Models, Solution Methods and Threshold Behaviour for the Teaching Space Allocation Problem

by Camille Beyrouthy, BEng, MSc

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy
September 2007

à mes parents, Pierre et Nawal

à ma soeur Maud

Contents

APPENDICES

List of Figures

List of Tables

Abstract

Universities have to manage their teaching space, and plan future needs. Their efforts are frequently hampered by, capital and maintenance costs, on one hand, pedagogical and teaching services on the other. The efficiency of space usage, can be measured by the "utilisation": the percentage of available seat-hours actually used. The observed utilisation, in many institutions, is unacceptably low, and this provides our main underlying motivation: To address and assess some of the major factors that affect teaching space usage in the hope of improving it in practise. Also, when performing space management, managers operate within a limited number and capacity of lecture theatres, tutorial rooms, etc. Hence, some teaching activities require splitting into different groups. For example, lectures being too large to fit in any one room and seminars/tutorials being taught in small groups for good teaching practise. This thesis forms the cornerstone of ongoing research to illuminate issues stemming from poorly utilised space and studies the nature of constraints that underlies those low levels of utilisation. We give quantitative evidence that constraints related to timetabling are major players in pushing down utilisation levels and also, devise "Dynamic Splitting" algorithms to illustrate the effects of splitting on utilisation levels. We showed the existence of threshold between phases where splitting and allocation is "*always possible*" to ones where "*it's never possible*", hence, introducing a practical application of `Phase Transition` to space planning and management. We have also worked on the long-term planning aspect of teaching space and proposed methods to improve the future expected utilisation.

Acknowledgements

Thank you GOD, for you made this venture happen, and cross the thorny tough path of what is a PhD. Thank you for the flood of love and care that got me to where I am now, through the most difficult and enduring times.

I would like to express my most sincere gratitude to my supervisor Professor Edmund Burke. His guidance and support all along my PhD has made it possible to successfully complete this work.

Many thanks to Dr Barry McCollum and the team of RealTime Solutions for their unrelenting help and advice provided through all years of my research.

Deep appreciation to Dr Dario Landa-Silva and Dr Andrew Parkes, for the technical help, encouragement and expertise they have provided. This research would not have been completed without their precious and sound advice.

I would like to thank, the staff and all students of the ASAP Group at the university of Nottingham. Thank you for creating a warm and friendly environment for academic research, specially, Matthieu Basseur, Jaume Bacardit, Ruibin Bai, Timothy Curtois, German Terrazas Angulo, Jason Atkin, Geert de Maere, Antonio Vazquez, Ziad Azar, Helen Newton and Lorna Goodwin.

Finally, I am very grateful to my external examiners, Professor Sanja Petrovic and Dr William Fawcett for their highly valuable comments and suggestions.

Chapter 1

Introduction

Space, like Time, is Money

(NAO[1])

1.1 Background and motivation

The university sector, in the UK, suffers from chronic space *under-usage*. Decisions related to the provision of teaching space are complex, over-constrained, and directly impact the costs and quality of service for students and staff. For example, over-provision of space leads to higher capital, maintenance and running costs, while under-provision leads to poorer service with teaching events and activities being unable to find space or being forced into inappropriate timeslots or locations.

[1]The National Audit Office, from "*Space Management in Higher Education: A Good Practice Guide* (1996)"

Furthermore, there is evidence that the current system is deficient. A simple metric, used by the HEFCE[2], 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 are rather low (around 20-30%) (McCollum and McMullan, 2004; McCollum and Roche, 2004). Paradoxically, 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.

As an example of tools that have been used to relate utilisation and space usage, from a statistical point of view, we can consider the *inefficiency multiplier*[3] studied by SMG[4], which explores the amount of space made available, for every m^2 in use, at a given utilisation level.

According to SMG, a higher utilisation rate implies that less space is being provided, and a lower rate, means more space is provided per m^2 in use. We can infer therefore, that, low utilisation figures lead to a high cost of space in support for a teaching activity, and a lower income ratio per m^2.

This in turn, will imply that an aggressive approach to improve low utilisation levels will have a large cost impact. SMG also reports that:

> "because of the non-linear relationship between utilisation rates and space provision, where utilisation is very low, a small increase can have a significant impact on the amount of space provided and on the attendant cost."

Therefore, acquiring extra space for academic activities is generally met with reluctance from space and estate managers on the basis of poor space management. Hence, an overall

[2]Higher Education Funding Council for England. See www.hefce.ac.uk
[3]Report on *Space utilisation: practice, performance and guidelines*
[4]UK Higher Education Space Management Group. See www.smg.ac.uk

aim of universities is to improve the utilisation of teaching space. Nevertheless there must be limits on what is possible to achieve within the context of practical objectives of timetabling in addition to 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 thesis forms the cornerstone of an ambitious, ongoing project to understand and perhaps predict what utilisations universities nowadays might be expected to reliably achieve, and how to achieve them.

1.1.1 Planning versus timetabling

Before proceeding further, it is important to set the context for the work in this thesis, and in particular, to emphasise the differences from general course or curriculum timetabling.

As is known 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[5]:

- ***Space Planning:*** "Long term" campus or new building design, with time horizon of around 5-50 years. Planning would probably require a new space artchitecture/modelling.

