HYBRID POPULATION-BASED METAHEURISTIC APPROACHES FOR THE SPACE ALLOCATION PROBLEM

E. K. Burke[1], P. Cowling[2], J.D. Landa Silva[3] †
Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science and IT, Jubilee Campus, University of Nottingham
Nottingham, NG8 1BB, UK
[1]ekb@cs.nott.ac.uk,[2]pic@cs.nott.ac.uk,[3]jds@cs.nott.ac.uk
http://www.asap.cs.nott.ac.uk

ABSTRACT.
A hybrid population-based metaheuristic for the space allocation problem in academic institutions is presented that is based upon previous experiments using a range of techniques including hill-climbing, simulated annealing, tabu search and genetic algorithms. The proposed approach incorporates the best characteristics of each technique, makes an automatic selection of the parameters according to the problem characteristics and surpasses the performance of these standard techniques in terms of the solution quality evaluated with a penalty function. This approach incorporates local search heuristics, adaptive cooling schedules and population-based techniques. Our experiments show that this technique produces competitive solutions for the space allocation problem. In this problem, it is often desirable to obtain a set of candidate solutions so that the decision maker can select the best among them. By controlling a common cooling schedule for the whole population in the simulated annealing component, it is possible to find one excellent solution or to produce a population of good solutions.

1 Introduction

Space allocation problems are a class of complex combinatorial optimisation problems, which can be defined as follows: *the distribution of the available areas of space among a number of objects with different sizes so as to ensure the optimal space utilization and the satisfaction of additional requirements and/or constraints* [5].

An investigation of the space allocation problem is being carried out in the Automated Scheduling, Optimisation and Planning Research Group (ASAP) at the University of Nottingham in the UK. The aim is to develop a comprehensive study of this problem to construct a complete model and to produce a robust framework for obtaining high quality solutions. A space allocation questionnaire was sent to 96 universities in the UK in 1996. An analysis of these results which highlights the complexity, diversity and scope of the space allocation problem was presented in [3]. In [5], we investigate the neighbourhood exploration in this problem using three techniques: hill-climbing, simulated annealing and a genetic algorithm. An initial prototype of the space allocation software was presented in [6]. Here, we propose a hybrid population-based metaheuristic that produces competitive results for space allocation and that might also be successful in other combinatorial optimisation problems.

In the proposed technique, we combine local heuristics (such as hill-climbing), adaptive cooling schedules in simulated annealing, tabu lists, list of favourable moves and mutation operators. The parameters for the hybrid metaheuristic are automatically selected depending on the characteristics of the specific space allocation problem. The incorporation of a population in the simulated annealing component provides interesting results since it is possible to adapt the cooling schedule to favour either the generation of one good quality solution in short time or a set of good quality individuals at the expense of more computation time.

In section 2 we give a brief description of the space allocation problem. Some real-world instances arising in academic institutions that we use as test problems are also presented. In section 3, we describe the hybrid metaheuristic and provide comparative results for our test problems. In section 4 we analyse the effect of incorporating a population in order to produce an improved approach. In section 4, we also analyse the control of the cooling schedule in the simulated annealing component of the hybrid population-based metaheuristic to find one single high quality allocation or a set of very competitive solutions. In section 5 some additional comments and conclusions are presented.

2 The Space Allocation Problem

2.1 A Model of the Problem

We have a set of n objects and a set of *m* available areas of space, where each object $O_1, O_2, ..., O_n$ has a size given by $S_1, S_2, ..., S_n$ respectively, and each area of space $A_1, A_2, ..., A_m$ has a capacity given by $C_1, C_2, \ldots, C_m$ respectively. S_{max} is the size of the largest object(s) and C_{max} is the capacity of the largest area(s) of available space. The total area of space required to allocate all the objects and the total area of space available are given by respectively:

$$S_T = \sum_{i=0}^{n} S_i \qquad\qquad C_T = \sum_{i=0}^{m} C_i$$

† The author acknowledges support from Universidad Autonoma de Chihuahua and PROMEP in Mexico.

