the space allocation problem due to the high number of constraints present. The population-based hybrid metaheuristic (extended variant of the single-solution hybrid) designed in this thesis permitted us to obtain sets of good quality allocations that are also highly diverse. It was shown that the shared memory structures and heavy mutation operator are crucial components of this approach because without them, the performance of the algorithm deteriorates considerably.

8.1.5. The Two-Objective Problem

Although multiple objectives can be considered when tackling the space allocation problem, experimental justification was provided in this thesis for approaching it as a two-objective minimisation problem. It was also shown that the hybrid algorithms developed are suitable for generating good sets of non-dominated solutions without the need to incorporate complex mechanisms to maintain diversity. From these hybrids, the PHMM algorithm produced the best non-dominated fronts.

8.1.6. Influence of Fitness Evaluation in Pareto Optimisation

The problem tackled here is highly constrained and the recombination of solutions while maintaining feasibility is difficult. The algorithms that performed well are based on the self-adaptation of solutions. Given these circumstances, it was shown that the method used to evaluate solutions during the search in Pareto optimisation has an impact on the performance of the algorithm. The aggregation of objectives and relaxed forms of dominance can be more beneficial than the standard dominance relation. This is because they allow detriments in some objectives in order to achieve improvements in others, facilitating the generation of promising candidate solutions. A tunable (for the trade-off between objectives) form of relaxed dominance which is very intuitive and simple to compute was also proposed in this thesis.

8.1.7. Cooperative Local Search

The concept of cooperative local search for the hybridisation of metaheuristics was proposed and illustrated here. It was shown that by adding elements of population-based techniques to algorithms based on local search, effective hybrid evolutionary approaches are created. A crucial element for this type of hybridisation is the design of a cooperation mechanism that permits the population of explorers to share the information gained during the search. The cooperation mechanism implemented here consisted of collecting good and bad genes (parts of solutions) in shared memory structures. Four single-solution algorithms were extended using the cooperative local search scheme and the population variants produced much better results than the single-solution methods. This hybridisation scheme is simple to implement and is particularly appropriate when the recombination of solutions requires considerable extra effort. The performance of the hybrid evolutionary approaches is not affected by the size and diversity of the initial population. For all the test data sets used in this investigation, the best known solutions are also reported which are obtained by the $\mathbf{MH_{PB}}$ hybrid evolutionary algorithm.

8.1.8. Scope of the Conclusions

Since the investigation presented in this thesis was focused on the space allocation problem in academic institutions, it should be kept in mind that the conclusions given above are within this context. However, the experiences of this study can also be beneficial for research in related areas such as space planning, shelf space allocation, academic timetabling, constrained knapsack problems, etc. Also, the algorithms described and tested here can be the starting point for the development of a fully automated system for the space allocation process (Burke and Varley, 1998).

8.2. Future Work

8.2.1. From the Space Allocation Perspective

The obvious suggested future step is the incorporation of the work presented here into a fully automated system and test it with a comprehensive range of data sets from as many different universities as possible. Another suggested step is to consider other aspects of the problem besides the complete construction of allocations. For

example, the modification of allocations given a change on the conditions of the problem (number of entities, number of rooms, constraints, etc.). It is also interesting to consider the situation in which construction work (for the modification of rooms) is required so that alternative layouts can be automatically generated.

8.2.2. From the Metaheuristics Perspective

Given the similarity of the space allocation problem with multiple knapsack problems, some heuristics proposed in the literature were tried in preliminary experiments (Abdelaziz et al., 1999; Jaszkiewicz, 2001). However, dissapointing results were obtained due to the existence of many constraints in the problem tackled here. Applying the hybrid metaheuristics developed in this thesis to problem domains which are similar to the space allocation problem, would permit us to assess their suitability and robustness. Another research direction is to compare the parameter sensitivity (also for assessing robustness) between the single-solution approaches and the extended variants. Further validation of the cooperative local search scheme can be achieved by extending other single-solution approaches based on local search (e.g. guided local search, iterated local search, variable neighbourhood search, etc.). Forms of relaxed dominance can be used to evaluate solutions in Pareto optimisation of other multiobjective combinatorial optimisation problems in order to investigate if the performance of recombinative methods can also be improved. Moreover, different fitness evaluation methods can be used to assess the fitness of different individuals within the same population. The evaluation of solutions in the population can be adapted in order to exploit the phenomenon of *global convexity* which implies that local optima can be concentrated in different small areas of the solution space (Borges and Hansen, 1998). An interesting way to evaluate solution fitness that can be investigated is *extremal optimisation* which is based on successively eliminating extremely undesirable parts of near-to-optimal solutions instead of successively improving the quality of poor initial solutions (Boettcher and Percus, 2000).

REFERENCES

Aarts, E. and Korst, J. (eds.) (1998). Simulated Annealing and Boltzman Machines. Wiley.

Aarts, E. and Lenstra, J.K. (eds.) (1997). Local Search in Combinatorial *Optimisation*. Wiley.

Abboud, N. Inuiguchi, M. Sakawa, M. and Uemura, Y. (1998). Manpower Allocation Using Genetic Annealing. *European Journal of Operational Research*, 111(3), 405-420.