- ***Space Management:*** "Short term" remodelling of existing space, with time horizon

[5]The division is likely to be approximate; phases might well overlap, planning and management are likely to be repeated, etc.

of around 1-5 years, mainly for an "at the time" purpose-fit accomodation.

- ***Course Timetabling:*** "Immediate", allocation of teaching events to times and rooms for the next term or semester. The time horizon is "immediate" because it needs to be done for the next teaching session.

Space Planning is concerned with decisions as to which rooms ought to be built or re-allocated to different tasks. In particular, the long-range nature of space planning implies that decisions need to be made before the exact details of current timetables, student numbers, etc, become available.

Common approaches to deal with this incomplete information are to consider, the so called `space norms`. For example, a norm could be a physical objective such as $5m^2$ per Ph.D. student. That will form the basis for space management and also office space allocation (OSA), and provides a reasonable foundation for space planning, since one can use these norms to estimate the overall demand for office space, and subsequently match this demand by a corresponding rooms supply while factoring in a projected utilisation factor.

While this works well for OSA (because, most offices are all used), attempting to do this for teaching space allocation is more difficult because the expected low utilisation leads planners to build in a corresponding excess capacity. Moreover, expected utilisations are such that this inbuilt "safety margin" has to be as much as a factor of two, or more.

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 teaching events and a set of rooms. From definitions, later in the thesis, it will be clear that utilisation

is obtained by simple counting of seat demand and availability, that is the seat-hours being actually used, and is *fixed* from the outset. One 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* lie on the stages before 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 for the demand (e.g. teaching events/modules) and cannot be predetermined. Hence, we must do at least one of (i) vary total demand, and (ii) vary total capacity.

Most of the approaches in coming chapters will either alter the total demand, keeping room sizes fixed, or alter the spacetype[6] of a given room.

1.1.2 Class splitting

Now we return to the topic of "splitting" of a class. Demand is represented by teaching events to be allocated to a single room and timeslot. The main assumptions are generally that all events will fit in available room. In reality, that is different: course splitting has been driven and motivated by both of the following requirements:

1. Small-Group Splitting: Classes that are intrinsically designed to be taught in small groups, such as seminars or tutorials.

[6] more on that in **chap.** 7

2. Constraint-Driven Splitting: Classes that could in principle be held without splitting, but for which splitting is forced because of other constraints:

(a) capacity constraints: the class is simply too large to fit into one room.

(b) timetable constraints: the enrollment is large and across such a wide spectrum of students, this class will conflict with many other classes, greatly reducing the chances of obtaining a conflict-free timetable. Splitting that class into multiple groups can greatly help ease timetabling pressures, as students are more likely to be able to find a group that is conflict free for them. We elaborate more on this aspect of splitting in **chap.** 6. We have also investigated constraints on numbers and sizes of groups, and found they can also drive down utilisation.

1.2 Scope and aims

This thesis forms the basis of ongoing research on Teaching Space Allocation **(TSA)** funded by the University of Nottingham (through an EPSRC Grant), and RealTime Solutions Ltd. We have collaborated closely with RealTime Solutions, and they have provided us with highly valuable advice, and made available several datasets for experimental usage. Informal conversations with Dr Barry McCollum and Dr Paul McMullan, who have broad knowledge and practical experience on the space planning sector, have helped us extend our vision on the most important issues and difficulties we might face in this investigation. Our main aims in this thesis are to:

1. Understand the main factors that affect space utilisation and identify potential issues.

2. Devise novel ways to do splitting, in order to study it's effects on space usage.

3. In the presence of real-world constraints, identify the "critical points" of target utilisations, where we detect areas where given utilisations are almost always achievable.

4. investigate the computational costs of achieving different levels of utilisation, in the presence of other objectives.

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

6. Study the effects of the timetabling constraints on utilisation in cases where splitting is involved, including a brief study of the interaction between timetabling and splitting.

7. Suggest new models to alter the "spacetype[7]" of rooms in the hope of improving expected utilisations.

1.3 Thesis overview

This thesis is organized as follows:

In chapter 2, we survey and discuss previous research on splitting and sectioning, and expose state-of-the-art models and methodology related to splitting in course timetabling. The nature of course timetabling requires a brief note on existing terminologies, to help explain the models. Some other problems closely related problems to space management and planning are also briefly exposed.

In chapter 3, we introduce the pure teaching space allocation problem, and investigate the effects of key **location** and **timetabling** constraints on utilisation levels. An initial

[7] Detailed in **chap.** 7.

study of the multiobjective nature of the problem, using two and three objectives is also included in this chapter. We show the real-world effects of various constraints on utilisation and space management and this serves as a gentle introduction to models introduced later in the thesis.

In chapter 4, we look at the problem of splitting in teaching space allocation. We provide a first model for splitting and study the effects that splitting, location and timetabling constraints have on the utilisation levels. We consider the optimisation problem of *"what is the maximum achievable utilisation in the presence of some of the constraints"*. This chapter also presents a new *Dynamic Splitting* algorithm in which splitting is performed as part of the local search.

In chapter 5, we look at the decision problem of whether random subsets of teaching events are fully allocatable or not. We found 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 should aid decisions as to how much teaching space to provide. We have studied also the computational hardness at this threshold and provided experimental evidence that the problem becomes intractable at the thresholds, by detecting an *Easy-Hard-Easy*[8] pattern in teaching space allocation.

