Threshold Effects in the Teaching Space Allocation Problem with Splitting

Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva
School of Computer Science, University of Nottingham,
Nottingham NG8 1BB, UK
{ cbb, ekb, jds }@cs.nott.ac.uk

Barry McCollum, Paul McMullan
School of Computer Science, Queen's University,
Belfast, BT7 1NN, UK
{ b.b.mccollum, p.p.mcmullan }@qub.ac.uk

Andrew J. Parkes*
School of Computer Science, University of Nottingham
Nottingham NG8 1BB, UK
{ ajp }@cs.nott.ac.uk

April 14, 2008

Keywords: space allocation, timetabling, phase transition, threshold

Abstract

Universities strive to manage space as effectively as possible. A major part of the management process is the allocation of teaching activities to rooms which have appropriate size, layout and resource availability. Matching the needs of the overall teaching requirement with the space provision within an institution is difficult to achieve and is currently performed poorly within the university sector. Furthermore, in many cases, it is a requirement to split an activity over more than one room. For example, lectures can be too large to fit in one room or good teaching practice requires that seminars and tutorials are taught in small groups. Space management then involves decisions on splitting as well as the assignments to rooms and time-slots. These decisions must be made whilst satisfying the pedagogic requirements of the institution and constraints on space resources.

The efficiency of such management can be measured by the "utilisation": the percentage of available seat-hours actually used. In many institutions, the observed utilisation is unacceptably low, and this provides our underlying motivation: to study the factors that affect teaching space utilisation, with the goal of improving it.

We give a brief introduction to our work in this area, and then introduce a specific model for splitting. We present experimental results that show threshold phenomena and associated easy-hard-easy

*Contact Author.

patterns of computational difficulty. We discuss why such behaviour is of importance for space management.

1 Introduction

Decisions regarding the provision of teaching space in universities are complex and difficult, yet have a direct impact on the costs and quality of service for students and staff. For example, over-provision of space leads to higher capital and running costs. Conversely, under-provision will lead to poorer service with events being unable to find space or being forced into inappropriate timeslots or locations.

Furthermore, there is evidence that the current system is deficient. A simple metric, used by the "Higher Education Funding Council for England (HEFCE)," to measure the efficiency of teaching space usage is "Utilisation" which gives an indication of the percentage of "seat-hours" being actually used. There are reports suggesting that levels of teaching space utilisation in UK universities is rather low (around 20-30%) [20, 21]. Paradoxically, though anecdotally, there is often a perception that there is lack of teaching space because it can be difficult to find appropriate rooms and timeslots when organising teaching events. However, acquiring extra space for academic activities is generally met with reluctance on the basis that space is only lacking because of poor space management.

Hence, an overall aim of universities is to improve the utilisation. However, of course, there must be limits on what is possible within the context of practical objectives for timetabling besides space issues. In a naive approach to space planning, one might think that if the overall *seats-supply* matches closely the overall *seats-demand*, it should be possible to allocate teaching space with high (close to 100%) utilisation levels. However in practice this will not be achievable. Instead, some lower utilisation might be expected; but precisely how low? This paper is part of an ongoing project [5, 6, 4] to illuminate the issue of what utilisations universities might be expected to reliably achieve, and how to achieve them.

Before proceeding further, it is important to give the context for the work, and in particular to emphasise the differences from course or curriculum timetabling [29, 25, 11]. As usual, for capacity planning problems, there are multiple time-scales and rolling time horizons involved. At a high level, we may think of the following rough division into time phases. (We emphasise that the division is likely to be approximate; phases might well overlap, planning and management are likely to be repeated, etc.)

- **Space planning:** "Long term." Campus or new building design, with a time horizon of around 5-50 years.
- **Space management:** "Short term." Remodelling of existing space, with a time horizon of around 1-5 years.
- **Course timetabling:** "Immediate." Allocate events to times and rooms for next term or semester. The time horizon is "immediate" because it needs to be done for the next teaching session.

Studies of space planning and management interact with course timetabling but there is a crucial difference. In instances of course timetabling we are given a set of events and a set of rooms. From the definitions ([6] and Section 2), it will be clear that utilisation is obtained by simple counting of seat demand and availability, and so is fixed from the outset. In some respects, we might even say that as far as utilisation is concerned, by the time we get to the course timetabling stage then 'the damage has already been done' and there is little that better timetabling can do to fix poor utilisation. In standard course timetabling, the aim is to optimise other objectives (such as student preferences) but our focus *must* be on the stages before this and their interaction with course timetabling. If we are to study the factors that can change utilisation then it must be allowed to take many different values and cannot be predetermined. Hence, we must do at least one of (i) vary total demand, or (ii) vary total capacity. In this paper, we follow the same approach as [6, 4] in that we vary the total demand by varying the set of events whilst keeping the set of rooms fixed.

In [5, 6], we found out that there is a *threshold* for "Is it possible to allocate all events into the available space and achieve a target utilisation U_R?". We identified "critical values" of target utilisation U_C, if $U_R < U_C$ the answer is "almost always yes" but if $U_R > U_C$ the answer is "almost always no". The threshold is pertinent because it gives a method to determine levels of utilisation that can be achieved reliably, and so support decisions as to how much teaching space to provide. Such thresholds have been extensively studied within Graph Theory and later within Artificial Intelligence (AI), however, we expect that they are not so well-known within the Operational Research (OR) community and so briefly review them here.

1.1 Thresholds: A Brief Review

In the area of graph theory, the study of thresholds has focused on random graphs [8]. A standard model for random graphs is denoted $G_{n,p}$ where n

is the number of nodes, and p the probability of an edge between any two nodes. The probability, $Pr(P)$, of a random graph having some boolean property, P, such as 'has a large cluster' or 'is fully connected', is studied as a function of n and p. It turns out that the (n, p) parameter space is typically divided into regions in which the probability is asymptotically close to one, $Pr(P) = 1 - o(n)$, stated as instances "almost always" have P, and other regions in which $Pr(P) = o(n)$, that is, instances "almost never" have property P, and with a sharp demarcation line between these regions. Often, these regions are called phases, and the demarcation line a phase transition, to follow the similar phenomena in physical systems; for example, ice, water, and steam are phases of H_2O with sharp phase transitions at temperatures 273K and 373K (at atmospheric pressure).

It is important to emphasise that such thresholds are not rare. As soon as one looks at large instances arising with some random component, but from some similar source, then thresholds become a common occurrence. There is even a remarkable "zero one law" within random graphs that any suitable property (non-trivial and monotonic under the addition of edges to a graph) will exhibit a threshold [34].

Thresholds in graph theory have also been applied within percolation theory [35]. An archetypal application would be the study of the feasibility of extraction of petroleum from oil-bearing rocks. Roughly speaking, the oil resides in pores in the rock which are considered as the nodes of a graph, and there is some probability of neighbouring pores being connected by a small channel along which oil may flow. In an associated graph model, nodes are arranged on some form of grid, and edges added between neighbours with some probability p. The property of interest is the number of nodes reachable, on average, from a given node, or the average size of connected clusters. For small p the clusters are small, but as p increases they increase in size, until at some critical value of p enough small clusters connect together such that very large clusters appear; corresponding, it is hoped, to being able to extract a large amount of oil from a single borehole. The point being that difficult-to-measure large-scale properties can be statistically predictable based on simple-to-measure local properties.