Abdelaziz, F.B. Krichen, S. and Chaouachi, J. (1999). A Hybrid Heuristic for Multiobjective Knapsack Problems. In: Voss S., Martello S., Osman I.H., Roucairol C. (eds.). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, 205-212.

Abramson, D. (1991). Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms. *Management Science*, 37(1), 98-113, 1991.

Aickelin, U. and Dowsland, K.A. (2000). Exploiting Problem Structure in a Genetic Algorithm Approach to a Nurse Rostering Problem. *Journal of Scheduling*, 3(3), 139-153.

Alves, M.J. and Climaco, J. (2000). An Interactive Method for 0-1 Multiobjective Problems Using Simulated Annealing and Tabu Search. *Journal of Heuristics*, 6(3), 385-403.

Baase, S. (1998). *Computer Algorithms: Introduction to Design and Analysis*. Addison Wesley.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

Bäck, T. Fogel, D. and Michalewicz Z. (eds.) (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press.

Bagchi, T.P. (1999). *Multiobjective Scheduling By Genetic Algorithms*. Kluwer Academic Publishers.

Baykasoglu, A. Owen, S. and Gindy, N. (1999). A Taboo Search Based Approach to Find the Pareto Optimal Set in Multiple Objective Optimisation. *Engineering Optimization*, 31, 731-748.

Belton, V. and Stewart, T.J. (2002). *Multiple Criteria Decision Analysis - An Integrated Approach*. Kluwer Academic Press.

Benjamin, C. Ehie, I. and Omurtag, Y. (1992). Planning Facilities at the University of Missoury-Rolla. *Journal of Interfaces*, 22(4), 95-105.

Bentley, P.J. and Corne, D.W. (eds.) (2002). *Creative Evolutionary Systems*. Morgan Kaufmann Academic Press.

Bland, J.A. (1999). Layout of Facilities Using an Ant System Approach. *Engineering Optimization*, 32, 101-115.

Bland, J.A. (1999b). Space-Planning By Ant Colony Optimisation. *International Journal Of Computer Applications In Technology*, 12(6), 320-328.

Blum, C. and Roli, A. (2001). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *Technical Report TR/IRIDIA/2001-13, IRIDIA*, Belgium.

Boettcher, S. and Percus, A. (2000). Nature's Way of Optimizing. *Artificial Intelligence*, 119, 275-286.

Borges, P.C. and Hansen, M.P. (1998). A Basis for Future Success in Multiobjective Combinatorial Optimization. *Technical Report IMM-REP-1998-8*, Technical University of Denmark.

Bremmermann, H. (1962). Optimisation Through Evolution and Re-Combination. In: Yovits, M. Sawbi, G. and Goldstein, G. (eds), Self-Organising Systems, Washington DC, Spartan Books.

Brizuela, C. Sannomiya, N. and Zhao, Y. (2001). Multi-objective Flow-Shop: Preliminary Results. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Zurich Switzerland, Springer, 443-457.

Burke, E.K. Cowling, P. De Causmaecker, P. and Vanden Berghe, G. (2001). A Memetic Approach to the Nurse Rostering Problem. *Applied Intelligence*, 15(3), 199-214.

Burke, E.K. Cowling, P. and Landa Silva, J.D. (2001b). On the Performance of Population-Based Metaheuristics for the Space Allocation Problem: An Extended Abstract. *Proceedings of the 2001 Metaheuristics International Conference MIC 2001*, Porto Portugal, 579-583.

Burke, E.K. Cowling, P. and Landa Silva, J.D. (2001c). Hybrid Population-Based Metaheuristic Approaches for the Space Allocation Problem. *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul Korea, 232-239.

Burke, E.K. Cowling, P. Landa Silva, J.D. and McCollum, B. (2001d). Three Methods to Automate the Space Allocation Process in UK Universities. In: Burke, E.K. and Erben, W. (eds.) The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000, *Lecture Notes in Computer Science*, 2079, Springer, 254-273.

Burke, E.K. Cowling, P. Landa Silva, J.D. McCollum, B. and Varley, D. (2000a). A Computer Based System for Space Allocation Optimisation. *Proceedings of the 27th International Conference on Computers and Industrial Engineering ICC&IE 2000*, Beijing China.

Burke, E.K. Cowling, P. Landa Silva, J.D. and Petrovic, S. (2001e). Combining Hybrid Metaheuristics and Populations for the Multiobjective Optimisation of Space Allocation Problems. *Proceedings of the 2001 Genetic and Evolutionary Computation Conference GECCO 2001*, San Francisco USA, 1252-1259.

Burke, E.K. De Causmaecker, P. Petrovic, S. and Vanden Berghe, G. (2002). A Multi Criteria Meta-heuristic Approach to Nurse Scheduling. *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, Hawaii USA, 1197-1202.

Burke, E.K. Elliman, D.G. and Weare R. (1995). Specialised Recombinative Operators for Timetabling Problems. *Proceedings of the Artificial Intelligence and Simulation of Behaviour Workshop on Evolutionary Computing AISB 1995*, University of Sheffield UK, Springer, 75-85.

Burke, E.K. and Landa Silva, J.D. (2002b). Improving the Performance of Multiobjective Optimisers by Using Relaxed Dominance. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning SEAL 2002*, Singapore, 203-207.

