The Teaching Space Allocation Problem with Splitting

Camille Beyrouthy[1], Edmund K. Burke[1], Dario Landa-Silva[1], Barry McCollum[2,3], Paul McMullan[2], and Andrew J. Parkes[1]

[1] School of Computer Science and IT, University of Nottingham, Nottingham NG8 1BB, UK
{cbb,ekb,jds,ajp}@cs.nott.ac.uk
[2] Queen's University of Belfast, Belfast, BT7 1NN, UK
{b.mccollum,p.p.mcmullan}@qub.ac.uk
[3] Realtime Solutions Ltd, 21 Stranmillis Road, Belfast, BT9 5AF
b.mccollum@realtimesolutions-uk.com

Abstract. A standard problem within universities is that of teaching space allocation which can be thought of as the assignment of rooms and times to various teaching activities. The focus is usually on courses that are expected to fit into one room. However, it can also happen that the course will need to be broken up, or 'split', into multiple sections. A lecture might be too large to fit into any one room. Another common example is that of seminars or tutorials. Although hundreds of students may be enrolled on a course, it is often subdivided into particular types and sizes of events dependent on the pedagogic requirements of that particular course.

Typically, decisions as to how to split courses need to be made within the context of limited space requirements. Institutions do not have an unlimited number of teaching rooms, and need to effectively use those that they do have. The efficiency of space usage is usually measured by the overall 'utilisation' which is basically the fraction of the available seat-hours that are actually used. A multi-objective optimisation problem naturally arises; with a trade-off between satisfying preferences on splitting, a desire to increase utilisation, and also to satisfy other constraints such as those based on event location and timetabling conflicts. In this paper, we explore such trade-offs. The explorations themselves are based on a local search method that attempts to optimise the space utilisation by means of a 'dynamic splitting' strategy. The local moves are designed to improve utilisation and satisfy the other constraints, but are also allowed to split, and un-split, courses so as to simultaneously meet the splitting objectives.

1 Introduction

An important issue in the management of university teaching space is that of planning for future needs. Support for such decision-making is generally divided into two broad, and sometimes overlapping, areas:

E.K. Burke and H. Rudová (Eds.): PATAT 2006, LNCS 3867, pp. 228–247, 2007.
© Springer-Verlag Berlin Heidelberg 2007

- *space management:* near-term planning,
- *space planning:* long-term planning, including capacity planning.

A fundamental stage of capacity planning aims to estimate the projected student enrollments, and multiply by the expected weekly student contact hours to obtain the total demand for ‘seat-hours’. Similarly, for the rooms we could just sum up the room capacities and multiply by the number of hours they are available in order to determine the ‘seat-hours supply’. A naive way to perform capacity planning, based on such seat-hours estimates, would be simply to ensure that the supply exceeds the demand. However, it is very rare that it is possible to use all of the seats. The efficiency of space usage is usually measured by giving a figure for the ‘utilisation’: i.e., the fraction (or percentage) of available seat-hours that actually end up being used. In real institutions, the utilisation can be surprisingly low, perhaps only 20–50%. To compensate for this, when planning the amount of teaching space to supply, we need to build in excess capacity [13,14].

Naturally, such excess capacity is expensive, because it entails planning for seats to be underused. Good planning should reduce the excess capacity without increasing the risks that expected activities will not find a space. However, this is difficult because there is little fundamental understanding of why the utilisation is so low in the first place, or of the interaction of various constraints and objectives with the utilisation.

A study of this issue was initiated in [5,6]. However, that work, like the majority of work on (university) course timetabling research was concerned with unsplittable ‘events’ (or ‘courses’ or ‘classes’). Such courses are ‘atomic’: i.e. they are not to be subdivided but need to be assigned to a single room and timeslot. However, in some circumstances, courses cannot be taken to be atomic, but must instead be subdivided, or ‘split’, before allocating them to rooms and timeslots. In this paper, we extend the work of [5,6] to the case of courses that require considerable splitting.

Our ongoing investigation into space management and space planning is closely related to research into automated timetabling but we emphasise that there is a crucial difference between the two. In automated timetabling, the set of events that should be accommodated into timeslots and rooms is usually fixed. This means that the space utilisation, in terms of seat-hours demand and offer, is also fixed from the outset. However, in this paper we want to study those factors that have an impact upon space utilisation (even before constructing the timetable). For this, we investigate a scenario in which the seat-hours demand (events to accommodate) is much larger that the seat-hours offer (available rooms). This allows us to vary the utilisation by selecting those events that will be accommodated and those that will be not. We note that although the algorithms presented here allocate events into rooms and timeslots, we are not proposing a timetabling approach. We are presenting a study that helps us to understand the interactions between space utilisation and aspects such as timetabling constraints and others.

Course splitting tends to be driven by one (or both) of the following requirements:

1. *Small-group splitting:* Courses that are intrinsically designed to be taught in small groups, such as seminars or tutorials.
2. *Constraint-driven splitting:* Courses that could, in principle, be held without splitting, but for which splitting is forced because of the following constraints:
   (a) *capacity constraints:* the course is simply too large to fit into one room.
   (b) *timetable constraints:* the enrollment is large and across such a wide spectrum of students that it will conflict with many other courses. This greatly reduces the chances of obtaining a conflict-free timetable. Splitting such a course into multiple sections can ease timetabling pressures, as students are more likely to be able to find a section that is conflict-free for them.

Standard university course timetabling methodologies (see [4,7,8,9,10,15,16,18]) assign events to rooms and timeslots, satisfying capacity constraints, so that students do not have to take two events at the same time (and possibly some sequencing or adjacency constraints) and aiming to improve the satisfaction of soft constraints such as the avoidance of unpopular times. The best-known problem that considers 'timetabling with splitting' is the 'student sectioning problem', e.g. [2,3]. This problem considers the enrollment of students into courses, but each course consists of multiple sections and students need to be assigned to sections in such a way as to avoid timetable clashes whilst respecting room capacities. This means that the student sectioning problem is most relevant to the short period between students enrolling into courses and students needing to know which section they should attend.