The work within graph theory focused on whether or not graphs had some property, P, however, it did not ask about the computational difficulty of solving the associated decision problem. In contrast, the work within Artificial Intelligence (AI) has focused on the computational hardness of the decision problems. The interest of AI arose from a paradox arising in early studies of propositional satisfiability (SAT). (AI tends to focus on logical decision problems such as SAT, in contrast to the OR focus on optimisation

problems such as the travelling salesman problem, TSP.) In order to develop better SAT solvers, AI practitioners wanted to have a large suite of test instances. However, initial studies on randomly generated instances found, that average solution times scaled polynomially with the problem size. This was somewhat paradoxical, as SAT is NP-complete, and so of course worst case solution times are expected to be exponential, and so hard instances must exist. The seminal work "Where the Really Hard Problems Are" [12] answered that they are associated with thresholds, and provided the foundations for over a decade of active study within AI.

An intensively-studied example is that of "Random 3-SAT" [31]; with n boolean variables c non-trivial clauses are generated (uniformly at random) from all possible clauses. There is a phase transition at $c/n \approx 4.2$. Instances with $c < 4.2n$ are almost always satisfiable, and those with $c > 4.2n$ always unsatisfiable. Furthermore, the computational time needed to decide satisfiability exhibits what is generally referred to as an "easy-hard-easy" pattern:

> **"Easy"**: "under-constrained region", with $c < 4.2n$ and outside of the phase transition, instances are always satisfiable and, on average, it is easy to find a solution to demonstrate this.
>
> **"Hard"**: "critically constrained region", with $c \approx 4.2n$ we are in the phase transition meaning there is a mix of satisfiable and unsatisfiable instances. Furthermore, it is hard to solve the instances, even for the best heuristic (local search) solvers it appears that runtimes scale exponentially with problem size [15, 24].
>
> **"Easy"**: "over-constrained region", with $c > 4.2n$ and outside the phase transition region, then instances are unsatisfiable, and as c increases they become ever easier to prove unsatisfiable with systematic complete solvers. With more constraints branches of the search tree are more likely to be pruned earlier, and so the search tree size is greatly reduced.

When the property P is NP-hard then the easy-hard-easy pattern is common. However, it should be emphasised that even when P is NP-hard it can be that the threshold is still relatively easy. An example, is that of random graphs with P being whether or not the graph has a Hamiltonian cycle. There is a threshold that was initially thought to be hard but was later found to be relatively easy for improved algorithms [37]. Generally, the work in AI has focused on empirical studies of problems with strong hardness peaks. In contrast, historically, the work on random graphs was mostly

mathematical, and for problems with "poly-time" decidable properties (such as connectivity); these seem to often be amenable to exact mathematical analysis. Generally the hard thresholds are much more recalcitrant and so far can only be partially mathematically analysed [1], though with the notable exception of number partitioning [22, 9].

From the OR perspective this sort of pattern might seem familiar from optimisation problems. Suppose, for concreteness, we are studying some instance of the TSP and using a systematic branch-and-bound or integer programming method. We can think in terms of a decision problem working with a varying bound B on the length of a tour; if the (unknown) optimal is d then there is again a pattern:

> **"Easy"**: $B > d$ meta-heuristics easily find solutions
>
> **"Hard"**: $B \approx d$ meta-heuristics fail to find solutions, and branch and bound suffers from large search trees
>
> **"Easy"**: $B < d$ lower bound techniques might well render it easy to prove infeasibility

These "AI easy-hard-easy" and "OR easy-hard-easy" are related; but there is a vital difference. In the "AI case" we are discussing collections ("ensembles") of instances – the easiness or hardness is an average over a collection of instances drawn from a statistical distribution and varying a parameter used to generate them. In contrast, in the "OR case", we are thinking of a single instance and varying a parameter used to control the solution quality.[1]

It is this switch to 'decision problem for each of an ensemble of instances', as opposed to 'optimisation of a single instance', that distinguishes threshold studies from the familiar hardness of approaching optimal values. Such a distinction is critical for this paper. We will be generating many thousands of instances according to various control patterns, and then solve a decision problem for each one. We do not focus on working with a few instances and then heavily optimising each one. But then why are we interested in thresholds and easy-hard-easy patterns at all? After all, such patterns are now ubiquitous and so simple presentation of "yet another hard threshold" would neither be particularly informative or surprising. What distinguishes our work is the usage and implications we draw from the hard thresholds.

[1] Such different views can be connected by viewing instances as not being selected, but rather as "evolving" as we vary the parameter, and then they all become hard "at the same time", but such a link is not needed here.

In AI work, hard thresholds were studied for their own intrinsic interest, however, their actual usage was almost exclusively[2] indirect, and limited to being a source of many hard and suitably-sized problems for driving forward the development of solution algorithms. For example, they were heavily used for SAT solvers, however as solvers improved they become more able to exploit structures within real instances and so benchmarks moved away from the previously used thresholds.

In contrast, we are proposing a case in which a hard threshold in itself can have a direct practical impact. We will see that the thresholds also have a novel and direct relevance in and of themselves; namely, they can have an impact on the short and medium-term planning process.

1.2 Splitting

Now we return to the topic of "splitting". In [6], events to be allocated were *atomic* in the sense that the total *seat-demand* for that event should be allocated to a single room and a time-slot and events could not be split. In [4], we considered the case when classes need to be split into groups (sections). For example, when a lecture class is too large to be accommodated in any of the available rooms or when a class has to be subdivided into tutorial groups. We investigated constraints on numbers and sizes of groups, and found they can drive down utilisation.

The literature on academic timetabling treats many different systems and cultures and so not surprisingly there is no single terminology. However, it is important to appreciate the difference between the following two approaches to modelling the divisions of events into groups:

> **splitting:** the only decisions are the number and sizes of the groups [33, 26, 10, 23, 30, 4]. For example, a tutorial of size 100 might be split into 4 groups of size 25 each, and these groups allocated a room and time in the standard fashion.
>
> **grouping or sectioning:** This addresses the assignment of students (or small student cohorts) themselves to groups (or sections). Sectioning models applied to real world instances either need to be solved in stages [19, 17, 2], or require model-specific meta-heuristics [18, 3, 28].

We do *not* work at the level of students, and so are doing splitting rather than grouping or sectioning. We focus on the interaction of splitting with management and planning, and how to achieve higher utilisations.

[2] A possible exception is the uses of thresholds as attempts to exploit and predict the hardness of instances with the view to improving solvers (e.g. [27])

Generally, previous work has only studied some aspects of the overall problem, but not their interaction. An early version of course planning by Longworth-Smith [33] took account of space resources: It looked at effects of uncertainty in the enrollments into different courses and the resulting probability that a set of rooms will be sufficient. However, it did not consider timetable clashes or other standard timetable issues. Course planning has also been based on methods to predict enrollments into courses [26], and used the predictions to plan which courses to offer.