Burke, E.K. and Landa Silva, J.D. (2003). The Influence of the Fitness Evaluation Method on the Performance of Multiobjective Optimisers. *Submitted to the European Journal of Operational Research*, February 2003.

Burke, E.K. and Landa Silva, J.D. (2003b). Hybrid Evolutionary Metaheuristics Based on Cooperative Local Search. *Submitted to the IEEE Transactions on Evolutionary Computation Journal*, March 2003.

Burke, E.K. and Newall J.P. (1999). A Multi-Stage Evolutionary Algorithm for the Timetable Problem. *IEEE Transactions on Evolutionary Computation*, 3(1), 1085-1092.

Burke, E.K. Newall, J.P. and Weare R.F. (1996). A Memetic Algorithm for University Exam Timetabling. In: Burke, E.K. and Ross, P. (eds.) The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT 1995, *Lecture Notes in Computer Science*, 1153, Springer, 241-250.

Burke, E.K. Newall, J.P. and Weare R.F. (1998). Initialisation Strategies and Diversity in Evolutionary Timetabling. *Evolutionary Computation*, 6(1), 81-103.

Burke, E.K. and Smith, A.J. (1999). A Memetic Algorithm to Schedule Planned Maintenance for the National Grid. *ACM Journal of Experimental Algorithmics*, 4(1), 1084-1054.

Burke, E.K. and Smith, A.J. (2000). Hybrid Evolutionary Techniques for the Maintenance Scheduling Problem. *IEEE Transactions on Power Systems*, 15(1), 122-128.

Burke, E.K. and Varley, D.B. (1998). Space Allocation: An Analysis of Higher Education Requirements. In: Burke, E.K. and Carter, M.W. (eds.) The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International Conference on the Practice and Theory of Autometed Timetabling PATAT 1997, *Lecture Notes in Computer Science*, 1408, Springer, 20-33.

Burke, E.K. and Varley D.B. (1998b). Automating Space Allocation in Higher Education. Selected Papers from the 2nd Asia Pacific Conference on Simulated Evolution and Learning SEAL 98, *Lectures Notes in Artificial Intelligence*,

1585, Springer, 66-73.

Calegari, P. Coray, G. Hertz, A. Kobler, D. and Kuonen, P. (1999). A Taxonomy of Evolutionary Algorithms in Combinatorial Optimization. *Journal of Heuristics*, 5(2), 145-158.

Chambers, Lance (ed.) (2001). *The Practical Handbook of Genetic Algorithms Applications*. Chapman&Hall/CRC.

Chen, W.H. and Lin, C.S. (2000). A Hybrid Heuristic to Solve a Task Allocation Problem. *Computers and Operations Research*, 27, 287-303.

Chu, P.C. and Beasley J.E. (1997). A Genetic Algorithm for the Generalised Assignment Problem. *Computers and Operations Research*, 24(1), 17-23.

Chu, P.C. and Beasley, J.E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4(1), 63-86.

Coello Coello, C.A. (1999). A Comprehensive Survey of Evolutionary Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3), 269-308.

Coello Coello, C.A. (1999a). An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends. *Proceedings of the 1999 Congress on Evolutionary Computation CEC 1999*, Washington USA, 3-13.

Coello Coello, C.A. (2000). Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32, 275-308.

Coello Coello, C.A. (2001). A Short Tutorial on Evolutionary Multiobjective Optimization. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Zurich Switzerland, Springer, 21-40.

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

Coley, D.A. (1999). *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Publishing.

Colorni, A. Dorigo, M. and Maniezzo, V. (1998). Metaheuristics for High School Timetabling. *Computational Optimization and Applications*, 9, 275-298.

Cook, S.A. (1971). The Complexity of Theorem-proving Procedures. *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, Association for Computing Machinery, New York, 151-158.

Corne, D. Ross, P. and Fang, H.L. (1994). Fast Practical Evolutionary Timetabling. *Selected Papers from the AISB Workshop on Evolutionary Computation, Lecture Notes in Computer Science*, 865, Springer, 220-263.

Corne, D. and Ross P. (1995). Some Combinatorial Landscapes on which a GA Outperforms Other Stochastic Iterative Methods. *Evolutionary Computing: Lecture Notes in Computer Science, Selected Papers of the AISB Workshop*, 993, Springer, 1-13.

Corne, D. and Ross P. (1996). Peckish Initialisation Strategies for Evolutionary Timetabling. In: Burke, E.K. and Ross, P. (eds.) The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT 1995, *Lecture Notes in Computer Science*, 1153, Springer, 227-240.

Corne, D. Dorigo, M. and Glover, F. (eds.) (1999). *New Ideas in Optimisation*. McGraw Hill.

Costa, Daniel (1994). A Tabu Search Algorithm for Computing an Operational Timetable. *European Journal of Operational Research*, 76, 98-110.

Czyzak, P. and Jaszkiewicz, A. (1998). Pareto Simulated Annealing - a Metaheuristic for Multiple-Objective Combinatorial Optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1), 34-47.

Dasgupta. P. Chakrabarti, P.P. and Desarkar, S.C. (1999). *Multiobjective Heuristic Search: An introduction to Intelligent Search Methods for Multicriteria Optimization*. Computational Intelligence – Vieweg.