However, in this paper, we are not studying such 'immediate' space management problems as the student sectioning problem. Instead, we are concerned with decision support for space capacity planning over a longer time frame. For space planning, we need to understand which utilisations are achievable and how they depend on the decision criteria such as section sizes and the constraints arising from location and timetabling. Our goals are:

- Devise algorithms to carry out splitting together with event allocation;
- Explore and understand the trade-offs between the various objectives;
- Understand the impact of such trade-offs on the use of expected utilisation as a safety margin within space planning.

It should be stressed that the splitting algorithms proposed here are used to investigate long-term space planning and not to address near-term space management which is associated to timetabling.

To achieve the above goals, our general approach can be outlined as follows:

1. Formulate or model the problem: This includes obtaining a model of splitting that contains the main aspects – although it does not need to contain all the details. For example, we will cover the small group requirements by simply introducing objectives related to the section size or number.

2. Use local search and simulated annealing to explore the solution space and deal with the splitting problem.
3. Carry out experiments in order to visualise the trade-off surfaces.

The specific contributions made in this paper are:

- *Dynamic splitting:* A local search based on exchanges of events, but in which we also make decisions on how to do the splitting. Moves can split courses, and can also rejoin them in order to suit the available rooms.
- *Preliminary trade-off surfaces:* We present results on the interaction of objectives such as location and timetabling, with preferences on section sizes.

Outline of the paper. Section 2 gives a basic description of the problem constraints and objective functions and a brief description of the data sets. In Section 3, we outline a form of local search that does not include splitting, but which forms a good basis for the algorithms for splitting presented in Section 4. In Section 5, we compare the performances of the various algorithms. In Section 6, we move to the exploration of the solution space itself, presenting results for the trade-offs between the various objectives.

2 Problem Description

Teaching space allocation is concerned with allocating events (courses/course offerings, tutorials, seminars) to rooms and times. In this section, we will cover the basic language of the problem; the constraints and objectives, and the dataset that we will use.

2.1 Courses, Events and Rooms

For each *course* we have

1. Size: the number of students in the course.
2. Timeslots: the number of timeslots the course uses during the week.
3. Spacetype: Lecture, Seminar, Tutorial, etc.
4. Department: the department that owns or administers the course.

One can consider other aspects. For example, special features that are imposed by some constraints. However, we shall not consider these here. Also note that the word 'course' can mean many different things; ranging from the entire set of classes constituting a degree down to a single class. However, in this paper, we use 'course' in the sense of a set of activities of a single type such as a lecture or tutorial, and associated with a single subject. In the case of lectures, the course would be taught by a single faculty member. In general, a 'course' might have multiple associated types. For example, lectures in French grammar might always be accompanied by seminars on French literature. However, for the purposes of this paper, we will disregard such cross-spacetype dependencies and regard the lectures and tutorials as separate courses.

Courses will generally be split into sections, though we generally use the term *event* to denote courses/sections that are ‘atomic’: that is, to be assigned to a single room and timeslot. Events have the same information as courses except that each takes only a single timeslot. For events we have

1. Size: Number of students
2. Spacetype: Lecture, Seminar, Tutorial, etc.
3. Department: Department offering/managing the event.

For every *room* we have

1. Capacity: Maximum number of students in the room.
2. Timeslots: The number of timeslots per week.
3. Spacetype: Space for Lecture, Seminar, Tutorial, etc.
4. Department: The one that owns/administers the room.

The basic hard constraints (i.e. those that we always enforce) are

1. *Capacity constraint*: Size of an event cannot exceed the room capacity.
2. *No-sharing constraint*: At most one event is allowed per ‘room-slot’, where by room-slot we refer to a (room,timeslot) pair.

In this paper, we also apply the condition that the spacetype of the event must be the same as that of the room. In general, this hard constraint can be softened, and the resulting spacetype mixing is an important issue, but will be left for future work. So, henceforth, in descriptions of the algorithms we will ignore spacetypes.

2.2 Penalty and Objective Functions

Merely allocating events to room-slots so as to satisfy the capacity constraints and no-sharing constraints on its own is not useful; we also need to take into account space utilisation objectives for additional soft constraints. Based on the work in [5,6], and also from considerations of what a good allocation is likely to mean in the presence of splitting, we use the following:

Utilisation (U). [5,6] The primary objective is that we want to make good use of the rooms, and have a good number of student contact hours. We will measure this by the ‘Seat-Hours’ – which is just the sum over all rooms and timeslots of the number of students allocated to that room-slot. The utilisation U is then defined as just the Seat-Hours achieved as a fraction of the total Seat-Hours available (the sum over all rooms and times of the room capacity):

$$U = \frac{\text{Seat-Hours used}}{\text{total Seat-Hours available}}. \tag{1}$$

This is usually expressed as a percentage: $U = 100\%$ if and only if every seat is filled at every available timeslot.

Timetabling (TT). [5,6] Teaching space allocation is also constrained by timetabling needs, and we take this aspect into account. Hence, we use here a timetabling penalty (TT) that is just a standard conflict matrix between events which represents pairs of events that should not be placed in the same timeslot. For this paper, we will simply use randomly generated conflict graphs. We use $TT(p)$ to denote that each potential conflict is taken independently with probability percentage, p. For example, TT(70) means that the conflict density is (about) 70%.

Conflict Inheritance Problem. Course conflicts are used to represent the case that students are enrolled for both of the courses in the conflict. In standard university timetabling, the conflict graph will be fixed, but with sectioning. Part of the point is that students can be assigned to sections with the intention of resolving conflicts. The problem of assigning students to sections is treated, for example, in [2,3,12]. In [3] a relaxed conflict matrix is created and, in particular, it is less dense than the matrix between courses. Hence, if a course has multiple sections, then not every section ought to have the same conflicts as the parent course. That is, there is a 'conflict inheritance problem': when a course is split, how should we decide upon the timetable conflicts given to the resulting events (also see [17])? This problem is not studied here, but it represents a promising direction for future work. In this initial study of splitting, we will look at the simpler case in which the inheritance is full; that is, on splitting, each event inherits *all* the conflicts of the course.