The effects of splitting on the minimal number of timeslots needed for a legal timetable was considered by Selim [30], and rule-based methods given to decide which courses ought to be split. It looked at a case where the number of available timeslots was smaller than the chromatic number of the conflict graph, and resolved this by splitting some events into multiple sections. Similarly, Tripathy [36] has discussed how multiple sections are needed so as to meet limits on group size but can also result in the resolution of timetable conflicts.

Mathematical models have been given to estimate student enrollments [10] and used to generate number of sections to be the minimal adequate number subject to an upper limit on the size of each section. This work also performed timetabling and assignment of faculty to sections. Splitting has also been performed "in reverse" [23] by placing students into small groups based on similarities in their enrolments, and then finding ways to combine these small groups into sections, that is, allowing different groups of students to be combined during timetabling so as to share lectures that they have in common. Integer programming formulations (for example [13]) have been given for solving complicated timetabling problems in the presence of multiple sections.

However, in such previous work, although search was used for the timetabling, the numbers of sections and their sizes were assumed to be known prior to the search. For example, the number and sizes of sections were supplied, or derived using fixed rules provided by the administration. In contrast, in our previous work [4], and this current work, search is used for the splitting decisions together with the timetabling itself, allowing the two to interact. Also, we look at the interaction between splitting and utilisation in order to develop methods to do space planning in the presence of the need for splitting and sectioning.

Finally, we emphasize that the point of this paper is to study the combination of all these issues. In particular, to show that the thresholds and hardness peaks familiar in AI, have implications for space planning.

Overall, our approach is as follows:

1. In the presence of appropriate constraints, identify the "critical points" of target utilisations U_C. That is, we bound the regions for which utilisations are almost always achievable.

2. Upon identifying critical points, investigate the computational costs of achieving different levels of utilisation as we approach the critical value.

3. Study how the patterns of computational costs will impact on space management and planning

The outline of this paper is as follows. Section 2 gives a description and an integer programming formulation of the teaching space allocation problem with splitting. Section 3 reviews the safety concepts and defines the approach in finding the critical point. In sections 4 and 5 we present experimental results on the thresholds. Section 6 gives conclusions and discusses future work.

2 Description and Formulation

In this section, we present some of the terminology that we use to describe instances, followed by an integer programming formulation of the constraints and objectives.

Unfortunately, the terminology associated with course timetabling and splitting can vary depending on the country, and even the institution. In this paper, a "course" will typically last for 2-4 years and consist of many "modules" each of which takes one teaching session (term or semester). For example, a course in "Computer Science" will likely have modules such as "Computer Architecture". Each module can consist of various types of class: lectures, and potentially tutorials and workshops. Splitting of a class will result in one or more groups. Classes have an "event-type" and will also have an associated "spacetype", i.e. the preferred type of space in which they are to be taught.

We use the same data-sets, from a University in Australia, as in [6, 4]. The set of event and spacetypes are Lecture (LEC), Tutorial (TUT), and Workshop (WKSP). We are concerned with splitting driven by small group teaching requirements, and so will only use the WKSP and TUT data-sets. Furthermore, we will force the event-type of a class to be the same as that of the room, and we will treat spacetypes TUT and WKSP entirely separately. (We are actively investigating the important problem of spacetype mixing and intend to report on it soon [7].) For each spacetype (and associated

event-type) we have sets of classes and rooms. For each class $i \in \{1, \ldots, n\}$ we have:

1. size S_i: number of students of the class
2. department d_i: department offering/managing the class

Similarly, for every room $j \in \{1, \ldots, r\}$ we have:

1. capacity C_j: maximum number of students in the room
2. timeslots T_j: the number of timeslots the room is available per week
3. department D_j: the one that owns/administers the room.

For the resulting assignment problem, we always enforce the standard constraints that room capacities cannot be exceeded, and that different events cannot share a room and timeslot. We also need to consider other important real-world constraints and objectives, notably the utilisation and restrictions on group sizes.

The utilisation measure [5, 6] is the number of seat-hours used (in a weekly schedule) as a fraction of all those available.

$$U := \frac{\text{seat-hours used}}{\text{total capacity for seat-hours}} \tag{1}$$

Usually we refer to U as a percentage: so $U = 100\%$ if and only if every seat is filled at every available timeslot. As discussed earlier, in course timetabling problem the utilisation is fixed by the instance and not the details of the solution (as long as one exists). Accordingly, directly optimising utilisation in itself is not meaningful. Instead, following [6], we use two separate modes:

> **Fixed Choice.** The solver is not given the freedom of choosing which module to allocate to rooms. Instead, the question that generally arises in that case is "Can we fully allocate all the modules to the rooms?"
>
> **Free Choice.** The solver selects which subset of the modules to allocate so as to maximise utilisation.

The free choice mode allows the solver to choose classes that better fits in available roomslots. In this mode we also always impose "No Partial Allocation" (NPA): even if a class is split, either all the resulting groups are allocated, or none at all. (NPA is automatically satisfied in fixed choice

mode.) The roles played by fixed and free choice will, we hope, become clearer in section 3.

Besides utilisation, we also take account of the penalties on locations and group sizes and numbers.

We use a location penalty, to model the desire to minimise physical travel distances [6]. If a group of a class i has a different department than the room j to which it is assigned, then it will be given a penalty L_{ij} (which we take to be a fixed matrix that depends only on the department, d_i, offering the class i, and the department, D_j owning the room j). The total Location penalty, L, is the sum of this penalty over all assignments.

Administrators are likely to want to restrict the number of students in groups, and so we will have "Groups Size (GZ)" penalties and constraints. As mentioned in [4], reducing the number of groups directly reduces teacher hours needed, hence, the "Group Number (GN)" penalty will be proportional to the total number of allocated groups. Note that we are not including conflict constraints between groups. In [4] we discussed the issue of the "partial inheritance" of conflicts: that is, the extent to which a group inherits the conflicts of its parent class. In this paper, we follow the general belief that the partial inheritance is effectively zero, that is, if two classs conflict then with good assignment of students to groups (sectioning) the conflicts can be removed. Accordingly, in this paper, we do not include timetabling conflicts. (In ongoing work, we are using a model that explicitly includes studies in order to investigate under which circumstances the partial inheritance is not zero. Also, teachers can generate conflicts, and for these the sectioning will not necessarily reduce conflicts in the same way as for the students. Future work will include such effects into the study of utilisation.)

2.1 Integer Programming Formulation

The following parameters and variables are used for modeling the problem as an Integer Program. Let's call a (room, timeslot) pair a room-slot. Roomslots are the available space to which classes/groups are allocated. So having r rooms and p timeslots per room, the number of roomslots will be $m = r \times p$. Other sets used are:

N : set of all classes, with total number of classes, $n = |N|$

P : set of all timeslots, with total number of timeslots, $p = |P|$

R : set of all rooms, with total number of rooms, $r = |R|$

M : set of all roomslots, with total number of roomslots, $m = |M| = rp$

Further parameters needed for each spacetype are:

S_i : number of students enrolled in class i

C_j : capacity of room (or roomslot) j
L_{ij}: location matrix between classes i and room (or roomslot) j
G_i^t: target group size for class i
G_i^{low}: lower limit on group size for class i
G_i^{up}: upper limit on group size for class i
G_i^{nb}: upper limit on number of groups of class i
O^l : minimum fraction (occupancy) of seats filled in used rooms

Notice that we will abuse notation slightly and use index "j" for both room and roomslot, for example the location matrix L_{ij} is defined over rooms, but trivially extended to also apply to roomslots; and similarly for the capacity C_j. The difference should be clear from the context.