Davis, Lawrence (ed.) (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

Dawande, M. Kalagnanam, J. Keskinocak, P. Ravi, R. and Salman, F.S. (2000). Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions. *Journal of Combinatorial Optimization*, 4(2), 171-186.

Deb, K. (1999). Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3), 205-230.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley.

Di Caspero, L. and Schaerf, A. (2001). Tabu Search Techniques for Examination Timetabling. In: Burke, E.K. and Erben, W. (eds.) The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000, *Lecture Notes in Computer Science*, 2079, Springer, 104-117.

Diaz, J.A. and Fernandez, E. (2001). A Tabu Search Heuristic for the Generalized Assignment Problem. *European Journal of Operational Research*, 132, 22-38.

Diminnie, C.B. and Kwak, N.K. (1986). A Hierarchical Goal-programming Approach to Reverse Resource Allocation in Institutions of Higher Learning. *Journal of the Operational Research Society*, 37, 1, 59-66.

Dorigo, M. Maniezzo, V. and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1), 1-13.

Dowsland, K.A. (1996). Simulated Annealing Solutions for Multi-Objective Scheduling and Timetabling. In: Rayward-Smith V.J., Osman I.H., Reeves C.R., Smith G.D. (eds.) *Modern Heuristic Search Methods*, John Wiley & Sons, 155-166.

Dowsland, K.A. (1998). Off-the-peg or Mafe-to-measure? Timetabling and Scheduling with SA and TS. In: Burke, E.K. and Carter, W. (eds.) The Practice and Theory of Automated Timetabling III: Selected Papers from the 2nd International Conference on the Pracatice and Theory of Automated Timetabling PATAT 1997, *Lecture Notes in Computer Science*, 1408, Springer.

Ehrgott, M. and Klamroth K. (1997). Connectedness of Efficient Solutions in Multiple Criteria Combinatorial Optimization. *European Journal of Operational Research*, 97, 159-166.

Ehrgott, M. and Gandibleux, X. (2000). A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR Spectrum*, 22(4), Springer, 425-460.

Elmohamed, M.A.S. Coddington, P. and Fox, G. (1998). A Comparison of Annealing Techniques for Academic Course Scheduling. In: Burke, E.K. and Carter, M.W. (eds.) The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT 1997, *Lecture Notes in Computer Science*, 1408, Springer, 92-112.

Erben, Wilhelm (2001). A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling. In: Burke, E.K. and Erben, W. (eds.) The Practice and

Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000, *Lecture Notes in Computer Science*, 2079, Springer, 132-156.

Erickson, M. Mayer, A. and Horn, J. (2001). The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001*, *Lecture Notes in Computer Science*, 1993, Springer, Zurich Switzerland, 681-695.

Falkenauer, E. (1994). A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems. *Evolutionary Computation*, 2(2), 123-144.

Falkenauer, E. (1996). A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics*, 2(1), 5-30.

Fonseca, C.M. and Fleming, P.J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo USA, 416-423.

Fonseca, C.M. and Fleming, P.J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1), 1-16.

Fonseca, C.M. and Fleming, P.J. (1996). On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. *Proceedings of the Parallel Problem Solving From Nature IV*, Berlin Germany, 584-593.

Fox, B.L. (1993). Integrating and Accelerating Tabu Search, Simulated Annealing and Genetic Algorithms. *Annals of Operations Research*, 41, 47-67.

Francis, R.L. McGinnis Jr., L.F. and White J.A. (1992). *Facility Layout and Location: An Analytical Approach*. Prentice-Hall.

Gandibleux, X. and Freville, A. (2000). Tabu Search Based Procedure for Solving the 0-1 MultiObjective Knapsack Problem: The Two Objectives Case. *Journal of Heuristics*, 6(3), 361-383.

Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W.H. Freeman.

Giannikos, J. El-Darzi, E. and Lees, P. (1995). An Integer Goal Programming Model to Allocate Offices to Staff in an Academic Institution. *Journal of the Operational Research Society*, 46(6), 713-720.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, 13, 533-549.

Glover, F. Kelly, J.P. and Laguna, M. (1995). Genetic Algorithms and Tabu Search: Hybrids for Optimization. *Computers and Operations Research*, 12(1), 111-134.

Glover, F.W. and Kochenberger, G.A. (eds.) (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Glover, F. Taillard, E. and De Werra, D. (1993). A User's Guide to Tabu Search. *Annals of Operations Research*, 41, 3-28.

Goicoechea, A. Hansen, D.R. and Duckstein L. (1982). *Multiobjective Decision Analysis with Engineering and Business Applications*. Wiley.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison Wesley.

Hanafi, S. Freville, A. and El Abdellaoui, A. (1996). Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem. In: Osman I.H., Kelly J.P. (eds.),

Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers, 449-465.

Hansen, P. (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Hansen, M.P. (1997). Tabu Search for Multiobjective Optimization: MOTS. *Technical Report Presented at 13th International Conference on MCDM*, Technical University of Denmark.

Hansen, P. and Mladenovic, N. (2001), Variable Neighbourhood Search: Principles and Applications. *European Journal of Operational Research*, 130(3), 449-467.