Note that the size of the objects may be changed, area can be added or removed, two or more areas may be merged and the available area may be changed through reconstruction[‡]. Then the problem is to find the optimal allocation of the n objects into the m available areas of space, given by $f: O \to A$, where

$f(O_i) = A_j$, if the object O_i has been allocated to the area A_j
$f(O_i) = 0$, if the object O_i has not been allocated

This should maximize the k objectives $g_1, g_2, g_3, \ldots, g_k$ subject to z constraints R_i for $i = 1,2,\ldots,z$, where each R_i is a specific constraint that affects either the location of one or more objects, the utilisation of a certain area(s) of space or both. The types of constraints or restrictions include but are not limited to the following:

- O_i together with O_j , i.e. both objects should be allocated in the same area
- O_i adjacent to O_j , i.e. the objects should be allocated in adjacent areas
- O_i grouped with O_j , i.e. the objects should be allocated close each other
- O_i not sharing, i.e. no other object should be sharing the same area of space with the object O_i
- O_i away from O_j , i.e. these objects should be allocated far away from each other
- O_i located in A_j , i.e. this object should be allocated in the specific area A_j

The mapping function $f: O \to A$ is not usually bijective, e.g. more than one object might be allocated to the same area of space if there is enough available space and no hard constraint is violated. Each objective g_i for $i = 1,2,\ldots,k$ is a measure of the solution's fitness according to a specific evaluation criterion. For example, we may have the sum of penalties associated with the violated constraints. Other objective functions could be defined for each specific instance of the space allocation problem to evaluate costs, profits, disturbance, functionality, etc.

2.2 Real-World Space Allocation Problem

In the context of academic facilities, the space allocation problem is the allocation of resources (staff, students, meeting rooms, lecture rooms, special rooms, etc.) to areas of space such as rooms, satisfying as many requirements/constraints as possible. In general, the real instances of the space allocation problem in universities can be classified as one of the following: reorganisation of the existing allocation or construction of a complete solution.

Reorganisation is defined as the rearrangement of the current space distribution among the objects and it is performed when either there is a requirement to improve the existing solution or the specifications (constraints, number of objects to be allocated, number of areas of space, size of the objects, capacities of the areas, etc.) of the problem have been modified. The construction of a complete allocation is defined as the generation of a solution to distribute all the available areas of space among all the objects in the problem. In this paper we consider the construction of complete solutions. Some other similar or related real-world problems and the techniques used to tackle them can be found in [1], [2], [8] and [11].

2.3 Construction of an Allocation

In many universities there exists 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.

The manual process to construct a complete allocation in academic institutions can be briefly described as follows. The space necessary for each resource, the available space in rooms, the constraints that must be satisfied (hard constraints) and those that are desirable to satisfy (soft constraints) are determined. With the aid of floor plans and room databases, information about the available areas of space is obtained (size, location, proximity, etc.). Resources are allocated to rooms in order of importance according to the specific situation. The satisfaction of space requirements and constraints is verified each time a resource 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. The most common evaluation criteria are: number of allocated resources, efficient space utilisation, satisfaction of constraints and user satisfaction. Due to the nature of this manual process, it is common that weeks or months are necessary to obtain a final solution. Additional information regarding the space allocation process in British universities in available in [3].

2.4 Evaluation of a Solution

In this paper we use the combination of the multiple criteria into a single criterion using the aggregating penalty function (1) to calculate the total penalty and hence to measure the allocation's fitness.

$$TP = \sum_{i=1}^{n} UP(O_i) + \sum_{i=1}^{m} [WP(A_i) + OP(A_i)] + \sum_{i=1}^{n} DP(O_i) + \sum_{i=1}^{n} SCP(O_i) \qquad (1)$$

In this function, TP is the total penalty in the solution, UP is the penalty for object (resource) O_i if not allocated, WP is the penalty for area (room) A_i if there is space wastage, OP is the penalty for area (room) A_i if there is space overuse, DP is the penalty for changing the allocation of object (resource) O_i in a reorganisation problem, SCP is

[‡] Recostruction refers to building work to alter existing rooms.

the penalty for violating a soft constraint for object (resource) O_i.

Given this penalty function, the fitness or quality of an allocation increases as the total penalty decreases. We wil use the total penalty for comparison among solutions. The test problems used in our experiments are described in the next section and all refer to the construction of completely new allocations. A feasible solution must have all the resources allocated and all hard constraints satisfied. This means that the penalties due to unallocated resources and disturbance must be equal to zero and have no effect in the total penalty. An optimal solution is a feasible allocation that minimizes the total penalty. Of course, it is usually not possible to satisfy all of the soft constraints. In a real space allocation problem, more that one optimal solution may exist.

2.5 The Solution Structure

We use the structure shown in Fig. 1 to represent an allocation in our algorithms. It is a string-to-string mapping where the key is the resource to be allocated and the value associated to each key is the room in which the resource is allocated (bin if unallocated). The constraints (hard and soft) are linked to the corresponding resource or room to which they apply. A binary constraint is assigned to the first resource or room following its description. For example, the constraint: A adjacent to B is assigned to the object A.