Finally, we also have parameters to impose non-negative upper limits on the penalties:

B_L^{up} : upper limits on the location penalty (L)
B_{GZ}^{up} : upper limit on group size (GZ) penalty

For example, $B_L^{up} = \infty$ will correspond to no limit on locations, whereas $B_L^{up} = 0$ will force that all locations are perfect matches.

Decision variables: We use the following integer non-negative decision variables:

$$v_{ij} = \text{number of students of class } i \text{ allocated to roomslot } j$$

$$y_{ij} = \begin{cases} 1 & \text{if one group of class } i \text{ is allocated to roomslot } j. \\ 0 & \text{otherwise} \end{cases}$$

$$x_i = \begin{cases} 1 & \text{if class } i \text{ is allocated} \\ 0 & \text{otherwise} \end{cases}$$

Constraints: We have the following constraints.

$$\sum_{j=1}^{m} v_{ij} = S_i x_i \qquad \forall i \tag{2}$$

$$v_{ij} \leq C_j y_{ij}, \quad \forall i \in N, j \in M; \tag{3}$$

$$\sum_{i=1}^{n} y_{ij} \leq 1, \quad \forall j \in M \tag{4}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} L_{ij} y_{ij} \leq B_L^{up} \tag{5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left|v_{ij}-G_i^t y_{ij}\right| \leq B_{GZ}^{up} \tag{6}$$

$$v_{ij} \geq O^l C_j y_{ij}, \quad \forall i \in N, j \in M \tag{7}$$

The No-Partial allocation, NPA, constraint is forced by (2). Room capacities are enforced by (3), which also links the v and y decision variables. Equation (4) limits solutions to at most one group per roomslot. Equations (5) and (6) place upper bounds on the location (L) and group size (GZ) penalties. (Only occupied roomslots will contribute to GZ, and the intent is to drive group sizes to the target size. When B_{GZ}^{up} is 0, all groups must have a size equal to τ_i). If a roomslot is used then the given fraction O^l of room seats needs to be filled due to (7). In this paper, we use $O^l = 0.3$, though other values gave very similar results to those presented here. In some cases, we also used some or all of the following:

$$v_{ij} \leq G_i^{up} y_{ij} \qquad \forall i \in N, j \in M \tag{8}$$

$$v_{ij} \geq G_i^{low} y_{ij} \qquad \forall i \in N, j \in M \tag{9}$$

$$\sum_{j=1}^{m} y_{ij} \leq G_i^{nb}, \quad \forall i \in N \tag{10}$$

Which impose upper and lower limits on the group sizes using (8) and (9). Constraint (10) imposes upper limits on the number of groups per class. Note that if $G_i^{nb} = 1$, the problem becomes pure teaching space allocation without splitting [6].

The following constraint simply expresses that the total number of assignments is no greater than the number of total roomslots, $m = rp$. It is entailed by the other constraints, but is added as it can lead to a considerable reduction in computation times.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \leq m \tag{11}$$

Objectives: In free choice mode the objective is to maximise the overall seat-hours used

$$Obj^* = \left(\sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij}\right) \tag{12}$$

However, for fixed choice mode we enforce $\forall i.\ x_i = 1$ giving

$$\sum_{j=1}^{m} v_{ij} = S_i \qquad \forall i \tag{13}$$

In this case, the value of the objective Obj^*, and so the utilisation, is fixed, and the problem is simply that of feasibility. We note that it is straightforward to show that the resulting splitting problem is NP-hard by reduction from subset sum. This NP-hardness underlies the hardness of the problem. However, more intuitively, notice that the number of choices for splitting can be large when the initial class size is large. The primary decision variable, v_{ij}, is integer and not just a 0-1 variable, and so can have a large range of potential values. Also, each splitting choice interacts with whether or not a room assignment can be found, contributing further to the hardness of the problem.

Finally, the solver we use is OPL with CPLEX 10.0 from ILOG[3].

3 Achievement Curves

[Figure 1 about here.]

In this section, we look at achievement curves and thresholds [5, 6] for the case of the splitting problem. Given a fixed set of rooms and a set of classes N of a given spacetype, and hence a corresponding demand SH_e for seat-hours, one can ask the following:

> Given a demand, SH_e, (for total "seat hours"), and taking account of the constraints, is there a splitting and allocation that allows full satisfaction of the demand?

Our key tool, "Achievement Curves", study how the probability of the answer being "Yes, all are allocatable" varies with the demand.

In our data-sets, the total seat-hours in the set of classes is significantly larger than in seat-hours available in the rooms. Hence, we can answer this by generating random instances by taking subsets of the classes using the following procedure:

Proc: subset-scan
1 Given a set N of classes of a given spacetype
2 For ν= 1 to 3000 (or similar large number)
3 For s= 1 to n
4 Randomly select a subset of N containing s different classes from set N
5 Run the solver in free choice mode on the subset to obtain U_A

[3] http://www.ilog.fr/

That is, we generate many random subsets of the classes. For each subset we compute the utilisation if all were to be events allocated; we call this the "requested" utilisation U_R. Running the solver in free choice mode optimises the set of allocated classes and gives the "Achieved" utilisation U_A. If all classes can be allocated then $U_A = U_R$, however, in general U_A can be lower. Hence, each randomly generated subset generates a point, with coordinates (U_R, U_A). An example is given in figure 1(a), in which we give the achievement curves with two different upper limits on the location penalty, $B_L^{up} = 800, 1200$, with resulting critical utilisations $U_C \approx 0.5$ and $U_C \approx 0.6$.

Alternatively, it can be clearer to present the "fractional achievement", or "fill factor"

$$K_f = \frac{U_A}{U_R} \tag{14}$$

against U_R. An example is given in figure 1(b). In this case, we are will be interested in the quantity

> $Pr(K_f = 1)$: probability of achieving full allocation, $K_f = 1$, on random choice of classes with given U_R

and in particular how it varies with the requested utilisation U_R.

The results were obtained using CPLEX as the solver. CPLEX is given a "MIP gap" of 1% meaning it only stops when it proves the current solution is within 1% of optimal. Hence we can be sure that the fall-off in achieved utilisation is not a result of any deficiency in the solver.

As observed in [6] the main features of these curves are their statistical predictability and the threshold behaviour. At a given value of U_R the achieved values do not vary widely but are rather closely clustered and so one can say that the expected U_A is fairly predictable. The threshold behaviour is the observation that the achievement curve exhibit a critical utilisation U_C that separates the under constrained region $U_R < U_C$ and over-constrained region $U_R > U_C$.

In terms of space management we might regard the "under-constrained" region $U_R < U_c$ as the "safe" region because the requests are very likely to be satisfied. In contrast, the critical and over-constrained regions, $U_R \geq U_C$, are considered "unsafe". The intended usage of such results is that administrators should aim to be in the safe region so as to be confident to satisfy demand, but as close as possible to the critical region so as to maximise utilisation.