[8]Thresholds have been extensively studied within Artificial Intelligence (AI), and it is known that such thresholds often exhibit an "Easy-Hard-Easy" pattern: with computational costs increasing rapidly as we approach the threshold

In chapter 6, we give evidence that when splitting occurs, the timetabling conflicts gets relaxed. We introduce the notion of *partial inheritance* to define the extent to which conflicts between events get resolved as we split. The question that naturally arises is: "how do conflicts get transmitted from parent events to respective groups as splitting occurs?". The main assumption being that, a proper assignment of students to groups can lead to canceling the effects of timetabling and relaxing conflicts, resulting in less constraints at the space management level.

In chapter 7, we look at the long-term planning aspect of the teaching space allocation problem: "How can we re-model teaching space, such that the overall critical utilisation is improved. We consider this from a spacetype mixing angle, where classes of a given type do not generally match rooms with the same corresponding spacetype. We provide two different methods (exact and heuristic) that can alter the spacetype of a room and show that altering the spacetype profile leads to an improvement in the expected utilisation.

1.4 Contributions of the thesis

We summarize the contributions of the thesis as follows:

1. Introduce, formulate, and provide an initial model for the teaching space allocation problem, with and without splitting.

2. For the first time, investigate the effects of splitting on the teaching space allocation problem, and provide a local search based algorithm that preforms *Dynamic Splitting* of large teaching events, while achieving optimal space utilisation.

3. A preliminary investigation of the multiobjective nature of the teaching space allocation problem with splitting and study trade-off surfaces as a decision tool for assessing factors affecting space usage.

4. For the first time, using an Integer Programming model for the teaching space allocation, provide *experimental* evidence, of how the "Easy-Hard-Easy" pattern of computational difficulty at phase transitions can interact with space utilisation. This thesis pioneers a practical aspect of using the `Phase Transition` to improve ***space planning*** in general and utilisation in particular.

5. Using a student-based allocation model, show that splitting has the potential to reduce timetabling conflicts, and that studying the splitting problem without timetabling constraints still provides a reasonable approximation for assessing utilisation levels.

6. From a space planning perspective, provide, for the first time, a model that allows spacetype mixing, and show that altering the space profile can potentially improve space utilisation.

7. Devise methods to alter the spacetype profile of rooms with the main goal of improving the critical utilisation.

1.5 Presentations and publications

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2004). SpaceMap–Applying Meta-Heuristics to Real World Space Allocation Problems in Academic Institutions. *Extended Abstract, in Proceedings of the fifth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004)*, pages 440–445, Pittsburgh, USA.

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

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2007). Clustering Within Timetabling Conflict Graphs. *Extended Abstract, in Proceedings of the 3rd Multidisciplinary International Scheduling : Theory and Applications (MISTA 2007)*, Paris, France, August 2007.

Journal and Conference Publications:

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2006). Towards Improving the Utilisation of University Teaching Space. *Accepted for publication, Journal of the Operational Research Society* , September 2007.

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2006). The Teaching Space Allocation Problem with Splitting. In *Selected Papers*

from Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), Brno, Czech Republic., Lecture Notes in Computer Science, Volume 3867, August 2006.

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2007). Threshold Effects in the Teaching Space Allocation Problem With Splitting. *Accepted for publication, the European Journal of Operations Research. March (2008)*

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2007). Improving the Room Profile of University Teaching Space Submitted to *Annals of Operational Research. October (2007)*

In preparation for submission:

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2006). SpaceType Mixing effects on the Teaching Space Allocation Problem. *To be submitted to Operations Research.*

Beyrouthy, C., Burke, E. K., Landa-Silva, D., McCollum, B., McMullan, P., and Parkes, A. J. (2006). Partial Inheritance in the Teaching Space Allocation Problem. *To be submitted to PATAT 2008.*

Presentations:

Beyrouthy, C., Burke, E. K., (2004). New Model for the Space Allocation problem *Practice and theory of automated timetabling PATAT 04, Pittsburgh USA*

Beyrouthy, C., Burke, E. K., (2004). Adaptive Metaheuritics in Space Allocation *EUME*

Workshop on Hybrid metaheuristics, Nottingham, 2004

Beyrouthy, C., Burke, E. K., (2005). Student Based Model for Teaching Space Allocation problem *Space Allocation Workshop, Nottingham UK*

Beyrouthy, C., Burke, E. K., (2005). New Multiobjective Approach for the Teaching Space Allocation problem *OR47 Conference, Chester UK*

Beyrouthy, C., Burke, E. K., (2006). Teaching Space Allocation problem with Splitting *Practice and Theory of Automated Timetabling, PATAT 06 , Brno, Czech Republic*

Chapter 2

Splitting-Sectioning-Grouping : State-of-the-Art

The way we see the problem is the problem

Stephen Covey

2.1 Introduction

Proper management of academic teaching space requires a thorough understanding of the "mechanics" of university timetabling and the different factors that result in space wastage. This chapter surveys some previous work in university course timetabling with a special attention to sectioning and splitting.