Hasan, M. AlKhamis, T. and Ali, J. (2000). A Comparison Between Simulated Annealing, Genetic Algorithm and Tabu Search Methods for the Unconstrained Quadratic Pseudo-Boolean Function. *Journal of Computers and Industrial Engineering*, 38, 323-340.

Hertz, A. and Klober, D. (2000). A Framework for the Description of Evolutionary Algorithms. *European Journal of Operational Research*, 126(1), 1-12.

Higgins, A.J. (2001). A Dynamic Tabu Search for Large-Scale Generalised Assignment Problems. *Computers and Operations Research*, 28(10), 1039-1048.

Holland, J.H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Anna Arbor.

Horn, J. Nafpliotis, N. and Goldberg, D.E. (1994). A Niched Pareto Genetic Algorithm for Multiobjective Optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE World Congress on Computational Intelligence, 1, Piscataway USA, 82-87.

Horn, J. (1997). Multicriteria Decision Making and Evolutionary Computation. In: Bäck, T. Fogel, D.B. and Michalewicz, Z. (eds.). *Handbook of Evolutionary Computation*, Institute of Physics, 1997.

Ingber, Lester (1996). Adaptive Simulated Annealing (ASA): Lessons Learned. *Control and Cybernetics*, 25(1), 33-54.

Ishibuchi, H. Murata, T. and Tomioka, S. (1997). Effectiveness of Genetic Local Search Algorithms. *Proceedings of the Seventh International Conference on Genetic Algorithms*, 505-512.

Ishibuchi, H. and Mutara, T. (1998). A Multi-Objective Genetic Local Search Algorithm and its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 28(3), 392-403.

Ishibuchi, H. Yoshida, T. and Murata, T. (2002). Selection of Initial Solutions for Local Search in Multiobjective Genetic Local Search. *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, Hawaii USA, 950-955.

Ishibuchi, H. Yoshida, T. and Murata, T. (2002a). Balance Between Genetic Search and Local Search in Hybrid Evolutionary Multi-Criterion Optimization Algorithms. *Proceedings of the 2002 Genetic and Evolutionary Conference GECCO 2002*, New York USA, 1301-1308.

Jaszkiewicz, A. (2001). Comparison of Local Search-based Metaheuristics on the Multiple Objective Knapsack Problem. *Foundations of Computing and Decision Sciences*, 26(1), 99-120.

Jaszkiewicz, A. (2002). Genetic Local Search for Multi-objective Combinatorial Optimization. *European Journal of Operational Research*, 137(1), 50-71.

Jones, D.F. Mirrazavi, S.K. and Tamiz, M. (2001). Multiobjective Meta-heuristics: An Overview of the Current State-of-the-Art. *European Journal of*

Operational Research, 137(1), 1-9.

Julstrom, B.A. (1995). What Have You Done for Me Lately? Adapting Operator Probabilities in Steady-Sate Genetic Algorithm. *Proceedings of the Sixth International Conference on Genetic Algorithms*, 81-87.

Kallarath, J. and Wilson, J.M. (1997). *Business Optimisation Using Mathematical Programming*. Macmillan.

Kellerer, H. and Pferschy, U. (1999). Cardinality Constrained Bin-packing Problems. *Annals of Operations Research*, 92, 335-348.

Kennedy, J. and Eberhart, R.C. (1999). The Particle Swarm: Social Adaptation in Information-Prcessing Systems. In: Corne, D. Dorigo, M. and Glover, F. (eds.), *New Ideas in Optimisation*, McGraw Hill.

Kim, J.G. and Kim, J.D. (1998). A Space Partitioning Method for Facility Layout Problems with Shape Constraints. *IIE Transactions*, 30, 947-957.

Kirkpatrick, S. Gelatt, C.D. and Vecchi, M.P. (1983). Optimization by Simulated Annealing, Science, 220, 671-380.

Knowles, J.D. (2001). Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization. *PhD Thesis*, Department of Computer Science, University of Reading, UK.

Knowles, J. and Corne D.W. (2000). Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2), 149-172.

Knowles, J.D. and Corne, D.W. (2000b). M-PAES A Memetic Algorithm for Multiobjective Optimization. *Proceedings of the 2000 Congress on Evolutionary Computation CEC 2000*, Piscataway USA, 325-332.

Knowles, J. and Corne, D. (2002). On Metrics for Comparing Nondominated Sets. *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, Hawaii USA, IEEE Press, 711-716.

Knowles, J.D. Watson, R.A. and Corne, D.W. (2001). Reducing Local Optima in Single-Objective Problems by Multi-objectivization. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Springer, 269-283.

Kokolo, I. Hajime, K. and Shigenobu, K. (2001). Failure of Pareto-based MOEAs, Does Non-dominated Really Mean Near to Optimal?. *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul Korea, 957-962.

Kumar, R. and Rockett, P. (2002). Improved Sampling of the Pareto-Front in Multiobjective Genetic Optimizations by Steady-State Evolution: A Pareto Converging Genetic Algorithm. *Evolutionary Computation*, 10(3), 283-314.

Kusiak, A. (2000). *Computational Intelligence in Design and Manufacturing*. Wiley.

Laguna, M. (2002). Scatter Search. In: P. M. Pardalos and M. G. C. Resende (eds.) *Handbook of Applied Optimization*, Oxford University Press, 183-193.