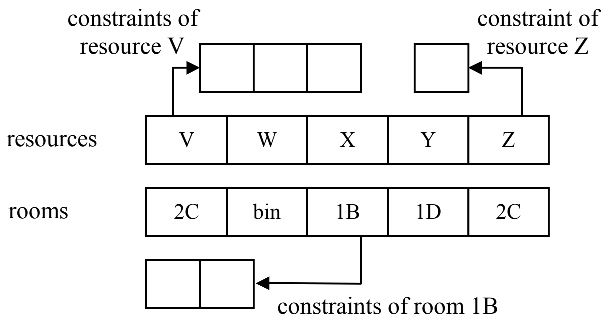

Fig. 1. Structure of a candidate solution.

2.6 Test Problems

Using real data from the School of Computer Science and IT at the University of Nottingham, the following test problems (available in [4]) were used in our experiments:

Problem 1. In this problem, there are 55 available rooms of different sizes and 55 resources to be allocated. The resources are distributed according to their level, indicating sharing and space requirements: 6 professors, 1 reader, 5 senior lecturers, 25 lecturers, 10 secretaries and 8 technical staff. There are 7 hard constraints and 18 soft constraints.

Problem 2. There are 93 resources to be allocated in 55 available rooms. The resources are distributed as follows: 9 common rooms, 11 laboratories, 12 meeting rooms, 6 storage rooms, 6 professors, 1 reader, 5 senior lecturers, 25 lecturers, 10 secretaries and 8 technical staff. There are 23 hard constraints and 38 soft constraints.

Problem 3. This problem consists of 115 resources and 115 available rooms for the allocation. The resources are: 5 research staff, 1 teaching assistant, 16 research areas, 9 common rooms, 11 laboratories, 12 meeting rooms, 6 storage rooms, 6 professors, 1 reader, 5 senior lecturers, 25 lecturers, 10 secretaries and 8 technical staff. There are 32 hard constraints and 35 soft constraints.

In our experiments, we executed 30 runs of the approaches described in this paper for each test problem, but here we only present the best results obtained. We also discuss the variation between runs in the following sections.

3 Designing a Hybrid Metaheuristic

3.1 Previous Work

Some of the first attempts to automate the space allocation process used linear programming techniques, for example Giannikos et. al. designed an integer goal-programming model to distribute the offices in an academic institution [8]. Reeves proposed the combination of Genetic Algorithms with simple heuristics to solve combinatorial problems with specific attention to the bin-packing problem [10]. More recently, Burke et. al. explored the use of Hill-Climbing, Simulated Annealing and a Genetic Algorithm to solve the space allocation problem in universities [5]. In that paper, we investigated the effect of the local search heuristic (for neighbourhood exploration) on the performance of these three methods when applied to some test instances.

3.2 Conceiving the Hybrid Metaheuristic

3.2.1 The Hybridisation

Osman and Kelly [9] define a metaheuristic as "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently high quality solutions". Our proposed hybrid metaheuristic consists of the following components:

- **Heuristic_Hill_Climbing** to initialise the solution and achieve a certain level of quality in the initial allocation.
- **Simulated_Annealing** with reheating in order to improve the initial solution produced by the heuristic hill-climbing algorithm and avoid the local optima by exploring different areas of the solution space.
- **Heuristic_Feasible_Move_Selection** for selecting the neighbourhood to be explored and in consequence the types of move to be attempted while improving the current solution. A move is a modification in the current allocation to obtain a different solution.
- **Heuristic_Parameters_Selection** to select the algorithm parameters according to the problem size. This heuristic might not produce the optimal parameter values for each problem, but will find a good set of parameters in general.

- **Mutation_Operator** to modifiy the current solution by removing some resources from the current allocation to achieve a better exploration of the solution space.

The pseudocode for the hybrid metaheuristic is shown below. Each part of this algorithm is described in more detail in the subsequent sections.

```
1. An initial feasible current solution is constructed
2. Heuristic_Parameters_Selection according to the problem size
/***** Heuristic_Hill_Climbing component *****/
3. For iterations = 1 to HCC_Iterations
    a. Using Heuristic_Feasible_Move_Selection, search for a feasible
       move to modify the current solution
    b. If a feasible move was found
        i. Accept the feasible move if it improves the current solution,
           otherwise the move is rejected
    c. If no feasible move was found, increment the counter of
       Failed_Move_Attempts
4. Make best solution = current solution
/***** Simulated_Annealing component *****/
5. While Termination_Criterion not satisfied
    a. Using the control cooling schedule, the temperature is established
    b. Using Heuristic_Feasible_Move_Selection, search for a feasible
       move to modify the current solution
        i. If a feasible move was found and it improves the current
           solution
            1. Accept the move
            2. If the current solution is better than the best solution,
               make best solution = current solution
        ii. If the found feasible move worsens the current solution and the
            temperature is not zero
            1. Random acceptance of the feasible move
        iii. If no feasible move was found
            1. Increment the counter Failed_Move_Attempts
            2. If Failed_Move_Attempts > Max_Failed_Attempts
               implement the Mutation_Operator to modify the current
               solution
```