The academic literature in course timetabling is vast and varied, and what complicates any survey is that, most of the research often use different problem formulations tailored to specific universities, resulting in a maze of different terminologies. For example, a course in a given institution, can mean a module in another, or maybe a lecture in a

	UK	AMERICAN	TSA	Others 1	Others 2
LEVEL X	course	program	**course**	curriculum	program
LEVEL Y	module	course	**module**	subject	course
LEVEL Z	Lec/Tut/Wksp	section	**class**	session	class periods
LEVEL S	group	lecture	**group**	N/A	N/A

TABLE 2.1: Course timetabling terminology in the UK, US and in some other universities. Column TSA contains the terminology used throughout the thesis

third one. We first provide a summary of the terminology[1][2] used across the literature in order to establish a common ground for the rest of thesis. Later we outline some of the main course timetabling models relevant to our work and discuss the differences between different sectioning methodologies. We conclude this chapter with a survey of the different methodologies used to tackle course timetabling.

2.2 Terminology

Different authors have described their sectioning and timetabling models using institution-specific ad-hoc terminology. In this section and in figures 2.1, 2.2, and table 2.1, We review some of the most used terms and relate them to the well known levels of study.

In the **American system** (table 2.1), mostly used in American and Canadian universities, `(LEVEL X)` defines a *program*: programs generally span multiple years and students enrol in such programs to obtain their qualifications. e.g. Business. Part of the requirements for a program , is a *course* `(LEVEL Y)` : e.g. Accounting or Finance. While *programs* are generally multi-year, *courses* are semester or single-year teachings. A *course* is taught in a series of lectures scheduled at different times in a weekly schedule, where a

[1]The TSA column of table 2.1, gives the terminology used in this paper.

[2]We do not intend to provide a typology in this approach, the main goal is establish a common ground that best describes the sectioning concepts.

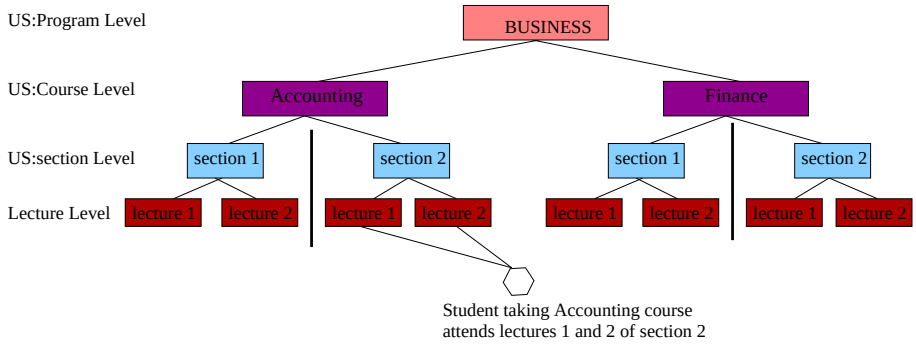

FIGURE 2.1: Terminology in `American` Universities

student is required to attend all of those lectures to complete the *course*. If the number of students enrolled in that course is larger then any available room capacity, then new *sections* `(LEVEL Z)` of this course are created (Note, *sections* can also be created for pedagogical reasons). Every *course section* is taught in its own series of *lectures* `(LEVEL S)`. Therefore, a student choosing course section 1 for example, will have to attend all the lectures of this course section. Papers following this model include (Hertz and Robert, 1998; Hertz, 1991; Carter and Laporte, 1998; Sampson *et al.*, 1995; Stallaert, 1997; Carter, 2000; Aubin and Ferland, 1989).

As shown in figure 2.1, a program requires more than one compulsory course (e.g. Accounting and Finance). Courses, in which, enrolment is high are offered in sections. Students have the choice of which section to join, and are enrolled in exactly one section of a course. Every section includes a number of lectures. A student is required to attend all of those lectures scheduled in different times.

The academic system in the **United Kingdom**, is explained in figure (2.2) and column **UK** of table 2.1. A given *course* `(LEVEL X)`, which is generally multi-year (referred to as program in the American system), includes many *modules* `(LEVEL Y)` of which some

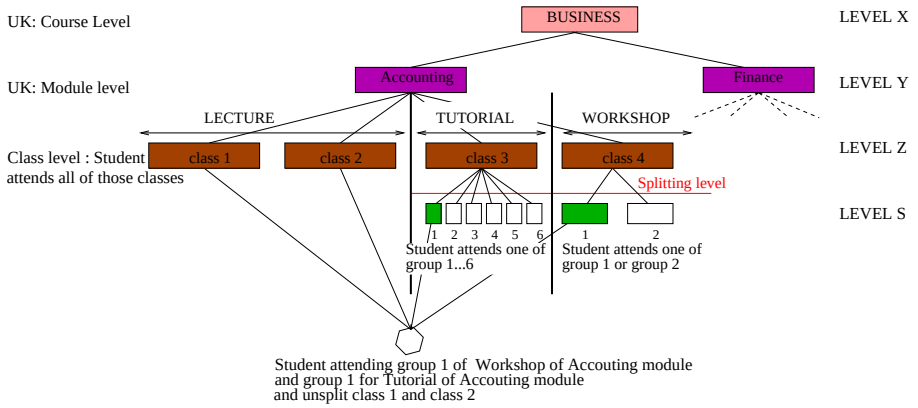

FIGURE 2.2: Terminology in UK Universities

are compulsory and other optional. In the example of figure 2.2, a module `(LEVEL Y)`, e.g. Accounting, has four classes: two of which are lectures, one is a tutorial type, and one a workshop type. A student enrolled in this module `(LEVEL Y)` should attend all of those classes `(LEVEL Z)` during the week, which consequently must be allocated at different times. The *classes* of type workshop, need to be split into 2 to 3 groups, `(LEVEL S)`, while tutorials need to be split in as many groups as required so that each group has 15-20 students (depending on the institution). Lectures on the other hand should not be split unless absolutely required, e.g. because all lecture rooms are fully booked, or this is no room with enough capacity for all students. Since every type of class splits differently, splitting would rather happen at the class level (as opposed to at the course level in the American system, `(LEVEL Y)`). For example, in the UK system, two students might belong to the same workshop group (e.g. group 1) but they can however be in 2 different tutorial groups.