Location (L). [5,6] A common objective in timetabling is the goal of reducing the physical travel distances for students between events. It also seems likely that students and faculty would prefer that the events they attend will be close to their own department. We do not attempt to model this exactly but instead use a simple model in which there is a penalty if the department of the event is different from that of the room-slot. Specifically, if an event i has department $D(i)$, and is allocated to a room r with department $D(r)$, then there is a penalty matrix derived from the department, $Y(D(i), D(r))$. Events in their own department are not penalised, $Y(d, d) = 0$, and the off-diagonal elements were selected arbitrarily (as we did not have physical data). The total Location penalty is just the sum of this penalty over all allocated events.

Section Size (SZ). For courses such as tutorials or seminars it is standard that they are intended to be in small groups. Hence, when splitting we need to be able to control the sizes of the sections. In this paper, we use a simple model in which we take a target size for the sections, and simply penalise the deviation from that target. Given an allocated event i, let the number of students be c_i, the total number of allocated events be I, and the target section size T. The section size penalty SZ that we use is

$$SZ = \sum_{i=1}^{I} |c_i - T|. \tag{2}$$

Section number (SN). Every section will need a teacher, and so the total number of sections allocated will have a cost in terms of teaching hours, and should not be allowed to become out of control. The penalty SN is simply the total number of allocated events. Pressure to minimise SN will tend to discourage courses from splitting into more events than are needed.

No Partial Allocation (NPA). The context in which we undertake the search is that we have a large pool of courses available and are investigating the best subset that can be allocated. However, if a course is broken into sections, then the course as a whole ought to be allocated or not. The NPA penalises those cases in which some of the sections of a course are allocated, but other events from the same course remain unallocated. Enforcing NPA as a hard constraint would disallow partial allocation: for every course, either all sections are allocated, or none are allocated.

2.3 Overall Objective Function

The overall problem is a multi-objective optimisation problem because there is conflict between improving utilisation and satisfying the constraints. However, we use a linearisation into a single overall objective or fitness F, which can be represented as follows:

$$\begin{aligned} F = W(U) \cdot U + W(L) \cdot (-L) + W(TT) \cdot (-TT) \\ +W(SZ) \cdot (-SZ) + W(SN) \cdot (-SN) + W(NPA) \cdot (-NPA) \end{aligned} \quad (3)$$

where the W(*) are simply weights associated with each objective or penalty. The minus signs merely change penalties into objectives and make all the 'dimensions' or objectives into maximisation problems.

The aim is to maximise F and consequently maximise utilisation (U) while reducing the penalties for L, TT, etc. In practice, we will consider a wide variety of relative weights. Of course, if a weight is large enough then it effectively turns the penalty into a hard constraint. The use of weights is also intended to allow modelling of the way that administrators will relax some penalties and tighten others.

2.4 Datasets

Table 1 gives an overview of the four datasets we use to test our splitting algorithms. All datasets are collected from a building of a university in Sydney, Australia. (We omitted the 'lectures only' dataset used for [5,6] as it is not relevant to splitting). Note that we use datasets in which the demand of seat-hours is much larger then their supply because this is the case that is relevant to our study. Improvements or detriments on space utilisation can only occur when the subset of events that are allocated changes.

The workshops dataset, `Wksp`, is mainly characterized by the non-uniform capacity of rooms ranging from 21 to 80, making it possible for some small

Table 1. The four datasets that we use, and some of their properties, including numbers of rooms and courses, the total *Seat-Hours* demanded by all the courses, and the *Seat-Hours* available in all the rooms

Data-set	`Wksp`	`Tut`	`Sem`	`Tut-trim`
Spacetype	Workshop	Tutorial	Seminar	Tutorial
No. of courses	1077	2088	3711	620
No. of rooms	16	184	88	47
Timeslots no.	48	46	46	50
Seat-Hours, courses	86,140	290,839	440,131	87,678
Seat-Hours, rooms	39,408	163,500	176,318	41,350

courses to fit without splitting. For `Tut`, the main characteristic of this dataset is the small capacity of rooms and their uniformity, e.g. most rooms have sizes in the range 8–20; enforcing a section size is therefore trivial in this case. The full dataset, `Tut`, is quite large and so, in order to be able to plot trade-off surfaces in a reasonable amount of time, we also created the set `Tut-trim` by randomly selecting a fraction of the rooms and courses. The seminar dataset, `Sem`, is similar in structure to `Tut`. It exhibits the same characteristics as `Tut`, and has room capacities ranging from 30 to 86 students. Both seminars and tutorials have relatively large courses and therefore splitting is essential for them.

3 Algorithms Without Splitting

In this section, we present the methods we use for cases when splitting is neither needed nor performed. Although, the focus of the paper is on splitting we think that describing the non-splitting local operators first helps the presentation of the paper.

3.1 Local Search Operators Without Splitting

The neighbourhood moves used to explore the search space are given below. Note that, by construction, all operators (implicitly) maintain feasibility of the solution. Figure 1 illustrates these local search operators.

1-swap-rand: Randomly select 2 different rooms and, in each room, randomly select an allocated event. The selected events are swapped between rooms. If the given events violate any of the hard constraints, we randomly search again for 2 other events to swap.

2-swap-rand: Similar to *1-swap* but it randomly selects 4 (2 from each room) rather than 2 events and swaps them. Special consideration is given to checking that the 4 events are all different and that one swap would not cancel the other.

Move-exterior: Randomly selects an allocated and an unallocated event and tries to swap them; assigning the unallocated event to the timeslot of the allocated one.