4 Computational Hardness

[Figure 2 about here.]

As discussed earlier, sharp thresholds (or phase transitions) as seen in the previous section, are common in physical and combinatorial systems and are often associated with a peak in the computational difficulty of solving each instance [12, 14, 16, 27, 32]. In general, we should expect an "easy-hard-easy" pattern, corresponding to the under-, critical-, and over-constrained regions. In this section, we present results investigating the hardness of our integer programming formulation of splitting problems.

The experimental methodology is straightforward; for each instance, we simply record the time taken when the instances is solved: classifying it as satisfiable, "sat", or unsatisfiable, "unsat". The one catch is that sometimes the CPLEX solver times out and leaves the instances undetermined "undet". Having obtained runtimes for each instances we produce histograms based on collecting instances together into bins according to their requested utilisation U_R, and also according to their status of sat, unsat, or undet.

This data is then used to produce plots such as Figure 2(a). Plots similar or equivalent to this are used in the artificial intelligence literature on phase transitions, but are perhaps unfamiliar in an operational research context, so we will now explain how to read and interpret them. For the purposes of clarity we will focus on figure 2(a), as later plots are similar. As for the achievement curves of figure 1, the x-axis is the requested utilisation, U_R, which is the utilisation that would be achieved if all classes can be successfully assigned to a roomslot.

Figure 2(a) then combines two sets of data measuring, as a function of U_R, firstly, the likelihood that all classes can be assigned, and secondly the runtime needed to solve the assignment problem. There are two corresponding sets of y-axes. The left-hand y axis is associated with the points in the lower half of the figure, and gives the probability, $Pr(K_f = 1)$, that a feasible solution exists with every class allocated a room and time. The right-hand y axis is associated with the points in the upper half of the figure, and gives the runtimes needed by the solver; notice that this axis is on a logarithmic scale because the runtimes can vary so over a wide range. More specifically, the figure is actually a fine-grained histogram with respect to U_R. Each point is the aggregation of results from a sample of many instances (each instance being a set of classes) from a small region around that value of U_R. The given probability is the fraction of such instances that were satisfiable. The points for the runtimes are the (arithmetic) mean runtime within the

sample, along with the quartiles given as "error-bars" and intended to show the variation of runtimes between different instances.

The first feature to extract from such figures is the position of the transition region; the region of U_R at which the probability drops from one to zero. The second feature to extract is the behaviour of the runtime as one approaches and passes through the transition. That is, to look at the upper runtime curves in relation to the lower probability curve (which is why it is useful to combine them into a single figure). For this interpretation of the runtimes it is the general trend of the runtimes that is important, and not details of individual points in the figures.

The results for runtimes as a function of U_R are given in Figure 2(a) for the WKSP data set and without imposing any limits on the location and group size penalties. As explained above, we also give experimental results for the probability of full allocation, $Pr(K_f = 1)$, and show a sharp threshold at $U_C \approx 92\%$. Note again that the run-time axis is logarithmic: instances at $U_R = 90\%$ take 10-100 times as long as those at $U_R = 80\%$. The pattern is:

> **Easy:** $U_R < U_C$, there are many different ways to fully allocate groups and so it is easy to find a solution
>
> **Hard:** $U_R \approx U_C$, There might be few solutions, and they are hard to find. Alternatively, the problem is unsatisfiable but there is no simple reason for this.
>
> **Easy:** $U_R > U_C$, full allocation is impossible and it can be straightforward prove this (e.g. from the linear relaxation)

Below the threshold, within any instance there are many solutions and so it is easy to find one. For $U \geq 100\%$ the linear relaxation will detect infeasibility and so the runtime will be small as branch-and-bound search is not needed. Around the threshold, there are few if any solutions, and so the search will pursue many failed branches and take a correspondingly longer time on average for each instance.

Figure 2(b) gives the results of a similar investigation of the TUT dataset. The outcome is similar to the Workshop case of figure 2, but the threshold is closer to 100%. We believe this is because the smaller rooms help achieve near optimal allocations compared to the larger rooms in the WKSP dataset. However both databases exhibit the same behaviour in terms of time requirements.

[Figure 3 about here.]

In the same fashion as figure 2, in figure 3 we show the effects of imposing an upper limit on the location penalty, $B_L^{up} = 800$. For both the WKSP and TUT datasets we see that there is still a clear threshold, however it is now at a significantly smaller critical utilisation: reduced to about 45% for WKSP, and 60% for TUT. These are consistent with the results in [6] that the need to reduce location penalties has the potential to significantly drive down utilisations. The computational hardness, and easy-hard-easy pattern, are similar to before.

[Figure 4 about here.]

For tutorials, it is usual that they are intended to be taught in small groups. We now look at the effects on the thresholds of two different ways to enforce the small group requirement.

In Figure 4 we see the effects of imposing for each group that the size must lie within the range 12,...,16. (That is, we use $G^{low} = 12$ and $G^{up} = 16$). Again there is a threshold, and the group size constraints have substantially reduced the achievable utilisation. However, it is also apparent that the hardness peak has essentially disappeared. We remark that is well-known that whilst thresholds are often associated with hardness peaks, it is not the case all thresholds automatically have a hardness peak [37]. These results suggest that enforcing small group sizes can easily lead to a large loss of utilisation. Presumably, any remedy to this will involve modifying the room sizes so as to be a better match. (Studies of such room sizes profiles are part of our ongoing research, and will be reported elsewhere.)

5 Runtimes for a stochastic solver

[Figure 5 about here.]

So far we have only considered a single (and exact) solution method, namely our integer encoding together with CPLEX, but hardness peaks at thresholds generally affect many algorithms (and maybe all). Hence, we also studied the 'Simulated Annealing for Splitting', SAS, algorithm of [4]. Figure 5(a) plots the run-times required by CPLEX and SAS on the pure splitting case, that is, without any limits on the location or group size penalties. (For clarity, the "errors bars" for quartiles are omitted in this case.) Overall, the SAS runtimes show the expected rapid increase as we approach the threshold. However, the pattern is significantly different from

that of CPLEX. In the under-constrained region the SAS beats CPLEX, though at the cost of a much greater variability in runtimes.

Close to the threshold, the SAS performs much worse than CPLEX, and generally fails. This relative performance of systematic and stochastic search is fairly typical of thresholds. This is reinforced in Figure 5(b) which shows that after $U_R \approx 80\%$, SAS almost always fails. This means that the simulated annealing exhibits a "False Threshold" which is clearly at a lower utilisation than the true threshold, which is revealed by the CPLEX results in Figure 5(b) to be around $U_R \approx 92\%$. Better versions of local search are likely to improve this, however, it is quite possible that they will still suffer from thresholds below the true threshold (e.g. see [24]). The impact of such false thresholds is that they could mislead space planners into settling for lower levels of utilisation than are potentially possible with a better solver.

[Figure 6 about here.]

Figure 6(b) plots the runtimes required by CPLEX and SAS in the presence of the location penalty for the WKSP dataset. Again, the SAS solver solves some instances quicker then CPLEX, but closer to the threshold it does worse.