3.2.2 *Parameter Selection*

The component Heuristic_Parameters_Selection examines the problem data and sets up those parameters that are necessary in the subsequent components. The temperature parameters for the Simulated_Annealing component are set as follows. The maximum temperature is set to the value in which the probability of accepting a non-improving move is between 95% and 100%. This acceptance probability depends on the value of the variation produced in the solution fitness by the attempted move. Then, the maximum temperature is a consequence of the type and number of constraints in the problem and the associated penalties. For the test problems considered here, the maximum temperature is set to 1000. The decrement in the temperature during the cooling process is set to one fifth of the maximum temperature. The number of iterations after which the temperature is decremented is equal to the total number n of resources to be allocated. Once the temperature is equal to zero (the process is cooled), the current temperature is again set to be equal to the maximum temperature if after a certain number of iterations no improvement has been achieved in the solution. This number of iterations (called the reheat interval) is set to 10 times the total number n of resources to be allocated. The number of iterations HHC_Iterations for the Heuristic_Hill_Climbing phase is set to be equal to the reheat interval parameter mentioned above. The value for Max_Failed _Attempts is set to one fifth of the total number of resources. To establish the termination criterion we note that in the majority of the runs with our test problems, the best performance of the hybrid metaheuristic is achieved if, after three times the reheat interval, no improvement has been obtained in the best solution. In our previous work [5] we examined the sensitivity of the hill-climbing and simulated annealing algorithms to these parameters. Since our hybrid metaheuristic is based on these two techniques, we used the values that produce the best results for this problem.

3.2.3 *Selection of a Feasible Move*

The pseudocode for Heuristic_Feasible_Move_Selection is shown below. This process uses the parameters calculated in Heuristic_Parameters_Selection, the state of the current solution and a list of tabu moves.

```
If Not All Resources are Allocated
    If Failed_Move_Attempts < Max_Failed_Attempts
        If previous Move_Type is not ALLOCATE
            Move_Type = Select_Move
    Else
        If Previous Move_Type is ALLOCATE
            Move_Type = Select_Move
        Else
            Move_Type = ALLOCATE
            Failed_Attempts_Counter = 0
Else
    Feasible_Move = Select_Move
feasible move = Find_Move (Move_type)
```

Basically, this module selects the type of move taking into account the number of allocated resources in the solution and the number of failed attempts to implement a specific type of move. Select_Move performs a random selection of the type of move among RELOCATE (relocation of an object), INTERCHANGE (interchange locations between two objects) and SWAP (swapping allocated objects between two areas of space). Once the type of move is determined, Find_Move chooses a feasible move (a move that keeps the current solution as feasible) of the selected type from the solution space. Find_Move uses a tabu list that keeps record of the unsuccessful attempted moves. The size of this list is fixed to 20 moves in our test problems. When the tabu list is full, the last unsuccessful move replaces the move that has been in the list for the greatest number of iterations. The selection of the move in Find_Move is done according to the type of move:

- ALLOCATE. An unallocated object is chosen and an area of space is selected to allocate this object.
- RELOCATE. An allocated object is chosen and an area of space is selected to change the location of the object.
- INTERCHANGE. Two objects are selected and their assigned areas of space are interchanged.

- SWAP. Two areas of space are selected and all the allocated objects in one of the areas of space are allocated to the other and vice versa.

3.2.4 Hill-Climbing Initialisation

The Heuristic_Hill_Climbing component takes the initial solution and constructs an improved feasible allocation using the Heuristic_Feasible_Move_Selection component explained in section 3.2.3. 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 hybrid metaheuristic.

3.2.5 Simulated Annealing Component

The Simulated_Annealing phase takes the improved feasible current solution obtained in Heuristic_Hill_Climbing and uses the cooling schedule and Heuristic_Move_Selection to explore the solution space and find a better allocation. Even if the quality of the current solution is decreased in this process, the best candidate solution is always maintained. The cooling schedule initializes the current temperature with a value equal to the maximum temperature and decrements it after a number of iterations. When the current temperature is equal to zero, the cooling schedule maintains this value while the searching process produces improvements in the best solution. If after a number of iterations (reheat interval), no improvement is possible in the best solution, the current temperature is again set to be equal to the maximum temperature. The values for the parameters in the Simulated_Annealing phase were defined in section 3.2.2.

