Automating Space Allocation in Higher Education

E.K. Burke, D.B. Varley[1][2]

Automated Scheduling and Planning Group,
Department of Computer Science,
University of Nottingham, UK.

Abstract. The allocation of office space in any large institution is usually a problematical issue, which often demands a substantial amount of time to perform manually. The result of this allocation affects the lives of whoever makes use of the space. In the higher education sector in the UK, space is becoming an increasingly precious commodity. Student numbers have risen significantly over the last few years and as a result, university departments have grown in size. In addition, universities have come under increasing financial pressure to ensure that space is utilized as efficiently and effectively as possible. However, space utilization is only one issue to take into account when measuring whether or not a particular allocation is of a sufficient high quality. The problem of space allocation is further complicated by the fact that no standard procedure is practiced throughout the higher education sector. Most institutions have their own standards and requirements, which are often very different to other institutions. Different levels of authority control the domains of rooms and resources in different institutions. The most common situation is where a central university office controls a number of faculties, each managing a number of departments. This paper will focus specifically on applying optimization methods to departmental room allocation for non-residential space in the higher education sector. It will look at the use of three methods (hill-climbing, simulated annealing and genetic algorithms) to automatically generate solutions to the problem. The processing power of computers and the repetitive search nature of this problem means that there is great potential for the automation of this process. The paper will conclude by discussing and comparing these methods and showing how they cope with a highly constrained problem.

Keywords. Space Allocation, Hill-Climbing, Simulated Annealing, Genetic Algorithms

1. Introduction

The problem of space allocation affects the lives of almost everyone in some way or another, whether it is the size or layout of their office or work environment, limited parking space or even the organisation of their homes. This paper will deal with the problem of efficiently allocating space within academic institutions.

As student numbers increase and university departments expand, there is significant pressure on estate managers and departmental heads to ensure space is utilised as efficiently as possible. Due to the varied requirements and constraints, this task is not simple. Obtaining just an acceptable solution often takes a large amount of man-hours.

---

[1] The author's names are listed in alphabetical order

[2] ekbldbv@,cs.nott.ac.uk (Tel. +44 (0)115 9514206, Fax. +44 (0)115 9514254)

There have been few papers which have addressed the problem of space allocation within academic institutions. Giannikos *et al.*[1] states that space allocation has received little attention in their paper on using goal programming for academic space allocation. Rizman *et al.*[2] presented a goal programming model to reassign 144 offices for 289 staff members at the Ohio State University. At a facilities level, Benjamin *et al.*[3] used the Analytical Hierarchy Process to determine the layout of a new computer laboratory at the University of Missouri-Rolla.

The size of the space allocation process is related to the number of resources that need to be allocated and the number of rooms available. In the education sector, sizes vary from 1600 rooms in 30 buildings to 20000 rooms in 600 buildings[4].

The organisation of most working environments exhibits specific structural properties, such as members of staff being grouped into departments. As such the large problem of university wide space allocation lends itself well to decomposition. This allows us to partition the problem into smaller clusters, which it is possible to deal with in a reasonable time-scale. In real life, the domain of rooms and resources within a university are managed by differing authorities, with each level managing a subset of the overall domain. These authorities may be at different levels of abstraction with a central authority administrating the whole domain. All this occurs with communication between the different levels, with lower levels regularly requesting more space and competing with other groups on the same level. It is a continually evolving problem, made more difficult by the addition and/or removal of resources/rooms all the time.

The actual allocation of resources to areas of space happens in two ways. The first being the initial allocation of a set of resources to a group of empty rooms. The second being the addition or removal of resources from a previous allocation. This paper will concentrate on the first problem of allocating resources to empty rooms, as all the principles discussed in the first problem hold for the second.

2. The Space Allocation Process

There are two different levels to the problem of space allocation, a space utilisation level and a constraint satisfaction/optimisation level.