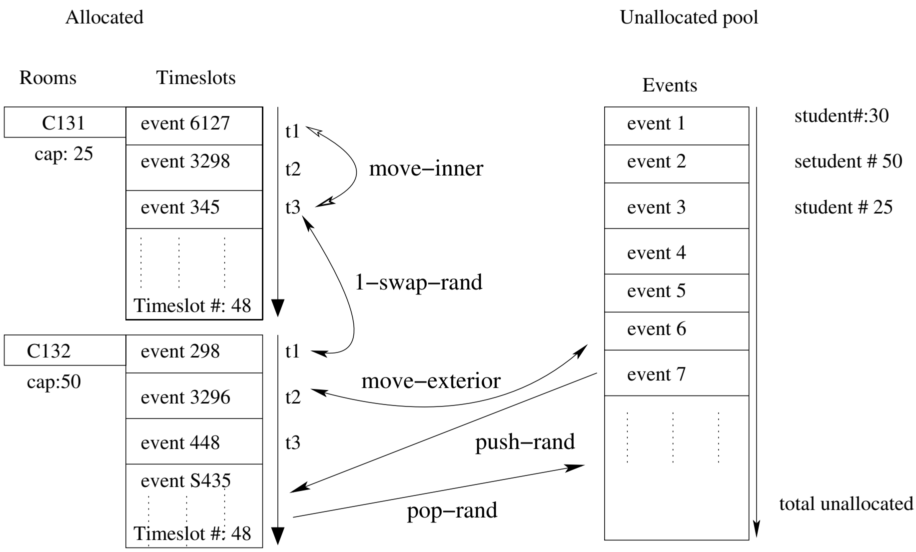

Fig. 1. Schematic of the local search operators (except ***2-swap-rand***) for the local search without splitting

Push-rand: Randomly selects one course from the unallocated set of events and tries to allocate it to a randomly selected room, also picking the timeslot at random.

Push-rand-p: This move is another version of *push-rand* but which gives priority to early timeslots in the rooms timetable, favouring them over late ones. The local search is allowed to switch probabilistically between the 2 different versions of *push-rand*.

Pop-rand: Randomly selects one event from a randomly selected room and deallocates it.

Move-inner: Swap 2 randomly selected events in a given room between 2 randomly selected timeslots.

3.2 Meta-heuristics

We only use hill-climbing and simulated annealing [1,11] implementations in this paper.

The hill climbing algorithm (HC) variant uses most of the moves given above to perform a search of the neighbourhoods. On each iteration, it selects an operator from the list above according to a given move probability and applies it to generate a candidate solution. If the candidate solution has better (or equal) fitness than the incumbent, we commit to the move, but otherwise disregard it.

Simulated Annealing (SA) was used as the main component for overcoming local optima. A geometric cooling schedule was used, specifically temperature $T \rightarrow \alpha T$ every 650 iterations with $\alpha = 0.998$. We generally used 6 million iterations and initial temperature $T_i = 0.6$. Such a slow cooling and such a large number of iterations were chosen to err on the side of safety.

4 Algorithms with Splitting

In this section, we describe the splitting heuristics that are incorporated into the HC and the SA approaches. Two strategies are implemented: (a) construction-based splitting, and (b) dynamic local search-based splitting. In the first case, the section size is calculated during the construction of an initial solution and remains fixed for all events throughout the local search. In dynamic splitting, the section size is calculated as the local search progresses according to the size of the event (and room capacity) that is being allocated. Hence, we will have

- SS-HC: Construction-based static splitting and hill-climbing
- SS-SA: Construction-based static splitting and simulated annealing
- DS-HC: Dynamic splitting and hill-climbing
- DS-SA: Dynamic splitting and simulated annealing.

4.1 Static Splitting

In static splitting we select a target section size (generally based on room profiles) and then split all the courses of size larger than that target size, into as many sections as needed during the process of constructing an initial solution. We use the term static, because once a split is enforced it cannot be changed. We afterwards run a local search algorithm (HC or SA) to improve the initial solution. So, in this strategy, splitting happens within the construction and this provides no flexibility in changing section size during the local search.

There can be many ways to calculate and fix the target section size. Here we compare three variants which are based on the notion of a 'target room capacity'. This means that the target section size is calculated based on the capacity of the rooms that are available for allocating course sections. Specifically, the target section size is fixed to one of three different values:

1. MAXCAP – the largest room capacity
2. AVGCAP – the average room capacity
3. MINCAP – the smallest room capacity.

We recognise that more elaborate ways to calculate the target section size are possible based on information from the room profiles. However, our interest here is to explore how splitting during the construction phase affects the search process in general, and to compare it to the case in which splitting is carried out during the local search (dynamic) which is described in the next section.

4.2 Dynamic Splitting Operators

In dynamic splitting, we calculate the section sizes during the local search itself. The dynamic splitting heuristic is also capable of un-splitting/rejoining sections and this gives more flexibility to determine an adequate target section size by changing, adding, deleting and merging sections as needed.

Dynamic splitting is embedded in the local search in such a way that there is freedom and diversity in the choices of section sizes. Thus, the dynamic splitting operators consider not only the room capacities (as in the case of static splitting) but also the location (L), timetabling (TT), section number (SN), section size (SZ), and no partial allocation (NPA) constraints. Note that, at the current stage, the operators themselves do not directly respond to penalties that are considered by the local search. Presumably, this leads to inefficiencies because good moves will need to be discovered via multiple iterations of the SA/HC rather than directly and heuristically with the operators; we intend to investigate this in future work.

In the search, it is important to note that the 'pool of unallocated courses' is a pool of the portions of courses that are not yet allocated. The unallocated portions contain no information about how they are going to be split. That is, it is not a pool of sections waiting to be allocated, but instead the sections are created during the process of allocation. That is, the main characteristic of the splitting operators lies in the fact that when a split occurs, we actually select a fraction of a course and allocate it. When a section is unallocated, we merge it back with the associated course without keeping track of previous section splits.

Below, we detail the neighbourhood operators used in the dynamic splitting (ordered roughly by their degree of elaboration):

1-swap-rand-sec: This operator works in a similar way to ***1-swap-rand*** described in section 3.1 but the move is carried out between 2 sections (not necessarily of the same course).

Move-inner-sec: This operator works in a similar way to ***move-inner*** described in section 3.1 but the move is carried out between 2 sections (not necessarily of the same course).