3.2.6 Mutation Operator

The mutation operator disrupts the current solution after a number of failed attempts to find a feasible move. The disruption consists of removing from their assigned room, those allocated resources that contribute the highest penalty. This releases the area of space assigned to those resources so that new possibilities of allocating them can be explored in the Simulated_Annealing component.

3.3 Hybrid Metaheuristic Performance

For each test problem, we have a manually constructed solution that is used as a reference to establish the quality of the allocations produced by the algorithms. Space administrators helped us to construct these manual allocations. The total penalty in this manual solution for the three problems is only due to space misuse (wastage and overuse) since all the constraints are satisfied. To illustrate the effect of the hybridisation we compare it with standard simulated annealing implementation. Except for Heuristic_Hill_Climbing and the mutation operator (not included), the standard algorithm has the same structure as the Simulated_Annealing component of the hybrid metaheuristic (including the local search strategy, the cooling schedule and parameters).

In Fig. 2 we show the results after 10 runs of each approach for the three test problems. A limited CPU time according to the problem size was given to each run. The bars in the graph show the contribution to the total penalty due to space misuse and contribution due to violation of soft constraints.

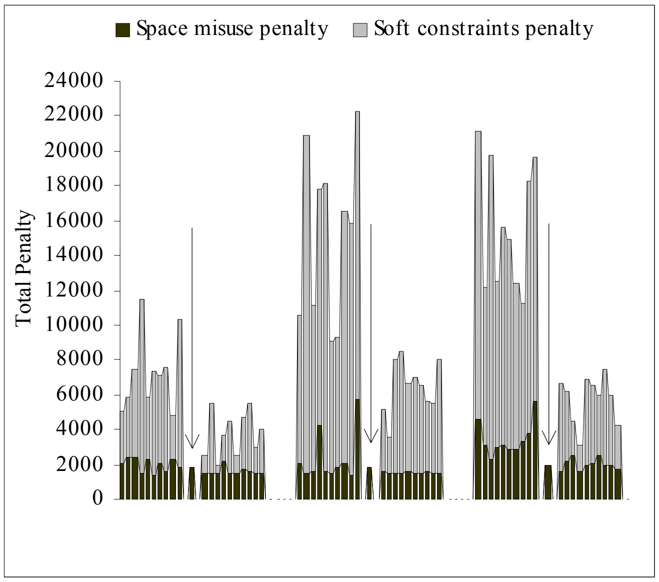

Fig. 2. Solutions obtained by the standard technique (ST) and the hybrid metaheuristic (HM) over 10 runs. A limited CPU time according to the problem size was assigned to each run. The manually constructed allocation (MA) is shown as a reference.

We observe that regarding space utilisation, it is apparent that both approaches achieve acceptable performance compared with the manual solution. The difference between the performance of both approaches is mostly in constraint satisfaction. With the exception of two solutions in test problem 1, even the worse solutions produced by the hybrid metaheuristic are better that those obtained with the standard technique. In all three problems, the hybrid metaheuristic finds solutions with a total penalty value nearly as low as the manually constructed solutions.

4 The Population-Based Approach

4.1 Adding a Population

We introduce a population for the hybrid metaheuristic, the component Heuristic_Hill_Climbing is used to initialise each individual. The Simulated_Annealing phase is applied to each individual in the population and instead of having a set of parameters for each individual (this would be like a parallel implementation of the metaheuristic), we establish a set of common parameters for the whole population. These parameters are defined by the individual that reaches a *stable state* in both the Heuristic_Hill_Climbing and the Simulated_Annealing phases. The best solution achieved by each individual during the evolution of the population is kept so that a set of best solutions is found. A *stable state* is when:

- while in the heuristic initialisation stage, an individual achieves a local optima
- during the simulated annealing phase, no move produces an improvement in the individual

A list of favourable moves is incorporated to the Heuristic_Feasible_Move_Selection (in addition to the list of tabu moves described in section 3.2.3). This list keeps a historic record of the successful moves in the population so that the individuals can share this information among them and explore these potentially profitable moves.

4.2 Parameters in the Population-Based Approach

In the Heuristic_Hill_Climbing component, HHC_Iterations is determined as described in section 3.2.2 and the estimation is based on the individual that reaches its local optima first. This provides a set of highly improved allocations but avoids the situation where each individual gets stuck in its local optima.

In the Simulated_Annealing phase, the individual that has not been improved over the greatest number of iterations controls the cooling schedule for the whole population. Then, the temperature is decreased after one individual reaches the number of iterations required (this parameter was explained in section 3.2.2). Similarly, when the temperature is zero (process *cooled*) the reheating (temperature set to the maximum value) is controlled by the first individual to reach the required number of iterations without improvement.