The following are examples of papers that use the **UK** terminology: Mirrazavi *et al.* (2003); Abdelaziz Dammak (2006).

Note, in the American system, two lectures of the same section have to have the

same number of students, however in the UK system with classes like (Lec/Tut/Wksp) `(LEVEL Z)` could split in different group numbers, for example a workshop class can be split in 3-4 groups, while tutorial classes split in 8-9 tutorial groups .

Examples of other terminologies used in course timetabling can be found here: Smith (1971); Socha *et al.* (2002). Columns (`others 1`, `others 2`) of table (2.1), include other terminologies used in course timetabling models for some European universities. When no sectioning or grouping is involved, "sessions" or "class periods" are the lowest level to be used.

Many of the papers, dealing with graph colouring, where no sectioning is required use the "course-lecture" terminology. "Lectures" is also a very common term, if no splitting is needed, usually the goal is to allocate a lecture to a room and time-slot. Lectures will therefore represent a `teaching event` that requires allocation. In **chap.** 3, as no splitting is performed, we will use the term `event` to represent allocation of classes that do not split.

2.3 Course Timetabling

2.3.1 Main course timetabling models

NOTE: The student sectioning problem is often called the "grouping sub-problem " (See Schaerf, 1999).

Course and University Timetabling has attracted many researchers, over the years. Various approaches have been studied, compared and tailored to specific academic needs and preferences. Course Timetabling (CT) problems[3] come under different distinct flavours, depend-

[3] *Extensive resources and conference proceedings, on timetabling in general, can be found in Burke and Ross (1996); Burke and Carter (1998); Burke and Erben (2001); Burke and Causmaecker (2003); Burke and Trick (2005); Burke and Rudová (2006)*

ing on specific institutions or departmental requirements. The standard course timetabling formulation can be found in Schaerf (1999); Rossi-Doria *et al.* (2003, 2002). In that formulation, courses can generally fit in available rooms and there is no need to split or consider any sectioning[4]. In the following survey, we will look at models where authors consider some sort of sectioning, whether directly of indirectly. Most of the models try capture the real-world aspects of course timetabling, and faced with sectioning requirements, have to adapt their model to allow some sort of grouping. Timetabling courses - offered in multiple sections - is problematic. Carter identifies the "timetabling paradox":

> *We cannot assign times to sections until we know which students are in each section. But we cannot assign students to sections until we know when the sections are timetabled*

Most of the approaches to tackle this "paradox" are centred around working in multiple stages and decomposing the problem into smaller ones, or by performing an initial grouping of requests to courses before the timetabling stage. Then, after the timetable is available, students are reassigned to sections, minimising clashes. Note, clashes happen when a student is assigned to 2 teaching events which are offered in the same timeslot.

There are various ways to classify approaches to sectioning in course timetabling. The variety of different models and motivations behind most of the research makes this task challenging. Throughout this chapter we survey the research according to common model characteristics and approaches relevant to splitting/sectioning. At the highest level, two large sectioning approaches can be suggested and summarised as follows:

- *Course-centred splitting*: These are approaches where the timetabling occurs at the

[4]Unless pedagogical constraints require having multiple sections.

course level, not considering any decision on students allocation. The interest being generally on how to assign classes/groups to available rooms, times and/or teachers.

- *Student-centred sectioning*: Also known as large scale timetabling. It generally includes a course-based timetabling scheme in addition to the allocation of individual students to sections of courses. The main goal would be to reduce conflicts resulting from student enrolment, while achieving the required objectives.

Note, such classification is not strict; some special problems or formulations cannot be easily placed within the above categories. Therefore, later in the chapter we group some of the papers, in additional methodology classes.

2.4 Course-centred splitting

In course-centred splitting, conflicts or student clashes are generally part of the input, and no direct decision on the assignment of individual students to particular sections is required. We survey some of the most important and relevant papers.

Back in the 70's, Smith (1971) constructed a statistical model that measures the probability of assigning courses to rooms, assuming known demand distribution. He considered student demands for courses as random variables, and the measure used to accommodate student demand is the probability of "overflow" which represents the probability that at least one course cannot be allocated. Dividing student into sections obeyed the following rules: 1) section size is less then a maximum allowable size determined by the administrators, 2) number of sections should be minimised, 3) sections should be balanced in size. Smith's approach does not study any relationship between the section sizes and

numbers with other objectives of interest like location, utilisation etc. The problem is not an optimisation one, and does not cover any space planning issues.

Selim (1988) proposed a *split vertices*, graph theoretical approach to model the effect of splitting on the chromatic number in a timetabling conflict graph. His main aim was to determine the number of sections and the number of periods needed to allocate all courses into the available timeslots. He provides upper bounds on the minimum number of sections needed to have a feasible solution and restricts the number of sections for some type of courses `(LEVEL Y)`(Lab courses). His approach was merely centred around resolving conflicts and providing a feasible timetable. It is a decision problem, a modified study of a graph colouring problem.