Push-rand: This operator works in a similar way to ***push-rand*** described in Section 3.1 but note that the events being 'pushed' to the allocation are sections of a course that are smaller than the chosen room, and so no splitting was needed.

Pop-unsplit: This operator is used to remove sections from their allocated room and unsplit/rejoin sections with their unallocated parent course. Note that this move can be seen as the reverse operation to splitting but not exactly, because we do not keep track of the splits made during the search by ***split-push*** and

split-max that we describe next. First, the

pop-unsplit operator chooses (at random) an allocated event from a randomly selected room. In the case that the chosen event is a section, the operator unallocates the section and merges it with its unallocated parent event. If the event is not a section it is simply added to the unallocated pool.

Split-push: This operator is used to handle courses whose unallocated portion is larger than the chosen room. This is the main operator that is used to create new sections. It is at the heart of the dynamic splitting:

Proc: split-push
1 Randomly select a room R_j with available timeslots.
  Let its capacity be C_j.
2 Randomly select a course P_i from the unallocated pool.
  Let the size of P_i be N_i.
3 Set size $s = floor(C_j * rand(\delta, 1))$
  though if $s > N_i$ then s=N_i
4 Randomly select empty room-slot t_j
5 Create section S_i with size s
  and resize the remainder P_i
6 Set that S_i inherits all conflicts from course P_i (see section 2.2)
7 Generate candidate move by allocating S_i to room R_j in timeslot t_j

Note that $rand(\delta, 1)$ means a number randomly selected from the interval $[\delta, 1]$ and the parameter δ is described below. After randomly selecting a room-slot and unallocated course, the main step in this operator comes in its decision as to how to split the course to create a new section. Assuming that the capacity of the room is smaller than the size of the remainder of the course, the new section size, s, is calculated by multiplying the capacity of the room by a randomly selected factor. The factor depends on a 'section re-sizing parameter', δ, that we give a value between 0.4 and 0.6. Suppose that we take $\delta = 0.4$ then this effectively means that the generated section size, s, will be between 40% and 100% of the selected room's capacity. The intention of this randomised selection of section size is that it enables the search to discover section sizes that match the penalties such as section size and section number. The new section inherits all of the conflict information from its parent course – see the discussion of the 'Conflict Inheritance Problem' in Section 2.2. The new section is then allocated to the chosen room. The remaining part of the parent course is left in the unallocated list of courses with its size reduced appropriately.

Split-max: This operator is a version of ***split-push*** with δ=1 and is designed so that courses with size larger then the chosen room are split so that sections are of the maximum size allowed within the chosen room.

4.3 Example of the Operator Application

An example of the search process, and the differences that can arise during search, are illustrated in the simple example of Figure 2. Two courses C1 and C2, of sizes 120 and 40 respectively, are to be allocated to the four rooms available. We have selected capacities so that total size of courses precisely equals the total capacity of the rooms. In the first case, it happens that the smaller event C2 is allocated first via a ***push-rand*** because it can be allocated to that room without a split. However, this inevitably means that 20 spaces within room R2

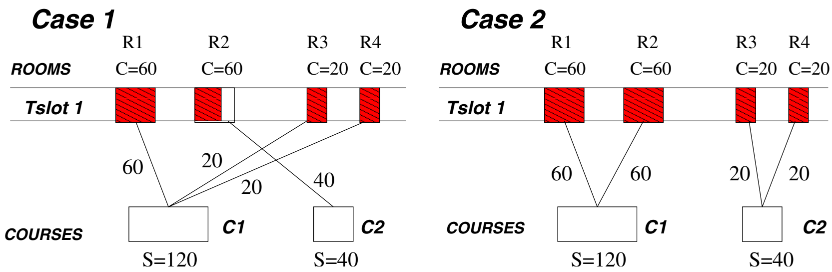

Fig. 2. Example in which applying operators to split courses has different effects. In case 1, course C2 first receives a ***push-rand*** into room R2, and then applications of ***split-push*** to C1 are able to allocate only $60 + 20 + 20 = 100$ students rather than the needed 120. However, in case 2 we see that reversing the order allows all of both courses to be allocated.

are wasted, and so it becomes impossible to allocate all of course C1. In the second case, the larger course, C1, is first split using ***split-max*** and then we end up with a perfect fit. The operator ***split-max*** with its implicit 'maximum size sections first' is often better at maximising the utilisation, although there are other cases in which ***push-rand*** is necessary. For this reason, and also from experimental evidence, we tend to give the operator ***split-max*** more probability of being selected than the operator ***push-rand***.

4.4 Controlling the Search

We have observed, in an informal manner, that the effectiveness of each operator varies during the search. As an example, suppose we are just carrying out non-splitting local search from Section 3. We start with an empty allocation, and then the ***Push-rand*** operator is the most important and successful in the early stages as events/courses need to be allocated, but for capacity reasons it remains stalled during the rest of the search, during which the other moves provide the bulk of the successful search efforts. This led to us taking a simple, though adequate, compromise with probabilities of around 10–20% for each operator.

5 Experimental Comparison of the Algorithms

In this section, we first investigate the 'static splitting' method in which only the construction does any splitting and is followed by simulated annealing. Note that 'construction followed by hill climbing' is not presented as, unsurprisingly, it performs no better than the simulated annealing version. We find that it is far inferior to the dynamic splitting. Moving to the dynamic splitting itself we then compare the HC and SA variants, and we see that the DS-SA variant is the better.

Although it seems evident from the datasets in Table 1, we illustrate the importance of splitting in our scenario. The following table compares some examples of the utilisation percentages obtained (and the number of events allocated) without any splitting (not even static splitting from the construction) and compares them with those obtained by DS-SA:

	Wksp	Tut	Sem
SA, no splitting	36% (264 ev)	0.015%	0.013%
DS-SA	70% (720 ev)	26% (1747 ev)	44% (3000 ev)

We clearly see that splitting is essential for the tutorials and seminars as, otherwise, virtually nothing is allocated. For the workshops, some courses can be allocated, but we still lose a lot compared to when splitting is allowed. So from now on we always permit splitting (we refer the reader back to Section 2.4 where the datasets are presented and the difference between the Wksp data set and the others was also noted). While, in the results above, utilisation figures seem a little higher than in real-world cases (30–40%) we show, in later sections, how the different actual constraints drive the utilisation down to more practical levels; the introduction of section size penalty along with the No-Partial-Allocation penalty can also generate a realistic level of utilisation figures.