This strategy of controlling the cooling schedule for the whole population using one individual, makes it possible to have a set of co-operating individuals that have the opportunity to react differently to the simulated annealing process. Having a different cooling schedule for each member in the population would be the same as restarting the algorithm, but a common cooling schedule for all individuals produces interesting results as discussed below.

4.3 Using the Population to Control the Parameters

The parameters for this population-based approach, and in particular the cooling schedule, have an important effect when deciding whether to provide one single high-quality solution or a set of reasonable quality solutions. Any individual in the population may control then the common parameters for the whole population. The parameters are: HHC_Iterations, the cooling schedule and the termination criterion in the Simulated_Annealing phase. We use two ways of controlling these parameters:

- the individual that firstly comes into a *stable state* or
- the last individual in the population that comes to this *stable state*

Four main parameters (see section 3.2.2) are controlled: HHC_Iterations, the number of iterations for the cooling phase, the number of iterations after which the process is reheated and the termination criterion for the whole algorithm. If the parameters are determined using the individual that firstly comes to the *stable state*, then a single high quality solution in the population is obtained. In this case, the other members in the population that have not reached that *stable state*, contribute to the further exploration of other areas in the solution space. On the other hand, if the individual that comes to the *stable state* last is used to control the parameters mentioned above, a population of good solutions is produced. This is because all members in the population are already in a *stable state* and have achieved a local or near local optima.

4.4 Hybrid Population-Based Metaheuristic Performance

In Fig. 3 below we present the results produced by the hybrid metaheuristic with and without a population of individuals. PMH-S refers to the use of the first individual that comes to a *stable* state while PMH-M uses the individual that comes to this *stable* state last. For each test problem, 10 runs of the hybrid metaheuristic were executed and a population of 10 was used in the population-based approaches. A limited CPU time according to the problem size was given to each approach. For the hybrid metaheuristic, this time was split between 10 runs and for the population-based variants the time refers to a single run.

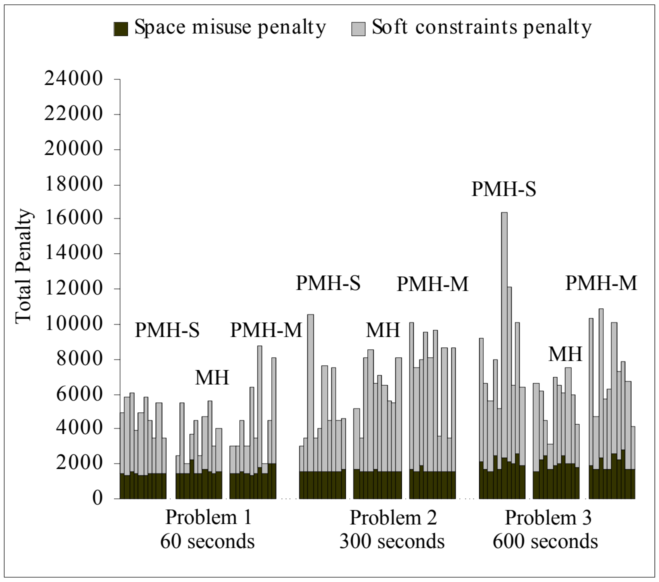

Fig. 3. Solutions obtained by the hybrid metaheuristic (MH) over 10 runs and the population-based variants (PMH-S and PMH-M) with a population size of 10. The CPU time shown was assigned to each approach according to the problem size.

We note that the population-based techniques also produce very good solutions in terms of space utilisation for the three problems. The difference is then in the achieved soft constraint satisfaction. It is clear from the histograms that none of these techniques outperforms the others. In the first problem, MH and PMH-M produced the best solutions. However, the average performances of MH and PMH-S are very similar. In problem 2, both PMH-S and PMH-M offer the best solutions. In this case, note that the average performances of MH and PMH-M are similar. In the most difficult case (problem 3), even when the best observed solution is produced by the MH approach, the number of good solutions (total penalty about 6000) produced by the three approaches is similar.

From these results it appears that the population-based variants offer similar performance to the single-solution hybrid metaheuristic and no real advantage is observed. However, the strategies to control the algorithm parameters as explained in section 4.3 are designed to offer their best performance over a maximum number of iterations between improvements rather that over the execution time. We illustrate the effect of the strategy used in the hybrid population-based metaheuristic performance in the following section.