2.1 Space Utilisation

The main requirement for the higher education sector is to find working space for all staff and students. This involves allocating specific areas/rooms for each individual resource. The amount of space required is dependent on the level, numbers and functionality of each resource. Space guidelines are published by the various education authorities and are used within this decision process. However, it is usual for universities to adapt these guidelines to suit their own requirements. The Full Time Education (FTE) 1987-space standards are the most widely adapted guidelines, with square metre values for each type of resource.

With a listing of all-available rooms and sizes, a listing of all resources, their type (e.g. Professor, Lecture Hall, Secretary, etc.) and these space guidelines it is possible to allocate resources to rooms while attempting to maximise the utilisation of the rooms.

The problem is complicated by the fact that not all resources are capable of sharing rooms with other resources, a majority actually requiring their own rooms. The problem then is to maximise the utilisation of the rooms without violating any of the sharing limitations.

2.2 Constraint Satisfaction

The most efficient utilisation of space is one which allocates all resources while minimising the amount of rooms and space wasted by these resources. However, space utilisation is simply the first part of an efficient space allocation process. There are additional constraints which need to be taken into account:

- **Resource specific requirements**
  Unique facilities that are required by certain resources impose a limit on which rooms that resource can be allocated to. For example, lecture theatres may require disabled access or Audio Visual Aid (AVA) facilities, etc. The option of modifying rooms through building work may be available, but it is more practical (and less expensive) to use rooms which already have the facilities required.
- **Ensuring grouping and close proximity of resources**
  There are two types of spatial grouping requirements: Adjacency and proximity. There is always a requirement to place resources belonging to the same group in close proximity, (e.g. members of the same department should not be widely spread over many buildings). Likewise, resources, which are highly dependent on each other, must be adjacent or at least close to each other (e.g. Wash Rooms must be adjacent to Operating Theatres etc.).
- **Ensuring distance between conflicting resources**
  This is the direct opposite to the previous requirement. It is often desirable that resources which conflict in some way should be kept at a distance from each other. For example, library space should not be located next to engineering workshops due to noise problems.

Each university will have its own constraints and opinions as to what requirements should be satisfied to make a good allocation of space.

The constraints that cause the most complexity as far as optimisation is concerned are the ones involving spatial locations. The ability to cope with these types of constraints requires information regarding the location and distance of all the rooms within the university. These graphs (fig. 1) of rooms and distances can be decomposed into subsets of buildings and floors, but even floors can hold many rooms which all require distances from each other. Obtaining this information, which is unlikely to be available at all universities, is a major task. It could be obtained through floor plans or Computer Aided Design (CAD) drawings, but would require substantial work.

To reduce the amount of information required in making decisions upon proximity information, optimisations could be made to reduce the number of proximity links between rooms such as subset grouping[5 6]. By grouping adjacent rooms together tightly and then obtaining and storing information regarding the distances of groups of rooms from each other, the amount of information can be dramatically reduced.

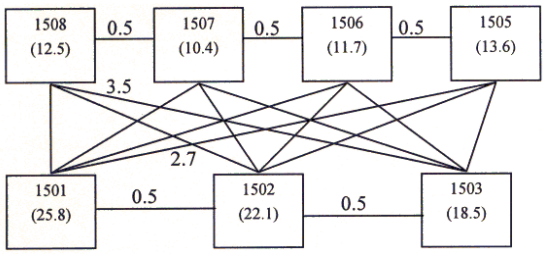

Fig. 1. Example of Room ID, size and distance information required

The same subset grouping method can be applied to the resources requiring allocation, making the decision process one of overlaying the resource subsets onto the best matching room subsets. The minimal method of satisfying proximity and adjacency constraints is to only hold information about room adjacencies, not distances. Knowing which rooms are adjacent to each other, allows the adjacency constraints to be easily satisfied, whereas the proximity constraints can be approximated by finding out whether two rooms are linked (by following adjacencies) and by how many rooms. This method allows for a compromise between excessive data gathering and constraint satisfaction and is used by the algorithms discussed in this paper.