Mirrazavi *et al.* (2003) used goal programming and a genetic algorithm, in a two-stage approach to solve another variant of university timetabling problems. While goal programming was mainly used to solve the room allocation problem, a genetic algorithm assigned timeslots to lectures[5]. The main "goals", in their model, are to maximise utilisation and satisfy teaching constraints. Lectures with sizes larger than available room capacities needed splitting. Hence, they used a pre-processing stage to divide big lectures into smaller groups using different (groups, room) combinations. In a later stage, they allow the goal programming engine to choose the best of these combinations that would improve all goals. Goals considered were: full lecture allocation, eliminate lecture conflicts, minimise physical time distance between lectures, and maximise timeslots preferences ("lunch breaks Goal"). Their approach is rather similar to Tripathy (1992) except that the latter uses a Lagrangian relaxation approach at the student level (more on Tripathy's approach in **sec.** 2.5).

[5]lecture is at `(LEVEL Z)`

Boronico (2000): suggested a multistage[6] decision support system, with the following goals : 1) Decide on courses to offer, 2) decide on number of sections per course, and 3) Assign timeslots and faculty to sections. Deciding on the number of sections is done by a separate mathematical sub-model, which defines a *level of coverage*[7] to courses, such that the expected demand for courses is met with a high probability.

Ram *et al.* (1987) used an algorithmic procedure to tackle the splitting aspect of academic course planning. Course splits depended on projected course demands and the size of a course.

2.4.1 Course-centred mathematical models

We now look at some course-centred splitting models that used advance mathematical programming techniques to handle the creation of sections and their assignment to timeslots and rooms. Al-Yakoob and Sherali (2006, 2007) addressed the class timetabling problem using elaborate mathematical programming models. They suggested large mixed integer, 2-stage formulations. The first stage performs the allocation of class sections to timeslots, and the second-stage assigns faculty to given sections. The number of sections per class `(LEVEL Z)` is fixed. They also consider many different "non-classical" constraints, e.g. all male/female sections, parking and traffic congestion, classes in consecutive timeslots, sections of classes spread over different timeslots, flexible student choices, etc. They compare their approach with a manual assignment and perform a sensitivity analysis by understanding changes in the standard deviation of different soft constraints.

[6]Using a Hierarchical mathematical model
[7]Probability that the expected demand will be met

Daskalaki *et al.* (2004, 2005) proposed an elaborate Integer programming model to tackle a variant of university timetabling problem. The approach includes assigning courses to teachers, timeslots and rooms while deciding on the number of periods for every session `(LEVEL Z)` of a course `(LEVEL Y)`, as well the number of sessions. The number of sessions is constrained by the availability of time periods. In Daskalaki *et al.* (2005) they suggest a 2 stage approach, to overcome the difficulty of solving large problem instances, whereby their first stage solved the LP relaxation and the second stage solved the integer program for each day of the week.

Papoutsis *et al.* (2003) used Column Generation (CG) to assign teachers to classes and timeslots in a Greek high school. Similarly, Qualizza (2005) used column generation in a branch and price algorithm and defined a master problem and a sub-model. Both authors defined (CG) patterns as being the assignment of weekly timetable of a single course. As it is known in (CG), the linear relaxation of a master model provides the prices for a sub-model. Solving the sub-model would provide new columns (patterns) for the master model. The process is iterative.

Avella and Vasil'ev (2005) proposed a formulation for the university course timetabling problem UCTP of Gaspero and Schaerf (2002) as an integer linear program. They studied the relationship between UCTP and set packing to introduce valid inequalities (cutting planes). The problem does not involve splitting of courses, and there is no decision support direction on that approach.

In the context of grouping, Dimopoulou and Miliotis (2001, 2004), proposed an integer programming approach to assign courses and groups of courses to timeslots and

classrooms. A group, as they define it, is a set of courses that cannot be assigned to the same timeslot, since they are taken by the same students.

Breslaw (1976) used integer programming to tackle the faculty assignment problem, assigning sections of courses to faculty. The number of sections is dictated by 1) the availability of teachers, 2) teachings (the total number of sections per teacher), and 3) available timeslots so that no time conflicts occur.

Tillett (1975) extended Breslaw's approach by including teacher preferences (preference ratings), as a main objective, with the main constraint that all sections of courses need to be fully allocated.

2.4.2 Course-centred goal programming models

This section reviews approaches focusing on the multiobjective and/or multicriteria nature of timetabling and splitting problems, when some objectives are favoured over the other depending on institutional preferences. Harwood and Lawless (1975) used Goal Programming (**Sec.** 2.8.1) to examine the conflicting goals in the faculty-course assignment problem. Their model assigned faculty to courses with the main goal of satisfying personal "faculty preferences" goals. Other goals included satisfying organisation teaching load requirements, and teaching staff personal preferences by: 1) selecting specific courses, 2) minimising the number of teaching preparations, 3) minimising the number of teaching days in a week, 4) minimising the number of night classes, and 5) maximising the number of sections for the same course (for variety). The major drawback to their model model is that it may be very difficult to implement. The small instance used the assignment of only seven faculty members resulting in a model with over one thousand constraints and almost 200

variables. Department chairpersons do not always have the time or the expertise to develop such a model for each particular instance. In Harwood and Lawless approach the number of sections per course is fixed and goals are more directed towards faculty preferences. Schniederjans and Kim (1987) extended and dealt with limitations of Harwood and Lawless (1975) approach, mainly in terms of size of the model. They considered that some of the goals are supposed to be hard constraints, and added extra, university specific goals, like departmental goal and teaching load requirements. A drawback is that, no time assignments to courses was considered. Badri (1996) suggested a new Goal Programming (**sec.** 2.8.1) model where they extended the work of Schniederjans and Kim by including most of the goals of Harwood and Lawless and also performing, the course time assignment.