The implication for space planning is that whilst false thresholds can occur, it can also happen that they do not occur (or do occur but do not have a significant impact). Hence, it would be good to know when they are, or are not, likely to occur.

6 Conclusions and Future Work

The problem of space planning and management in the context of requirements such as splitting classess into groups in the 'Teaching Space Allocation Problem' is of considerable importance for various academic institutions. In previous work [5, 6, 4], we introduced the notion of achievement curves as a tool to study the levels of utilisation that can be reliably achieved. These curves then revealed the underlying threshold phenomena. For example, in [4], we studied the thresholds in the presence of splitting, and found that different penalties and constraints have significant effects on the utilisation that can safely be achieved.

However, practically, it is also important to know the computational effort (i.e. runtimes of solvers) that are needed in order to reach the target utilisations. Accordingly, we extended the previous work by investigating the runtimes needed to solve the problems arising. Here, we have shown that

there is a standard easy-hard-easy pattern in the hardness, with the problems near the threshold being much harder. The results are exemplified by Figure 2(a). Whilst some utilisation levels are easily achievable, some levels require high computational resources. For example, a 5% improvement of utilisation might well require a hundred-fold increase in computational time. The dramatic increase in runtimes at the threshold implies that better solution methods will be required if we are to approach the threshold on larger data sets. Also, a particular concern for space planners is that "false thresholds" can arise. If only incomplete solvers are studied then (by definition) instances cannot be proven unsatisfiable, and so no non-trivial upper limit can be obtained on the position of the threshold. The resulting apparent limit can then be substantially below the true threshold. (Though, arguably, even if some level of utilisation is achievable with a very good splitting and allocation, it is not of much use if the solver is incapable of finding it.)

We conclude that there is a close link between improving utilisation and improving solver technologies. This paper has shown, that attempts, in space management, to achieve acceptable utilisation figures beyond the normal standard ones seen in academic establishments might well require improvements in solvers. Note that the hardness peak occurred for two different algorithms giving strength to believing that the hardness peaks are real. It can be that a specialised algorithm can make hard peaks disappear (as happened in [37]), but even if these were to be the case here the results are still important because the administrator or planner needs to be aware of effects of the choice of solver on the achievable utilisation.

This paper also provides a novel role for the easy-hard-easy patterns that have been so extensively studied with Artificial Intelligence. Not only do they provide a source of challenging instances for algorithm development, but they can also have a direct interaction with practical issues in space planning and management.

There are many directions for future work. For example, in the real world, there is often "spacetype mixing": events are have a primary spacetype, but can be allocated to a secondary spacetype when needed. In this paper, we have investigated spacetypes independently, but an important direction for future study is to also allow spacetype mixing. For example to allow tutorials to use a lecture room when necessary. Such spacetype mixing is likely to significantly affect utilisation. We are actively studying this issue on the basis of extensions to the models and methods presented in this paper, and we intend to report on it soon [7].

Acknowledgments: Andrew Parkes has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant GR/T26115/01.

References

[1] D. Achlioptas. A survey of lower bounds for random 3-SAT via differential equations. *Theoretical Computer Science*, 265(1–2):159–185, 2001.

[2] S. M. Al-Yakoob and H. D. Sherali. A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations. *European Journal of Operations Research (EJOR)*, 180:1028–1044, 2007.

[3] R. Alvarez-Valdes, E. Crespo, and J. M. Tamarit. Assigning students to course sections using tabu search. *Annals of Operations Research*, 96:1–16, 2000.

[4] C. Beyrouthy, E. K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes. The teaching space allocation problem with splitting. In E. K. Burke and H. Rudova, editors, *Practice and Theory of Automated Timetabling VI: Revised Selected papers from the 6th international conference, PATAT, Aug 30-Sep 1, 2006*, volume 3867 of *Lecture Notes in Computer Science*, pages 228–247. Springer-Verlag, Brno, Czech Republic, 2007.

[5] C. Beyrouthy, E. K. Burke, J. D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes. Understanding the role of UFOs within space exploitation. In *Proceedings of the 2006 International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 359–362, Brno, Czech Republic, 2006.

[6] C. Beyrouthy, E. K. Burke, J. D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes. Towards improving the utilisation of university teaching space. *Journal of the Operational Research Society (JORS)*, Nov. 2007. Online version: http://dx.doi.org/10.1057/palgrave.jors.2602523.

[7] C. B. Beyrouthy, E. K. Burke, B. McCollum, P. McMullan, and A. J. Parkes. Interactions of spacetype mixing and profiles with teaching space utilisation. In Preparation, 2008.

[8] B. Bollobas. *Random Graphs*. Academic Press, London, England, 1985.

[9] C. Borgs, J. Chayes, and B. Pittel. Phase transition and finite-size scaling for the integer partitioning problem. *Random Structures and Algorithms*, 19(3-4):247–288, 2001.

[10] J. Boronico. Quantitative modeling and technology driven departmental course scheduling. *Omega*, 28(3):327–346, June 2000.

[11] E. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research (EJOR)*, 140(2):266–280, 2002.

[12] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sydney, Australia*, pages 331–337, 1991.

[13] S. Daskalaki, T. Birbas, and E. Housos. Efficient solutions for a university timetabling problem through integer programming. *Eur. J. Op. Res.*, 160:106–120, 2005.

[14] W. Erben. A grouping genetic algorithm for graph colouring and exam timetabling. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, pages 132–158, London, UK, 2001. Springer-Verlag.

[15] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The scaling of search cost. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 315–320. American Association for Artificial Intelligence, 1997.

[16] I. P. Gent and T. Walsh. Phase transitions and annealed theories: Number partitioning as a case study. In *European Conference on Artificial Intelligence*, pages 170–174, 1996.

[17] A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research (EJOR)*, 54(1):39–47, Sept. 1991.

[18] A. Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35(3):255–270, 1992.

[19] G. Laporte and S. Desroches. The problem of assigning students to course sections in a large engineering school. *Computers & Operations Research*, 13(4):387–394, 1986.

[20] B. McCollum and P. McMullan. The cornerstone of effective management and planning of space. Technical report, Realtime Solutions Ltd, Belfast, Jan 2004. http://www.realtimesolutions-uk.com/SpaceManagement.doc.

[21] B. McCollum and T. Roche. Scenarios for allocation of space. Technical report, Realtime Solutions Ltd, Belfast, 2004. http://www.realtimesolutions-uk.com/Scenarios.doc.

[22] S. Mertens. Random costs in combinatorial optimization. *Phys. Rev. Lett.*, 84(6):1347–1350, Feb 2000.

[23] S. K. Mirrazavi, S. J. Mardle, and M. Tamiz. A two-phase multiple

objective approach to university timetabling utilising optimisation and evolutionary solution methodologies. *Journal of the Operational Research Society (JORS)*, 54(11):1155–1166, 2003.

[24] A. J. Parkes. Easy predictions for the easy-hard-easy transition. In *Eighteenth national conference on Artificial intelligence (AAAI02)*, pages 688–694, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[25] S. Petrovic and E. K. Burke. University timetabling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 45, pages 1–23. Chapman Hall/CRC Press, Boca Raton, FL, 2004.