Our results are generically presenting trade-off curves which are approximations to Pareto fronts. These are generally representing the trade-off between two of the objective functions. We select a wide range of relative values for the weights associated with the two chosen objectives, and then call the local search with those weights. For example, we often plot the trade-off between utilisation, U, and location, L. In this case, we pick a non-zero value for $W(U)$, and then just search at each of many values for $W(L)$. This leaves some gaps in the curves due to the presence of unsupported solutions. However, generally the gaps are small and we do not expect that filling them would significantly change the overall messages from the results. Note that since L is a penalty, then the objective is essentially $-L$, and we use this for the y-axis, so that 'better' is towards the top-right corner (and similarly for all others of our trade-off graphs).

5.1 Dynamic vs. Static Splitting

Figure 3 shows the trade-off curves between utilisation and location for the three different methods from the static splitting (see Section 4.1), and compares them to the results from the dynamic splitting method, DS-HC.

We see that for the construction, splitting based on the average room capacity (AVGCAP) outperforms the other two (MINCAP and MAXCAP). This is reasonable, as when splitting by the smallest room capacity there is capacity wastage in larger rooms and when splitting is based on the larger room size there is a wastage caused by violating room capacities, since we cannot allocate a section to a room with smaller capacity.

However, it is also clear that all our construction-based splitting methods are easily outperformed by the dynamic splitting. This is unsurprising, as it is

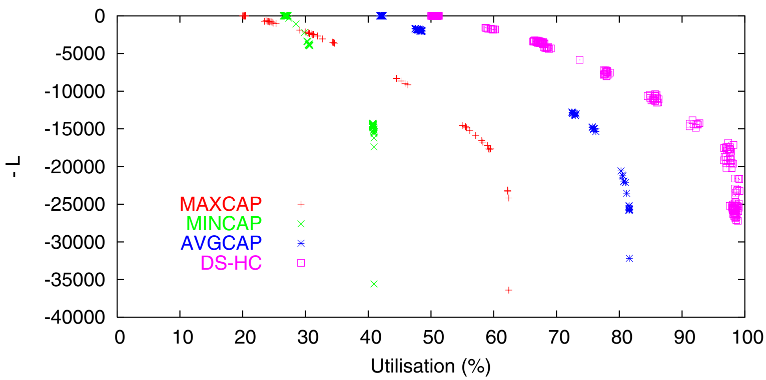

Fig. 3. Comparison of dynamic and static (construction-based) splitting for the `Wksp` data set. Plots give the trade-offs obtained between Utilisation and Location, all the other objectives being disregarded (W_{TT}=W_{SZ}=W_{SN}=W_{NPA}=0). The first three sets of points are from the three constructive methods of Section 4.1; the last 'DS-HC' from the dynamic splitting with hill-climbing.

entirely reasonable that it is best to do splits based upon the availability of room capacities rather than on a uniform target capacity. It is possible that a more sophisticated constructive method would perform much better. However, for the purposes of this paper we will henceforth consider only dynamic splitting.

5.2 Dynamic Splitting: HC vs. SA

Figure 4 illustrates the different performances of DS-HC and DS-SA on the workshop problems in the presence of timetabling. Figure 5 is the same except that it is for a tutorials dataset. As is well known, the conflict graph of the timetabling penalty moves the problem to a variant of graph colouring. So it is not surprising that the SA is likely to outperform the HC, as SA can escape local minima but HC cannot. Perhaps more surprising is the observation that the performances in the absence of TT are often very similar.

In any case, it is clear that DS-SA is the best of the algorithms that we have considered, and so will be assumed from now on whenever we have a TT penalty (and in the absence of a TT penalty it seemed to matter little which one is used).

6 Trade-Offs Between the Various Objectives

Having selected dynamic splitting as our algorithm of choice, we now change focus: we no longer pursue the solution algorithm itself, but instead focus on the solution space. In particular, we present some preliminary results on how the various objectives interact and, in particular, the magnitude of their effect on the utilisation.

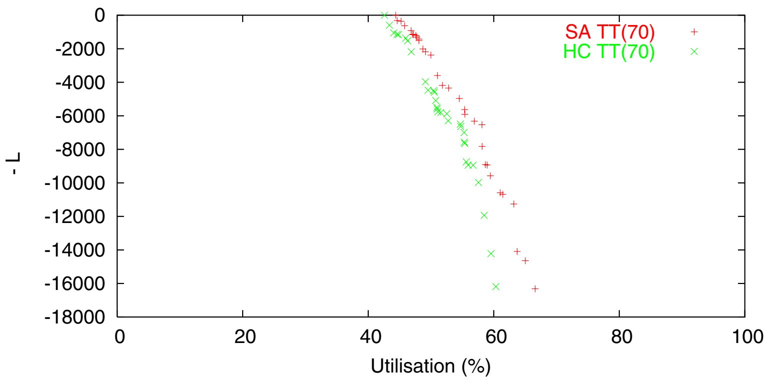

Fig. 4. Trade-off of utilisation and location as obtained with dynamic splitting, and using the HC and SA algorithms. For the `Wksp` data, and in the presence of TT(70), and no other constraints beside U, L, and TT.