4.5 Single High Quality Solution vs. High Quality Population

To observe the effect of the strategy selected to control the algorithm parameters, a different termination criterion is used. A maximum number of iterations between two improvements is set for each test problem. In Fig. 4 we present the results obtained by 10 runs of the single solution hybrid metaheuristic and the population-based approaches with a population of 10 solutions.

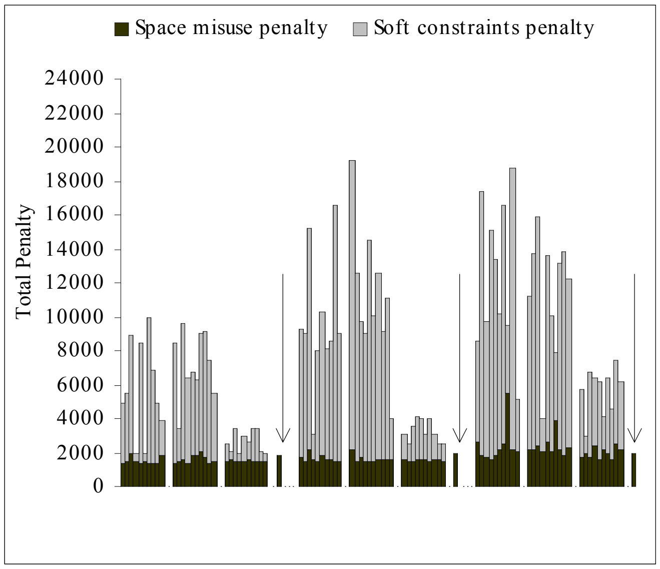

Fig. 4. Solutions obtained by the hybrid metaheuristic (MH) over 10 runs and the population-based variants (PMH-S and PMH-M) with a population of 10 individuals. The indicated number of iterations for each problem refers to the maximum between two improvements. The manual allocation (MA) for each test problem is also shown.

In the results of Fig. 4, it is clear that using this termination criterion, the population of solutions produced by PMH-M is effectively the best overall. For the three test problems, this approach finds a population of high quality solutions. The performance of PMH-S over the three problems appears to be similar, i.e. an outstanding high quality solution is found. This solution is clearly the best in the population, while the rest of the individuals have high total penalty values. We observe that the single-solution technique is also capable of producing high quality solutions for the three problems, but the variation on the results produced over several runs is also considerable.

Populations produced by the hybrid population-based metaheuristic for the single high quality solution (PMH-S) and the one for a high quality population (PMH-M) are different not only with respect to the total penalty as shown in Fig. 4. We observe in Fig. 5 that the drawback of obtaining a population of good solutions is the computation time. It is also stressed that for all the three test problems, the quality of the population produced by the PMH-M approach is clearly better that the population produced by PMH-S. So, while PMH-S finds a good quality solution quickly, PMH-M requires more CPU time to achieve a set of high quality allocations.

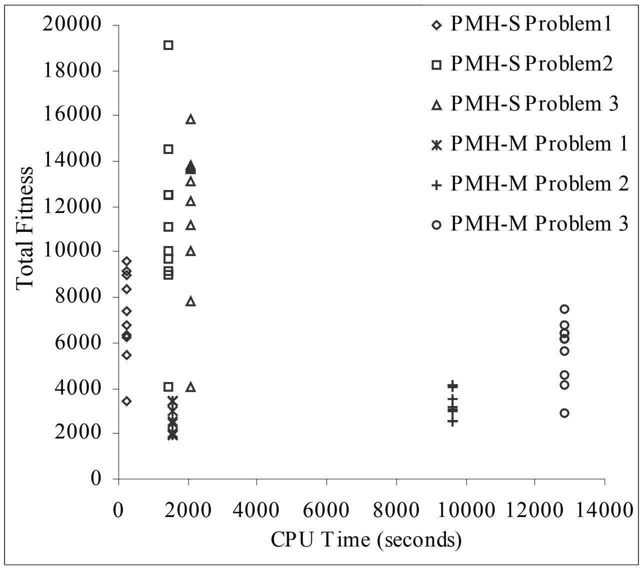

Fig. 5. Performance of the hybrid population-based metaheuristic for a single quality solution (PMH-S) and for a high quality population (PMH-M) over the execution CPU time. Each point represents a member of the population.

4.6 Some Results on Reliability

It was mentioned in section 2.6 that 30 runs were executed for each approach with all test problems. Here we present some preliminary results regarding the robustness observed in our experiments. Table 1 shows the mean (m) and standard deviation (sd) of the solutions produced (in terms of total penalty) by each approach for the test problems (P1, P2 and P3). The termination criterion in this comparison is a maximum number of iterations between improvements for each test problem.