[26] B. Ram, S. Sarin, and A. Mallik. A methodology for projecting course demands in academic programs. *Computers & Industrial Engineering*, 12(2):99–103, 1987.

[27] P. Ross, D. Corne, and H. Terashima-Marin. The phase transition niche for evolutionary algorithms in timetabling. In E. K. Burke and M. A. Trick, editors, *Selected papers from the First International Conference on the Theory and Practice of Automated Timetabling (PATAT 95)*, volume 1153, pages 309–324. Lecture Notes in Computer Science, Springer-Verlag, NY, 1996.

[28] S. E. Sampson, J. R. Freeland, and E. N. Weiss. Class scheduling to maximize participant satisfaction. *Interfaces*, 25:30–41, 1995.

[29] A. Schaerf. A survey of automated timetabling. *Artif. Intell. Rev.*, 13(2):87–127, 1999.

[30] S. M. Selim. Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. *The Computer Journal*, 31(1):76–82, 1988.

[31] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[32] B. M. Smith and M. E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, Mar. 1996.

[33] R. L. Smith. Accommodating student demand for courses by varying the classroom-size mix. *Operations Research*, 19:862–874, 1971.

[34] J. Spencer. Survey/expository paper: zero one laws with variable probabilities. *Journal of Symbolic Logic*, 58:1–14, 1993.

[35] D. Stauffer and A. Aharony. *Introduction to Percolation Theory*. Taylor & Francis Ltd, 1994.

[36] A. Tripathy. Computerised decision aid for timetabling: a case analysis. *Discrete Appl. Math.*, 35(3):313–323, 1992.

[37] B. Vandegriend and J. Culberson. The $G_{n,m}$ phase transition is not hard for the Hamiltonian cycle problem. *Journal of Artificial Intelligence Research*, 9:219–245, 1998.

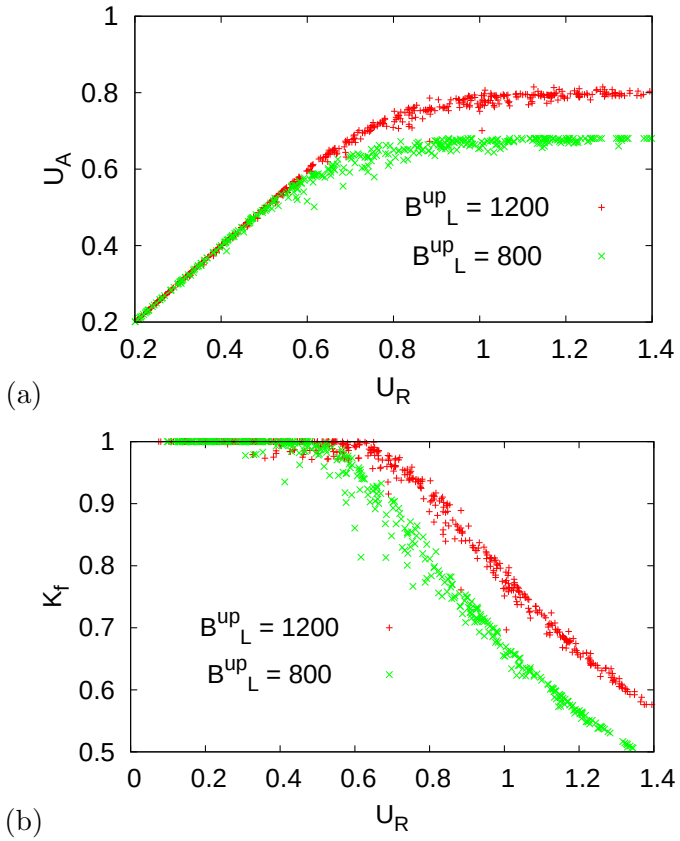


Figure 1: Achievement curves for the WKSP data set with two different upper limits on the location penalty. (a) requested utilisation (U_R) vs. achieved utilisation (U_A) (b) achievement fraction K_f vs. U_R.

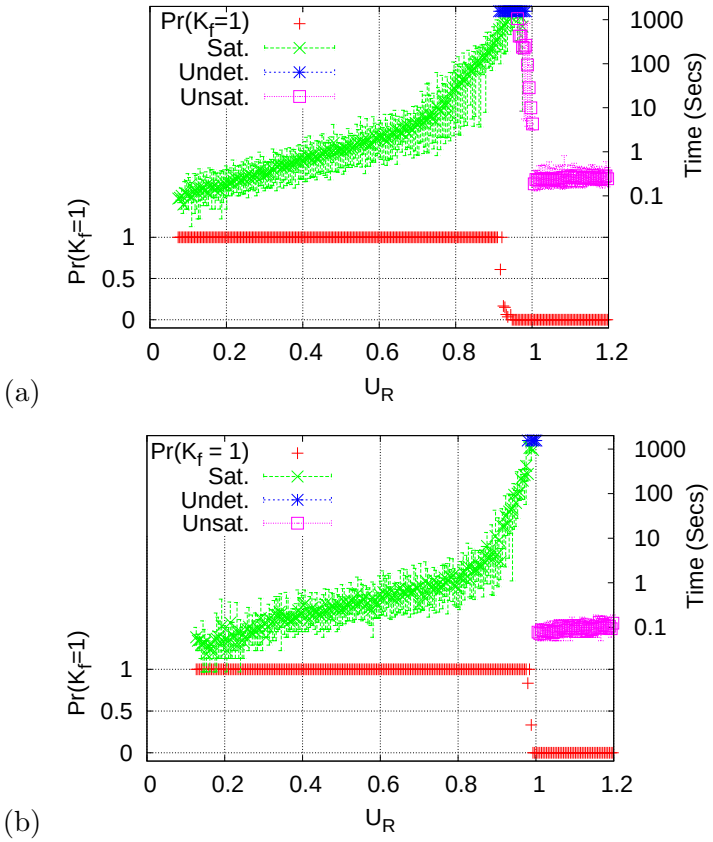


Figure 2: The x-axis is the requested utilisation U_R. The left-hand "y1-axis" is the probability of full allocation, $\Pr(K_f{=}1)$. The right-hand "y2-axis" is the run-time (on a log scale). The average run-times are plotted together with "error bars" representing the lower and upper quartiles of the run-time. For the run-times, the instances are separated into: "Sat", satisfiable; "Unsat", unsatisfiable; and "Undet" for undetermined by the solver. The "Undet" points correspond to the solver reaching the time-limit and so are all at the cap on runtimes of 1200 seconds. For a full explanation of such figures see section 4. Results are with no bounds on location and group size, i.e. $B_L^{up} = B_{GZ}^{up} = \infty$. Data-sets used are (a) WKSP, and (b) TUT.