Larson, N. and Kusiak, A. (1995). Work-in-progress Space Allocation: a Model and an Industrial Application. *IIE Transactions*, 27, 497-506.

Laumanns, M. Zitzler, E. and Thiele, L. (2001). On the Effects of Archiving, Elitism, and Density Based Selection in Multi-objective Optimization. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Zurich Switzerland, Springer, 281-196.

Laumanns, M. Thiele, L. Deb, K. and Zitzler, E. (2002). Combining Convergence and Diversity in Evolutionary Multiobjective Optimization. *Evolutionary Computation*, 10(3), 263-282.

Liggett, R.S. (2000). Automated Facilities Layout: Past, Present and Future. *Automation in Construction*, 9, 197-215.

Liu, J. (1999). The Impact of Neighbourhood Size on the Process of Simulated Annealing: Computational Experiments on the Flowshop Scheduling Problem. *Computers & Industrial Engineering*, 37(1-2), 285-288.

Man, K.F. Tang, K.S. and Kwong, S. (1999). *Genetic Algorithms: Concepts and Design*. Springer.

Marett, R. and Wright, M. (1996). A Comparison of Neighbourhood Search Techniques for Multi-Objective Combinatorial Problems. *Computers and Operations Research*, 23(5), 465-483.

Martello, S. and Toth, P. (1990). *Knapsack Problems - Algorithms and Computer Implementations*. Wiley.

Menczer, F. Degeratu, M. and Street, W.N. (2000). Efficient and Scalable Pareto Optimization by Evolutionary Local Selection Algorithms. *Evolutionary Computation*, 8(2), 223-247.

Metropolis, N. Rosenbluth A.W., Rosenbluth, M.N., Teller A.H. and Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21(6), 1087-1092.

Michalewicz, Zbigniew (1999). *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd. Ed., Springer.

Michalewicz, Z. and Fogel, D.B. (2000). *How to Solve It: Modern Heuristics*. Springer.

Miettinen, K. (2001). Some Methods for Nonlinear Multi-Objective Optimization. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993,

Zurich Switzerland, Springer, 1-20.

Mlandenovic, N. and Hansen, P. (1997). Variable Neighbourhood Search. *Computers and Operations Research*, 24(11), 1097-1100.

Morrison, R.W. and De Jong, K.A. (2001). Measurement of Population Diversity. *Artificial Evolution: Selected Papers of the 5th International Conference on Artificial Evolution EA 2001, Lecture Notes in Computer Science*, 2310, Le Creusot France, Springer, 31-41.

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena CA, USA.

Moscato, P. (1999). Memetic Algorithms: A Short Introduction. In: Corne, D. Dorigo, M. and Glover, F. (eds.), *New Ideas in Optimisation*, McGraw Hill.

Moscato, P. and Cotta, C. (2003). A Gentle Introduction to Memetic Algorithms. In: Glover, F.W. and Kochenberger, G.A. (eds.), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Murata, T. Ishibuchi, H. and Tanaka, H. (1996). Multi-Objective Genetic Algorithm and its Applications to Flowshop Scheduling. *Computers and Industrial Engineering*, 30(4), 957-968.

Murata, T. Ishibuchi, H. and Tanaka, H. (1996b). Genetic Algorithms for Flowshop *Scheduling Problems. Computers and Industrial Engineering*, 30(4), 1061-1071.

Murata, T. Ishibuchi, H. and Gen, M. (2000). Cellular Genetic Local Search for Multi-Objective Optimization. *Proceedings of the 2000 Genetic and Evolutionary Computation Conference GECCO 2000*, 307-314.

Murata, T. Ishibuchi, H. and Gen, M. (2001). Specification of Genetic Search

Directions in Cellular Multi-objective Genetic Algorithms. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Zurich Switzerland, Springer, 82-95.

Nagar, A. Haddock, J. and Heragu, S. (1995). Multiple and Bicriteria Scheduling: A Literature Survey. *European Journal of Operational Research*, 81, 88-104.

Osman, I.H. (1995). Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches. *OR Spektrum*, 17, Springer, 211-225.

Osman, I.H. and Kelly J.P. (eds.) (1996). *Meta-Heuristics: Theory & Applications*. Kluwer Academic Publishers.

Osman, I.H. and Laporte, G. (1996). Metaheuristics: A Bibliography. *Annals of Operations Research*, 63, 513-623.

Papadimitriou, C.H. and Steiglitz, K. (1999). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications.

Pirlot, M. (1996). General Local Search Methods. *European Journal of Operational Research*, 92(3), 493-511.

Poole, D. Mackworth, A. and Goebel, R. (1998). *Computational Intelligence - A Logical Approach*. Oxford University Press.

Preux, Ph. and Talbi, E.G. (1999). Towards Hybrid Evolutionary Algorithms. *International Transactions in Operational Research*, 6, 557-570.

Purshouse, R.C. and Fleming, P.J. (2001). The Multiobjective Genetic Algorithm Applied to Benchmark Problems - An Analysis. *Technical Report No. 796*, Department of Automatic Control and Systems Engineering, University of Sheffield, UK.

Randall, M. and Abramson, D. (2001). A General Meta-Heuristic Based Solver for Combinatorial Optimisation Problems. *Computational Optimization and Applications*, 20, 185-210.