2.4.3 Other methodologies

Other different methodologies for splitting can be found in the following models. Dyer (1976) used a flow formulation to model the course-teacher-timeslot problem. They presented an optimisation system where the number of sections per course is not fixed but bounded. There is no consideration for the section size, and the limiting effects on the section number is mostly the availability of teaching hours.

Glassey and Mizrach (1986) suggested a decision support system with an optimisation module for classroom assignment. They've used a heuristic algorithm to assign classes/sections to rooms and timeslots. The number of sections is fixed in their model.

Bloomfield and McSharry (1979) approach was among the first models to assign sections to timeslots, rooms and teachers, in a real world problem. They extended the work of Mulvey (1983) and realised the need to use heuristics in multistage approaches to tackle

the problem. Their model decides on the number of sections (LEVEL Z) required for every course (LEVEL Y), based on capacity requirements.

Hertz (1992) used Tabu Search to tackle a variant of class-teacher timetabling. He considers in addition to the set of classical course timetabling constraints, the teachers availability, precedence, and compactness of a timetable. The terminology he used, very similar to that of Column **Others 1** in table 2.1, includes curriculum, classes, topics, etc. There are several classes of students, and every class follows a specific curriculum. Every curriculum consists of a set of subjects and each subject consists of a set of topics. A teacher is assigned to a topic and a class. No splitting is allowed, and it's an optimisation problem, tackled in a single stage. One characteristic of his approach is that periods can have different length.

2.5 Student-centred sectioning

In this section we look at the problem of *large scale timetabling.* which involves constructing a timetable and assigning individual students to available sections (LEVEL S). Different approaches were suggested in the literature. Some, for example, construct an initial timetable in a first stage, then allocate students to corresponding sections so that conflicts are reduced, student preferences met and/or size of sections is balanced. The sectioning part (Grouping Sub-problem[8]) further complicates the timetabling process since extra decisions on student assignment need to be taken.

[8]The sectioning problem also known as the (grouping sub-problem), should not be confused with the concept of grouping which tends to group students or courses so as to minimise timetabling clashes

2.5.1 Multi-staged models

Approaches to sectioning using multi-stages, attracted many researchers, because these methods allow more flexibility when solving large scale problems. Carter (2000) suggested decomposing the problem and solving it in multiple stages. He introduced the concept of "homogenous sectioning" where in a pre-assignment step he groups students with similar course requests and then assigns those groups to sections. This allows an estimation of the required number of sections. After the timetable is constructed, a later stage assigns individual students to sections so that conflicts are minimised.

In his earlier work, Tripathy (1984) used a Lagrangian relaxation algorithm to allocate subjects `(LEVEL Y)` to periods and rooms: the problem is to assign students enrolled in streams[9] `(LEVEL X)` which are represented by 900 subjects, to different periods. Students from 4 to 6 streams have to study some particular subjects simultaneously (an extra, complicating constraint). Therefore, those subjects cannot be assigned to the same time-slot. His main goal was to allocate all subjects and have a feasible conflict free assignment where no students in two different subjects are assigned to the same time-slot. Tripathy opted for a grouping approach to reduce the problem size and introduced: `Student-groups and subject-groups`.

Grouping is part of the pre-processing stage: first he groups students of a stream in a `student-group` and then splits the given `student-group` in two or more splits based on the enrolment of students to subjects (a split would include students enrolled in optional and mandatory subjects), in that way if the initial non-split group requires k time-

[9] specialisations

slots then the split groups would require less than k. Moreover, Tripathy introduced `subject-groups` which represent a group of one or more subjects being taken by the the same `student-group`. With this grouping the author is able to reduce the number of variables representing subjects to be assigned to timeslots. Rooms are also grouped in `room-groups` sorted by capacity such that `subject-groups` are assigned to `room-groups` with similar capacities. The main purpose of using this strategy is to reduce the size of the problem, so that it can be tackled with a Lagrangian relaxation algorithm. That approach differs from the work in this thesis because here, splitting is dynamic and is dictated by the interaction between utilisation and all the other constraints, rather then just the student conflicts.

In a later work, Tripathy (1992) proposes a decision support system to aid administrator to perform student sectioning, subject to the following constraints : 1) generation of conflicts matrix, 2) multiple section grouping and 3) generation of a class schedule. In this approach the number of sections `(LEVEL S)` is bounded to three sections per course. The conflict matrix is reconstructed after deciding on the number of sections. In multiple section grouping he tries to assign sections to courses so as to minimise conflicts and number of time-slots used and if it's not possible to satisfy a complete conflict reduction he then restricts student choice (enrolments) in subjects.