2.3 Evaluation of the space allocation process

In order to ascertain the quality of a space allocation solution, a measure of the overall resource allocation, space utilisation and constraint satisfaction is needed. The following equation represents a generalised penalty function for any algorithmic method:

$$Penalty = \mathrm{Re}\,sourcesUnscheduled + \sum_{i=0}^{NoOfRooms-1}\left(Wastage(i) + SpacePenalties(i)\right) + \sum_{x=0}^{NoOf\ \mathrm{Re}\,sources-1}\ \sum_{y=0}^{NoOf\ \mathrm{Re}\,sources-1}\mathrm{Re}\,sourceConflicts(x, y)$$

Applying weights allows certain constraints to be considered to be more important than others and therefore have differing penalties associated with them. Table 1 shows the weighting functions used by the algorithms, for each of the sections of the equation above. Each constraint has an exponent and factorial weighting allowing greater versatility in applying penalties. An exponent of one represents a consistent penalty, i.e. each resource that is unscheduled increases the penalty by 5000. An exponent of greater than one represents an increasing exponential weighting depending on the size of the violation, i.e. exceeding room capacity by $2.0m^2$ increases the penalty by 4, exceeding by $15.0m^2$ increases the penalty by 225.

Constraint	Exponential	Factor
Resources Unscheduled – resources not allocated to rooms	1.0	5000.0
Space Wastage (per m^2) – not using full capacity of room	1.0	2.0
Space Penalties (per m^2) – exceeding room capacity	2.0	2.0
Sharing Violations – resources sharing when not allowed	1.0	2000.0
Grouping Violations – members of different groups sharing	1.0	1000.0
Adjacency Requirements – resources requiring adjacent placing	1.0	500.0
Grouping Requirements – resources requiring same/adjacent room placing	1.2	50.0
Proximity Requirements – resources requiring proximity placing	1.0	750.0

Table 1. Constraint types and penalties used in all algorithms

3. Three Methods for Automating the Problem

The methods analysed in this paper have been run using real allocation data from the School of Computer Science at the University of Nottingham. This school is moving to new premises next year due to expansion and serious space limitation of the current building, therefore this test data gives a true example of a difficult, highly constrained problem. The data consists of 83 resources, 52 rooms with 69 specific constraints. The 83 Resources consist of 1 Lecture Room, 4 Laboratories, 2 Meeting Rooms, 6 Storage Rooms, 3 Professors, 4 Senior Lecturers, 11 Lecturers, 3 Teaching Assistants, 6 Technical Staff, 8 Secretaries and 35 Researchers. The constraints consisted of 42 Resources requiring sole occupancy of a room, 7 Research groups unable to share with other groups and requiring same or adjacent rooms, 3 Secretaries needing to be adjacent/close to a manager, 3 Technical staff needing to be adjacent/proximity to laboratories/workshops and 7 Group supervisors needing to be adjacent/close to research groups.

3.1 Hill-Climbing

The hill-climbing algorithm consisted of three functions: Allocate Resource, Move Resource and Swap Rooms. Allocate resource took an unallocated resource and allocated it to a room using the appropriate fit method. Move resource took an already allocated resource and reapplied the fit method to find another room. Swap rooms took a room and swapped all the resources in that room with the resources in another. All functions chose a random source unit to work on and applied one of two fit methods to choose the target unit. The first fit method used random selection of rooms (random fit), the other chose the room with the greatest reduction in penalty (best fit).

Algorithm: Hill Climbing

1. Evaluate Current Allocation
2. Loop until the current allocation has not improved in n iterations
   - a) Select one of Allocate Resource, Move Resource or Swap Rooms in that order and apply it to produce a new allocation
   - b) Evaluate the new allocation
     - i) if it is better then make it the current allocation
     - ii) It if is not better, continue

Table 2 shows the results from 20 runs of both variations of the hill-climbing algorithm, broken down into the main groups of penalties. The average entry is a calculation of the average of all 20 runs.