Rayward-Smith, V.J. (1986). *A First Course in Computability*. Blackwell.

Rayward-Smith, V.J. Osman, I.H. Reeves, C.R. and Smith, G.D. (eds.) (1996). *Modern Heuristic Search Methods*. Wiley.

Reeves, C.R. (ed.) (1995). *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill.

Reeves, C. (1996). Hybrid Genetic Algorithms for Bin-packing and Related Problems. *Annals Of Operations Research*, 63, 371-396.

Reeves, C. (1996b). Integrating Local Search into Genetic Algorithms. In: Rayward-Smith V.J., Osman I.H., Reeves C.R., Smith G.D. (eds.), *Modern Heuristic Search Methods*, John Wiley & Sons.

Reynolds, R.G. (1999) . Cultural Algorithms: Theory and Applications. In: Corne, D. Dorigo, M. and Glover, F. (eds.), *New Ideas in Optimisation*, McGraw Hill.

Ritzman, L. Bradford, J. and Jacobs, R. (1980). A Multiple Objective Approach to Space Planning for Academic Facilities. *Journal of Management Science*, 25(9), 895-906.

Romero, D. and Sanchez-Flores, A. (1990). Methods for the One-dimensional Space Allocation Problem. *Computers and Operations Research*, 15(5), 465-473.

Rosenthal, Richard E. (1985). Principles of Multiobjective Optimization. *Decision Sciences*, 16, 133-152.

Salman, F.S. Kalagnaman, J.R. Murthy, S. and Davenport A. (2002). Cooperative Strategies for Solving Bicriteria Sparse Multiple Knapsack Problem. *Journal of*

Heuristics, 8, 215-239.

Schaerf, A. (1999). A Survey of Automated Timetabling, *Artificial Intelligence Review*, 13, 87-127.

Schaerf, A. (1999b). Local Search Techniques for Large High School Timetabling Problems. *IEEE Transactions on Systems, Man and Cybernetics- Part A: Systems and Humans*, 29(4), 368-377.

Schaffer, J.D. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, 93-100.

Serafini, Paolo (1992). Simulated Annealing for Multiobjective Optimization Problems. *Procceedings of the 10th International Conference on Multiple Criteria Decision Making*, Taipei Taiwan, 87-96.

Srinivas, N. and Deb, K. (1995). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3), 221-248.

Steuer, Ralph E. (1986). *Multiple Criteria Optimization: Theory, Computation and Application*. Wiley.

Strenski, P.N. and Kirkpatrick, S. (1991). Analysis of Finite Length Annealing Schedules. *Algorithmica*, 6, Springer, 346-366.

Suppapitnarm, A. Seffen, A. Parks, G.T. and Clarkson P.J. (2000). A Simulated Annealing Algorithm for Multiobjective Optimisation, *Engineering Optimization*, 33(1), 59-85.

T'kindt, V. and Billaut, J.C. (2002). *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer.

Taillard, E.D. Gambardella, L.M. Gendreau, M. and Potvin, J. (2001). Adaptive

Memory Programming: A Unified View of Metaheuristics. *European Journal of Operational Research*, 135, 1-16.

Talbi, E.G. (2002). A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8, 541-564.

Tan, K.C. Lee, T.H. and Khor E.F. (2001). Evolutionary Algorithms for Multi-Objective Optimization: Performance Assessments and Comparisons. *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul Korea, 979-986.

Thiel, J. and Voss, S. (1994). Some Experiences on Solving Multiconstraint Zero-one Knapsack Problems with Genetic Algorithms. *INFOR*, 32(4), 226-242.

Thompson, J.M. and Dowsland, K.A. (1996). General Cooling Schedules for a Simulated Annealing Based Timetabling System. In: Burke, E.K. and Ross, P. (eds.) The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT 1995, *Lecture Notes in Computer Science*, 1153, Springer, 345-363.

Thompson, J.M. and Dowsland, K.A. (1996b). Variants of Simulated Annealing for the Examination Timetabling Problem. *Annals of Operations Research*, 63, 105-128.

Tuson, A. and Ross, P. (1998). Adapting Operator Settings in Genetic Algorithms. *Evolutionary Computation*, 6(2), 161-184.

Ulungu, E.L. and Teghem, J. (1994). Multi-objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis*, 3, 83-104.

Ulungu, E.L. Teghem, J. Fortemps, P.H. and Tuyttens, D. (1999). MOSA Method: A Tool for Solving Multiobjective Combinatorial Optimization Problems. *Journal of Multicriteria Decision Analysis*, 8, 221-236.

Vaessens, R.J.M. Aarts, E.H.L. and Lenstra, J.K. (1998). A Local Search Template. *Computers and Operations Research*, 25(11), 969-979.

Van Veldhuizen, D.A. and Lamont, G.B. (2000). Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 8(2), 125-147.

Van Veldhuizen, D.A. and Lamont, G.B. (2000b). On Measuring Multiobjective Evolutionary Algorithms Performance. *Proceedings of the 2000 Congress on Evolutionary Computation CEC 2000*, Piscataway USA, 204-211.