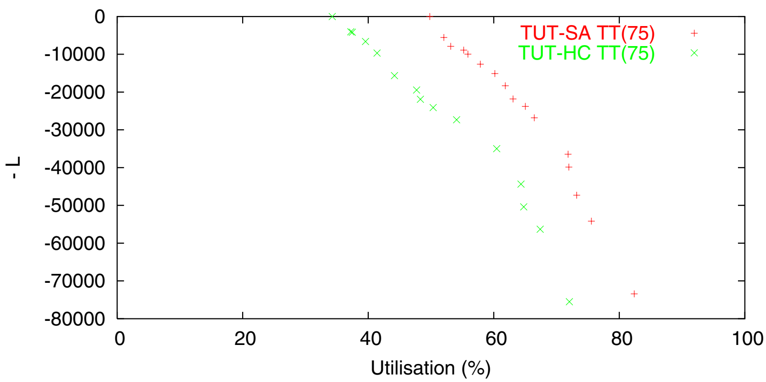

Fig. 5. As Figure 4 but instead using the tutorials dataset, `Tut-trim`, and with TT(75)

6.1 Interaction of Section Size Penalty (SZ), Location Penalty (L), and Utilisation (U)

Figure 6 gives plots of the trade-off between utilisation (U) and location (L), in the presence of various weights, $W(SZ)$, for the section size penalty (SZ), with a target section size of 25, but with no other penalties. Note that the case $W(SZ) = 0$ was seen previously as the best line in Figure 3, and illustrates that (even without section size constraints) demanding a low location penalty has the potential to significantly reduce the utilisation (from about 98% down to 50%). The non-zero values for $W(SZ)$ drastically reduce the utilisation: dropping to the range 10–50%. This corresponds to a policy of a fixed size, but with such an excessively-strict adherence to that policy that the overall room usage suffers.

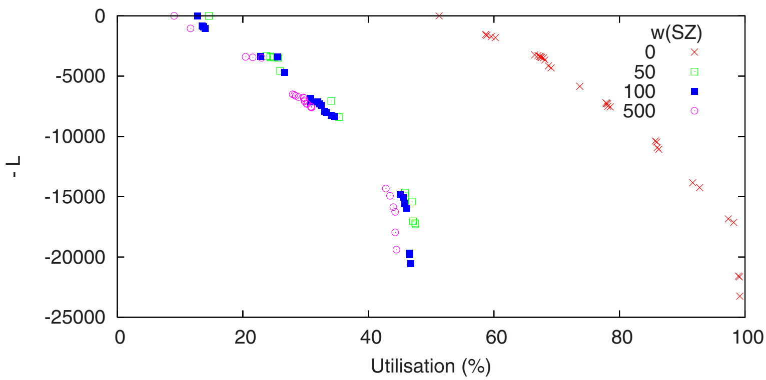

Fig. 6. Trade-off surfaces for the given values of the weight $W(SZ)$ for the section size policy. On the `Wksp` dataset, with a target section size of 25, and aiming to optimise only utilisation U, location L, and section size SZ.

6.2 Trade-Offs Arising from Section Size Penalty and Utilisation

So far, we have only looked at trade-offs between Utilisation and Location. However, now, in Figure 7 we show the trade-off between utilisation, U, and section size penalty (SZ). This happens to be with a small weight given to the section number penalty, SN; however, with no other penalties: $W(L) = W(TT) = W(NPA) = 0$, so in this case location penalties are ignored. Each curve illustrates the drastic drop in utilisation as we move towards the section size becoming a hard constraint. We also see that reducing the target for the section size reduces utilisations though by a lesser amount. Part of this effect is possibly because our current section size penalty does not allow a range of values for the section size and because it penalises under-filling a section just as much as overfilling.

6.3 Effects of Timetabling Constraints

Figure 8 is a plot of the usual trade-off between utilisation and location objectives, but comparing the presence and absence of a timetabling constraint. The case with timetabling is with conflict matrix of density 70%, and with an associated weight W(TT) that is large enough that the timetabling is effectively enforced as a hard constraint. This illustrates that timetabling issues have the potential to significantly reduce the utilisation, and so again could be part of the explanation for the low values of utilisation observed in real problems.

6.4 Inclusion of the No-Partial-Allocation Penalty

So far, we have presented results for cases in which the ‘No Partial Allocation’ (NPA) objective is ignored: that is, $W(NPA) = 0$. This means that some sections from a course can be allocated even though others are unallocated. This gives the search extra freedom, and so it is reasonable that enforcing NPA will only further

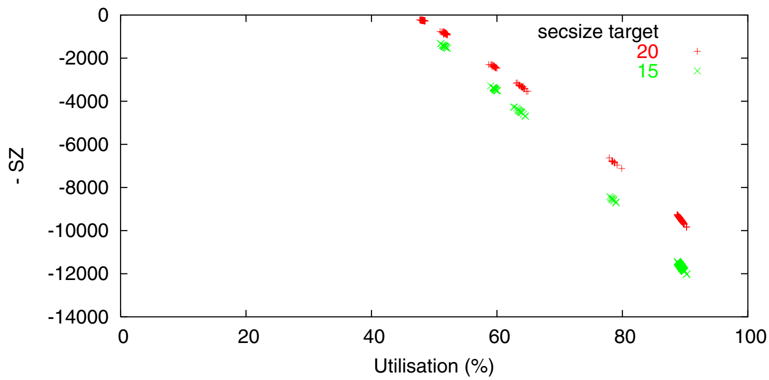

Fig. 7. Utilisation vs. section size penalty, SZ, for the `Wksp` data set, and for two values (15 and 20) of the target section size

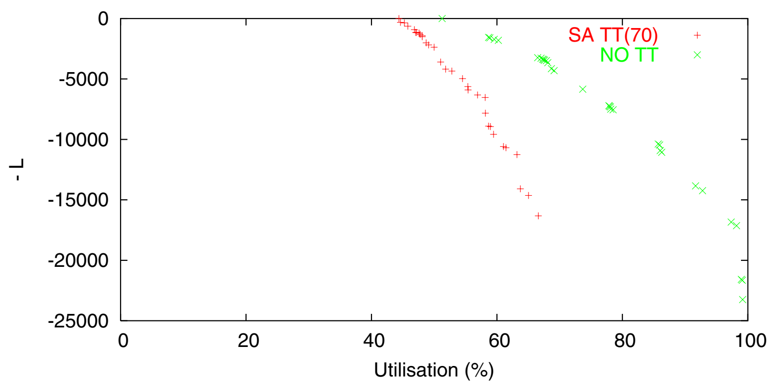

Fig. 8. Trade-offs between Utilisation and Location for the `Wksp` dataset. 'No TT' means that no objectives besides L and U are weighted, in particular W(TT) = 0. In contrast, 'TT(70)' means that a timetabling constraint with a density of 70% is enforced as a hard constraint.

reduce the utilisations obtained. The magnitude of this effect is seen in Figure 9. We see that giving NPA high weights can further reduce the utilisation by about 10–20%. This is a significant effect, although it is somewhat smaller than the effects seen in the trade-offs with the timetabling and section size objectives. It is also interesting that the effect of the NPA becomes very small when selecting solutions with small location penalty.