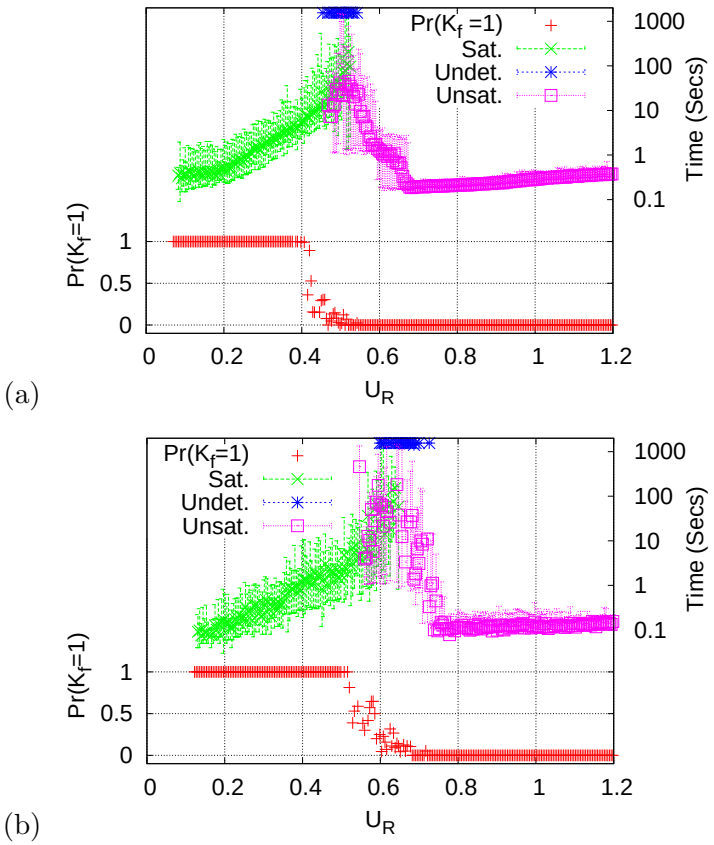


Figure 3: Threshold results, as Figure 2, but with an upper limit on location penalty, $B_L^{up} = 800$. Data-sets: (a) WKSP (b) TUT

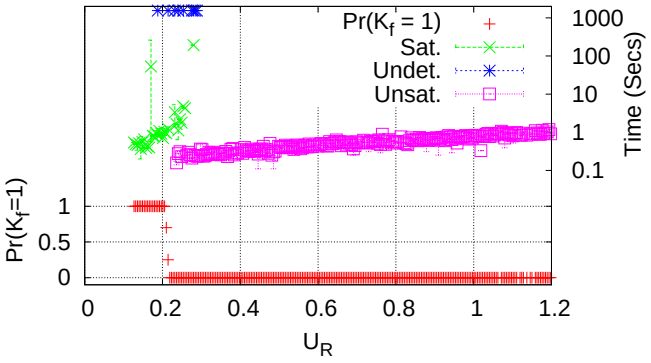


Figure 4: TUT Data-set with $B_L^{up} = 800$ and with group size strictly constrained in the range 12,...,16.

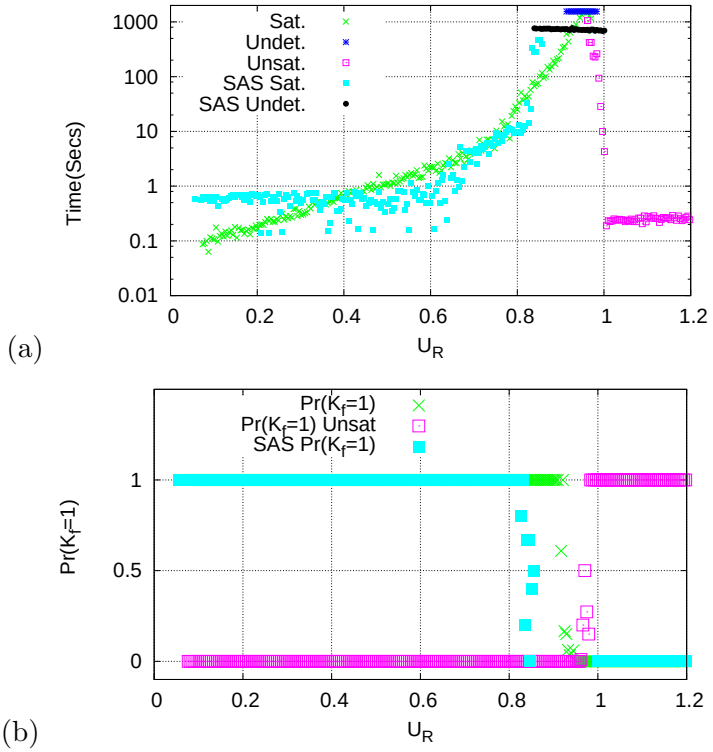
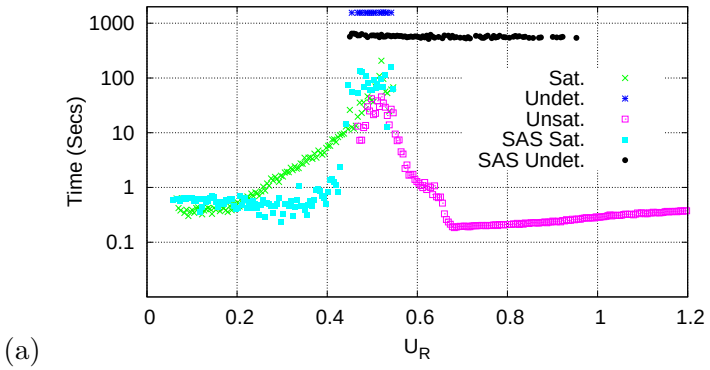
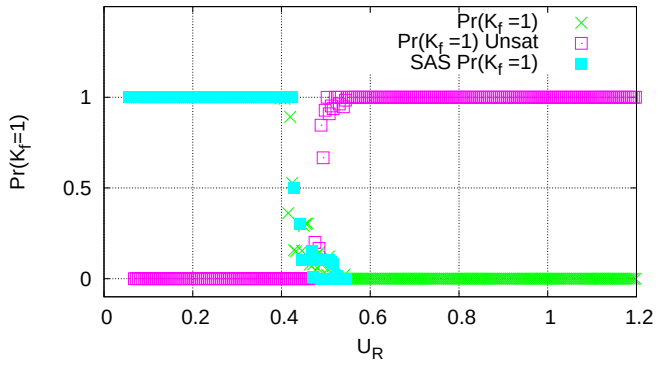


Figure 5: (a) The average run-times of CPLEX on the integer encoding compared to runtimes from the local search method, "SAS". The CPLEX instances are separated into: "Sat", satisfiable; "Unsat", unsatisfiable; and "Undet" for undetermined by the solver. Whereas, "SAS Sat" refers to average runtimes for instances with complete solutions from SAS, and "SAS Undet" for those in which the SAS was unable to find a complete solution with all events allocated. (b) "Pr($K_f = 1$)" gives the fraction of instances found by the CPLEX to be satisfiable in the sense of having a solution with all events allocated. Similarly, "Pr($K_f = 1$) Unsat" gives the fraction proved by CPLEX to be unsatisfiable. Then, "SAS Pr($K_f = 1$)" gives the fraction of instances for which SAS was able to fully allocate all events.



(a)



(b)

Figure 6: The same as figure 5 except that we impose an upper bound on the location penalty, $B_L^{up} = 800$. (a) gives runtimes for SAS and CPLEX, and (b) gives the empirical probabilities of full allocations of all events being found for the instances.