We observe that the hybrid metaheuristic (HM) overcomes the standard technique (ST). If we compare the hybrid population-based approach for a single quality solution (PMH-S) and the approach for a high quality population (PMH-P), it is clear that the mean and the standard deviation for the population are better in the second approach. Note also that the standard deviation is better in PMH-S than in MH, but the mean is worse in two cases (P1 and P2). This is because PMH-S produces only one high quality solution, but the rest of the population has low quality. However, over all the runs the results produced by the technique are also reliable as reflected by the standard deviation value.

	ST		MH		PMH-S		PMH-P	
	m	sd	m	Sd	m	sd	m	sd
P1	7293	2046	5718	2629	7200	1816	2648	621
P2	15154	4593	9707	3593	10567	2398	3261	620
P3	15768	3490	12429	4206	11561	3299	5762	1300

Table1. Comparison of the reliability between the four techniques.

5 Conclusions

We agree with Van Valdhuizen and Lamont in [12] 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". We have presented a competitive hybrid metaheuristic for the space allocation problem using the best features of several heuristics and a certain amount of knowledge about the problem domain.

As with other combinatorial optimisation problems, in the real instances of the 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 [7]. In such situations, we suggest two possible paths: reinitiate the hybrid metaheuristic to find several solutions, or use the population-based approaches.

We propose the combination of adaptive cooling schedules in simulated annealing with population-based techniques as an alternative technique to tackle this and other similar combinatorial optimisation problems. This methodology is capable of producing one single high quality solution or a population of high-quality allocations as discussed in section 4.5.

Over a limited CPU time, the three approaches have a similar performance in our test problems as was shown in Fig. 3. The advantage of the population is evident when we allow the cooling schedule to be controlled over a maximum number of iterations.

It is worth pointing out that in the three test problems, all of the approaches achieved excellent solutions with respect to space utilisation, i.e. the reduction of the penalty due to space misuse. We contend that it is worth investigating the applicability of these techniques to other bin-packing and knapsack related problems.

Future work includes experiments using other space allocation problems to establish more specific guidelines for the parameters' values, including the population size. Additionally, an investigation of the multiobjective aspect of the space allocation problem using the techniques proposed in this paper will be carried out.

Bibliography

1. Benjamin C., Ehie I., Omurtag Y., Planning Facilities at the University of Missoury-Rolla, Journal of Interfaces, Vol. 22, No. 4, pp. 95-105, 1992.
2. Bland J.A., Space-Planning By Ant Colony Optimisation, International Journal of Computer Applications in Technology, Vol.12, No.6, pp. 320-328, 1999.
3. E.K. Burke, Varley D.B., Space Allocation: An Analysis of Higher Education Requirements, Selected papers from the PATAT '97 Conference, Toronto, Canada, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1408, pp. 106-109, 1998.
4. Burke E.K., Landa Silva J.D., The Space Allocation System – Test Data, [Online], Available: www.asap.cs.nott.ac.uk/ASAP/space/spacedata.html, [2001, February 1st].
5. Burke E.K., Cowling P., Landa Silva J.D., McCollum B., Three Methods to Automate the Space Allocation Process in UK Universities, Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, PATAT 2000, Konstanz, Germany, pp. 374-393, 2000.
6. Burke E.K., Cowling P., Landa Silva J.D., McCollum B., Varley D., A Computer Based System for Space Allocation Optimisation, Proceedings of the ICC&IE 2000, The 27th International Conference on Computers and Industrial Engineering, Beijing, China, 11-13 October 2000.
7. Dasgupta P., Chakrabarti P.P., Desarkar S.C., Multiobjective Heuristic Search: An introduction to Intelligent Search Methods for Multicriteria Optimisation, Computational Intelligence - Vieweg, 1999.
8. Giannikos G., El-Darzi E., Lees P., An Integer Goal Programming Model to Allocate Offices to Staff in an Academic Institution, Journal of the Operational Research Society, Vol. 46, No. 6, pp. 713-720, 1995.
9. Osman I.H., Kelly J.P., Meta-Heuristics: Theory & Applications, Kluwer Academic Publishers, 1996.
10. Reeves C., Hybrid Genetic Algorithms for Bin-Packing and Related Problems, Annals of Operations Research, 63, pp.371-396, 1996.
11. Ritzman L., Bradford J., Jacobs R., A Multiple Objective Approach to Space Planning for Academic Facilities, Journal of Management Science, Vol. 25, No. 9, pp. 895-906, 1980.
12. Van Valdhuizen D.A., Lamont G.B., Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art, Evolutionary Computation, Vol. 8, No. 2, pp. 125-147, 2000.