7 Summary and Future Work

We have devised methods, and performed preliminary studies, to support long-term teaching space planning in the presence of courses that will need to be split down into multiple sections.

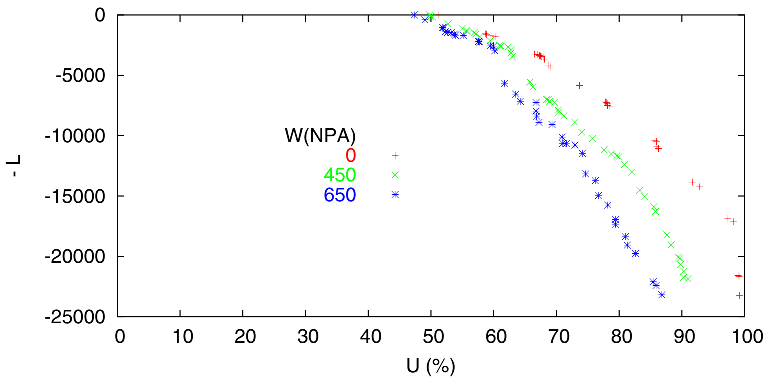

Fig. 9. Trade-offs between Utilisation and Location, in the presence of various strengths of the 'No Partial Allocation' (NPA) penalty, but with no TT or other penalties

The work has two broad aspects. Firstly, we provided algorithms to perform splitting and optimisation in the presence of multiple objective functions, including overall space usage, constraints inspired from timetabling, and also objectives relating to desirable properties of the splits themselves. In particular, we devised a splitting algorithm in which the decisions as to course splitting are incorporated within a local search.

Secondly, we used an implementation of the dynamic splitting in order to explore the trade-offs between various objectives. We found that the incorporation of objectives other than solely employing utilisation can result in the utilisation dropping from over 90% down to much lower figures such as 30–50%. This is significant because such low utilisations are consistent with the real world; and so our model ultimately has the potential to explain real-world utilisation figures. The intended longer term consequences of such better understanding will enable an improved ability to engineer the safety margins that need to be built into capacity planning. We acknowledge that other factors apart from those considered in this study might also have an impact on the utilisation inefficiency that occurs in real-world problems. The evidence presented here is a first step towards a wider investigation of this issue.

In future work, we intend to improve the speed and scope of the methods. This will have multiple aspects, of which perhaps the most important is to model the conflict inheritance issues that we discussed in Section 2.2. At the moment, we do not answer, or indeed model this problem. In the absence of a good model for this inheritance, we do not answer here the questions as to how the degree of inheritance affects results. Our inheritance is either total or nonexistent. That is, all sections inherit either all conflicts of the associated course, or else they inherit none (equivalent to simply turning off the timetable penalty). Although a deficiency, this does at least allow us to put bounds on the effect of the timetabling. The effect of partial inheritance must lie between the two extremes of total and no inheritance. Building a model for the partial inheritance, and exploring its effects is a high priority for future work.

References

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzman Machines. Wiley, New York (1990)
2. Alvarez-Valdes, R., Crespo, E., Tamarit, J.: Assigning students to course sections using tabu search. Annals of Operations Research 96, 1–16 (2000)
3. AminToosi, M., Yazdi, H.S., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 147–160. Springer, Heidelberg (2005)
4. Bardadym, V.: Computer-aided school and university timetabling: The new wave. In: Burke, E.K., Ross, P. (eds.) Practice and Theory of Automated Timetabling. LNCS, vol. 1153, pp. 22–45. Springer, Heidelberg (1996)
5. Beyrouthy, C., Burke, E.K., McCollum, B., McMullan, P., Landa-Silva, D., Parkes, A.: Towards improving the utilisation of university teaching space. Technical Report NOTTCS-TR-2006-5, School of Computer Science & IT, University of Nottingham (2006)
6. Beyrouthy, C., Burke, E.K., McCollum, B., McMullan, P., Landa-Silva, D., Parkes, A.: Understanding the role of UFOs within space exploitation. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, pp. 359–362 (August 2006)
7. Burke, E.K., Jackson, K., Kingston, J., Weare, R.: Automated timetabling: The state of the art. The Computer Journal 40, 565–571 (1997)
8. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research 140, 266–280 (2002)
9. Carter, M.: Timetabling. In: Gass, S., Harris, C. (eds.) Encyclopedia of Operations Research and Management Science, pp. 833–836. Kluwer, Dordrecht (2001)
10. Carter, M., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
12. Laporte, G., Desroches, S.: The problem of assigning students to course sections in a large engineering school. Computers and Operations Research 13, 387–394 (1986)
13. McCollum, B., McMullan, P.: The cornerstone of effective management and planning of space. Technical Report, Realtime Solutions Ltd (2004)
14. McCollum, B., Roche, T.: Scenarios for allocation of space. Technical Report, Realtime Solutions Ltd (2004)
15. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, pp. 1–23. Chapman and Hall/CRC Press, London (2004)
16. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review 13, 18–27 (1999)
17. Selim, S.: Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. The Computer Journal 31, 76–82 (1988)
18. de Werra, D.: An introduction to timetabling. European Journal of Operational Research 19, 151–162 (1985)