Random fit managed an average utilisation of 76.5% whereas best fit managed 78.5%.

Constraints	**Random Fit**			**Best Fit**		
	Worst	**Ave**	**Best**	**Worst**	**Ave**	**Best**
Space Wastage (per m^2)	997.8	929.4	508.4	553.2	540.6	512.2
Space Penalties (per m^2)	5225.5	3502.4	177.9	932.5	738.9	184.2
Resource Penalties	10272.7	5544.12	500.0	8500.0	4700.0	0.0
Unscheduled Resources	10000.0	5000.0	0.0	0.0	0.0	0.0
Approx. Time Taken	1 minute 20 seconds			1 minute 6 seconds		
Total Penalty	26496.0	15538.4	1186.3	9985.7	5979.6	696.4

Table 2. Results from Hill Climbing algorithm tests

3.2 Simulated Annealing

Simulated Annealing is an extension to hill climbing, reducing the chance of converging at local optima by allowing moves to inferior solutions under the control of a temperature function $\exp(-\Delta/T) > R^7$, where Δ = the change in the evaluation function, T = the current temperature and R = a random number between 0 and 1

The initial temperature value and the rate of cooling affects how the simulated annealing algorithm performs. Through extensive tests, the combination of a 2200 initial temperature, a 300-iteration interval with a 100 decrement performed best.

The simulated annealing tests managed an average utilisation figure of 78.9%.

Constraint	**2200 temp /300 interval**		
	Worst	**Ave**	**Best**
Space Wastage (per m^2)	555.2	507.9	475.8
Space Penalties (per m^2)	695.56	281.7	0
Resource Penalties	2539.14	1307.6	0
Unscheduled Resources	0	0	0
Approx. Time Taken	7 minutes 46 seconds		
Total Penalty	3777.3	2097.2	475.8

Table 3. Results from Simulated Annealing algorithm tests

3.3 A simple Genetic Algorithm

Genetic algorithms (GA's) use progressive generations of potential solutions and through Selection, Crossover and Mutation, aim to evolve solutions to the problem through the principles of evolutionary survival of the fittest.

The GA in this paper consisted of a data encoding structured so that each gene represents a room, with a linked list of all the resources allocated to that room.

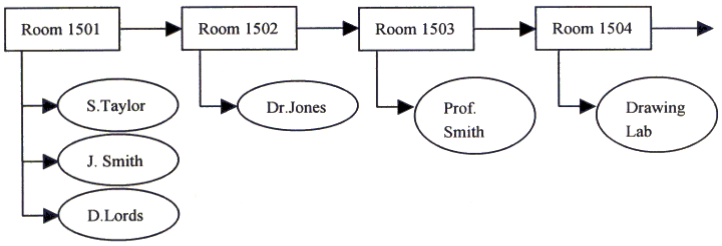

Fig. 2. Graphical representation of Genetic Algorithm encoding

The Roulette-Wheel method was used in the Selection process and the Mutation operator simply moved a resource from one room to another. However, the Crossover operator required more consideration as the standard methods frequently result in invalid solutions, i.e. a resource being allocated to two rooms. The method implemented into the GA involved checking each room in the parents and where both parents had the same resource-to-room allocation, copy that to the child. Otherwise, take a resource-to-room mapping from one parent, as long as that resource has not already been allocated.

The GA was tested with various population sizes and with various initial populations, the first of random room allocations, the second using the best fit hill climbing algorithm and the last using the simulated annealing (S.A.) algorithm. Table 4 summarises the results for a population size of 50, using elitism to ensure the best result so far is not lost.