Varela, R. Vela, C.R. Puente, J. Gomez, A. and Vidal, A.M. (2001). Solving Job-shop Scheduling Problems by Means of Genetic Algorithms. In: Chambers, L. (ed.) *The Practical Handbook of Genetic Algorithms Applications*, Chapman&Hall/CRC, 275-294

Vasquez, M. and Hao, J.K. (2001). A "Logic-Constrained" Knapsack Formulation and a Tabu Search Algorithm for the Daily Photograph Scheduling of an Earth Observation Stellite. *Computational Optimization and Applications*, 20(2), 137-157.

Voss, S. Martello, S. Osman, I.H. and Roucairol C. (eds.) (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers.

White, G.M. and Xie, B.S. (2001). Examination Timetables and Tabu Search with Longer-Term Memory. In: Burke, E.K. and Erben, W. (eds.) The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000, *Lecture Notes in Computer Science*, 2079, Springer, 85-103.

Wolpert, D. and Macready, W. (1995). No Free Lunch Theorems for Search.

Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, USA.

Wolpert, D. and Macready, W. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67-82.

Wren, A. (1996). Scheduling, Timetabling and Rostering, a Special Relationship?. In: Burke, E.K., and Ross, P. (eds.) The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT 1995, *Lecture Notes in Computer Science*, 1153, Springer, 46-75.

Wright, M.B. and Marett, R.C. (1996). A Preliminary Investigation into the Performance of Heuristic Search Methods Applied to Compound Combinatorial Problems. In: Osman I.H., Kelly J.P. (eds.), *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers, 299-317.

Yamada, T. and Futakawa, M. (1997). Heuristic and Reduction Algorithms for the Knapsack Sharing Problem. *Computers and Operations Research*, 24(10), 961-967.

Yang, M.H. and Chen, W.C. (1999). A Study on Shelf Space Allocation and Management. *International Journal of Production Economics*, 60-61, 309-317.

Yang, M.H. (2001). An Efficient Algorithm to Allocate Shelf Space. *European Journal of Operational Research*, 131, 107-118.

Youssef, H. Sait, S.M. and Adiche, H. (2001). Evolutionary Algorithms, Simulated Annealing and Tabu Search: A Comparative Study. *Engineering Applications of Artificial Intelligence*, 14, 167-181.

Zitzler, E. (1999). Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. *PhD Thesis*, The Swiss Federal Institute of Technology Zurich Switzerland, Shaker Verlag.

Zitzler, E. Deb, K. and Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2), 173-195.

Zitzler, E. Laumanns, M. and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. *Proceedings of the EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, Barcelona Spain.

Zitzler, E. and Thiele, L. (1998). Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. *Proceedings of the Parallel, Problem Solving From Nature PPSN V*, 292-301.

Zitzler, E. and Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4), 257-271.

Zufryden, F.S. (1986). A Dynamic Programming Approach For Production Selection And Supermarket Shelf-Space Allocation. *Journal of the Operational Research Society*, 37(4), 413-422.

Zydallis, J.B. Van Veldhuizen, D.A. and Lamont, G.B. (2001). A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA-II. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes in Computer Science*, 1993, Zurich Switzerland, Springer, 226-240.

APPENDIX – List of Publications

[Bur2000] Burke, E.K. Cowling, P. Landa Silva, J.D. McCollum, B. and Varley, D. (2000). A Computer Based System for Space Allocation Optimisation. *Proceedings of the 27th International Conference on Computers and Industrial Engineering (ICC&IE 2000)*, Beijing China, China Machine Press, ISBN 7-900043-38-1, 11-13.

[Bur2001] Burke, E.K. Cowling, P. Landa Silva, J.D. and McCollum, B. (2001). Three Methods to Automate the Space Allocation Process in UK Universities. *The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000), Lecture Notes in Computer Science*, 2079, Konstanz Germany, Springer, 254-273.

[Bur2001b] Burke, E.K. Cowling, P. and Landa Silva, J.D. (2001). Hybrid Population-Based Metaheuristic Approaches for the Space Allocation Problem. *Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001)*, Seoul Korea, IEEE Press, 232-239.

[Bur2001c] Burke, E.K. Cowling, P. and Landa Silva, J.D. (2001). On the Performance of Population-Based Metaheuristics for the Space Allocation Problem: An Extended Abstract. *Proceedings of the 2001 Metaheuristics International Conference (MIC 2001)*, Porto Portugal, 579-583.

[Bur2001d] Burke, E.K. Cowling, P. Landa Silva, J.D. and Petrovic, S. (2001). Combining Hybrid Metaheuristics and Populations for the Multiobjective Optimisation of Space Allocation Problems. *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO 2001)*, San Francisco USA, Morgan Kaufmann, 1252-1259.

[Bur2002] Burke, E.K. and Landa Silva, J.D. (2002). Improving the Performance of Multiobjective Optimisers by Using Relaxed Dominance. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002)*, Singapore, ISBN 981-04-7523-3, 203-207.

[Bur2003] Burke, E.K. and Landa Silva, J.D. (2003). The Influence of the Fitness Evaluation Method on the Performance of Multiobjective Optimisers. *Submitted to the European Journal of Operational Research*, February 2003.

[Bur2003b] Burke, E.K. and Landa Silva, J.D. (2003). Hybrid Evolutionary Metaheuristics Based on Cooperative Local Search. *Submitted to the IEEE Transactions on Evolutionary Computation Journal*, March 2003.