Constraints	**Best in Initial Population**			**Best Individual after GA run**		
	Random	**Best Fit**	**S.A.**	**Random**	**Best Fit**	**S.A.**
Space Wastage (per m^2)	997.8	725.6	536.0	733.2	725.6	536.0
Space Penalties (per m^2)	3265.7	254.6	234.2	1532.5	254.6	234.2
Resource Penalties	40251.3	1265.5	500.0	7565.3	1265.5	500.0
Unscheduled Resources	15000.0	0.0	0.0	0.0	0.0	0.0
Total Penalty	59514.8	2245.7	1270.2	9831.0	2245.7	1270.2

Table 4. Results from Genetic Algorithm tests

4. Conclusions

The analysis of the figures for the space utilisation layers throughout the tests shows exactly how the different requirements and constraints of the space allocation problem conflict and work against each other. The more highly constrained a problem is, the less likely it is to ensure an acceptable level of utilisation. The methods analysed show that the automation of this process can help balance utilisation and constraint satisfaction.

Applying a polynomial-time approximation scheme bin packing algorithm[8] on the problem showed how the additional constraints affect the space utilisation. Removing all the constraints except the space wastage and space penalty constraints, allowed the binpacking algorithm to obtain 97% utilisation. Applying the sharing constraints

reduced this to 82.5%. This related to the other three methods (taking into account the constraints), managed around 76-79%.

On the full space allocation problem, the three methods all performed in a reasonably acceptable manner. The Simulated Annealing algorithm performed the best with a minimum penalty of 475.8 allocating all 83 resources with zero room capacity penalties and zero resource penalties, and a small variation between worst and best. Random Fit Hill Climbing performed the worst with an approximately 25309.7 penalty variation between best and worst. In this case, 10000 was the penalty for being unable to allocate two of the 83 resources. Best Fit Hill Climbing performs better with a 9289.3 variation, never failing to allocate all 83 resources. However, Best Fit still has problems allocating all those resources without exceeding room capacities. These results, specifically the variations in best to worst, emphasise the benefit of simulated annealing over hill-climbing methods which regularly get stuck in local optima.

The results however, are offset by the amount of time taken by each method. The hill climbing methods took around 1 minute to finish, whereas the simulated annealing method took nearer 8 minutes. Improvements on the simulated annealing algorithm can be obtained by halving the cooling interval to 150. This results in a negligible increase in the average result (+839.7) and an approx. 50% reduction in time taken

The genetic algorithm managed to obtain reasonable results from the random initialisation data, however, it failed to improve on the hill-climbing and simulated annealing initialised populations. Variations on the operators used may produce more productive results, specifically the crossover operator. It may prove effective to rely completely on the mutation operator or a combination of mutation operators, as crossover consistently required more work to ensure legal solutions are produced.

Further work is required to analyse potential improvements from further testing, specifically the possible hybridisation of the three methods with each other and with other methods such as Tabu-Search.

5. References

1. I Giannikos, E El-Darzi and P Lees, *An Integer Goal Programming Model to Allocate Offices to Staff in an Academic Institution* in the Joumal of the Operational Research Society Vol. 46 No. 6. 713-720.
2. L Rizman, J Bradford and R Jacobs, *A Multiple Objective Approach to Space Planning for Academic Facilities* in the Joumal of Management Science, Vol 25. 895-906.
3. C Benjamin, I Ehie and Y Omurtag, *Planning Facilities at the University of Missouri-Rolia*. Joumal of Interfaces, Vol. 22 No. 4. 95-105.
4. EK Burke and DB Varley. *An Analysis of Space Allocation in British Universities,* in Practice and Theory of Automated Timetabling It, Lecture Notes in Computer Science 1048, Springer-Verlag 1998. Pg 20-33
5. KB Yoon. *A Constraint Model of Space Planning.* Topics in Engineering Vol. 9, Computation Mechanics Publications, Southampton, UK.
6. F Glover, C McMillan and B Novick, *Interactive Decision Software and Computer Graphics for Architectural and Space Planning,* Annals of Operations Research 5, Scientific Publishing Company, 1985.
7. E Rich and K Knight, *Artificial Intelligence,* international Second Edition, McGraw-Hill, Ine, 1991. Pg. 71.
8. S Martello, P Toth. *Knapsack Problems.. Algorithms and Computer Implementations.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Pg. 50-52