Aubin and Ferland (1989) tackled the large scale timetabling problem by solving the two sub-problems *timetabling and grouping*. They formulate each sub-problem as an assignment problem. Then, they iteratively modify the timetable and the grouping while reducing timetable course conflicts, to provide better load balancing in sections. Their

algorithm terminates when no further improvement in the objective function is obtained. They identify two types of conflicts: 1) conflicts due to lectures and 2) conflicts due to students, and assign penalties for violating each of them. While the section number is predetermined and fixed for each course, the section size is not. The solution method in Aubin and Ferland's model is a heuristic procedure where in every iteration, courses/students are reassigned to different periods of the week. Many authors e.g. Hertz, later contested Aubin and Ferland's work by claiming that their model delivered local but not a global optimum. Laporte and Desroches (1986), used a multiple pass algorithm to assign students to sections in a fixed and available course timetable . They made sure all student conflicts were resolved, student preferences/selections satisfied and section sizes well balanced. Ferland and Fleurent (1994) presented a decision support system (SAPHIR) that is currently used in two Canadian universities: Montreal and Sherbrooke. SAPHIR includes several heuristic improvement procedures that yield improved solutions for a given schedule, as well as an automatic optimisation procedure to generate a timetable from scratch. The system solves the grouping as well as the timetabling problem. It is also highly interactive and allows the user to change many of the input parameters obtaining a different timetable.

Hertz and Robert (1998) used a tabu search `Meta-Heuristic` to tackle the large scale timetabling problem in an interactive manner, modifying the grouping so as to reduce conflicts between courses and maintain load balancing between sections. Hertz uses a fixed timetable and maintained the number of students in every section `(LEVEL S)` as close as possible to a given target. Heuristics, in his model can alter the section size.

Winters (1971), was among the first papers in sectioning. He suggests a multiple

pass algorithm. In the first pass they fix the number of sections for every course and the number of students assigned to a section which should be less then a given target. On later passes, they identify 3 cases: 1) if not enough sections for a given course they add one section, 2) if there are too many sections for a course, containing a small number of students, then drop one section, 3) if not enough seats for section of a course, then add an amount of seats as dictated by the algorithm, and 4) if no sufficient demand for a given section to be opened, then a decision is made (heuristically) to keep the section or not. It's an elegant multiphase-multipass algorithm.

In fairly recent work, Alvarez-Valdes *et al.* (2000) suggested a two phase algorithm using `Maximum Cardinality Independent Sets` and `Tabu Search` for sectioning, focusing on satisfying student preferences and maintaining load balancing. They search for feasible solutions and model the problem as a decision problem. The number of sections is their model is fixed, part of the input, and no change allowed to the timetable post assignment. Also Sampson *et al.* (1995) suggested a heuristic approach to course timetabling where they first timetable courses and then assign the students. They construct a timetable after students enrol for classes. Their approach included objectives to satisfy teacher preferences and student satisfaction. The procedure used allows interactive feedback from the scheduler.

2.5.2 Single-staged approaches

This section looks at some more challenging models where sectioning is performed in a single stage. Boland *et al.* (2006) used an integer linear program to course timetabling, showing that it is possible to find exact solutions to the problem of grouping and room assignments in a single stage. With a fixed number of sessions per class, they perform teacher and

student allocation but don't assign timeslots to courses. They grouped classes in blocks, such that all classes of a block can be timetabled in parallel.

Head and Shaban (2007) used a heuristic approach to create student and course timetables. they aimed to minimize the total section number and load balance for all sections. It's an an optimisation approach, where they use a large set of real-world constraints. The system is aimed at improving student satisfaction and reducing timetabling violations.

Busam (1967) tackled the problem of assigning students to sections in a fixed timetable environment. AminToosi *et al.* (2004b,a) used clustering with a fuzzy function to tackle the sectioning problem stressing on respecting student course selections and sections load balancing. Wood and Whitaker (1998) used non-linear goal programming on a secondary school timetabling problem, where they assign students to timeslots and rooms while assigning as well, teachers to classes. Other student-oriented approaches, include Cheng *et al.* (2003); Miyaji *et al.* (1988); Banks *et al.* (1998); Laporte and Desroches (1984); Mingers and O'Brien (1995).

Rudová and Matyska (2000) suggested constraint programming with variable annotations and a special objective function to help solve over-constrained parts of the sectioning and room assignment problem.

Also Müller *et al.* (2005) introduced a new iterative forward search algorithm to solve the minimal perturbation problem in course timetabling. They applied their algorithm on real-world instances at Purdue University. Rudová and Murray (2003) is another advanced constraint satisfaction approach also applied to the `Purdue` data-sets.

2.5.3 Starting Point of the study

In this section, before surveying the litterature further, we look at the most relevant approaches that initially formed the cornerstone of this study. Generally, the literature on course and university timetabling is focused on courses with a **specific/fixed number of sections**. Every course has a pre-assigned number of sections and the main aim is to allocate courses/sections to timeslots, teachers or rooms. Similarly, one looks at the assignment of students to specific sections, e.g. Al-Yakoob and Sherali (2006, 2007); Harwood and Lawless (1975).

However, recently, some papers tackled, indirectly, the idea of **dividing large courses** into smaller ones creating sections of a given size, or deciding on number of sections needed. Those were instrumental in inspiring our approach, with for example:

- Smith (1971) through an advanced, generalized statistical model.
- Selim (1988) with **split vertices** in a graph theoretical approach.
- Mirrazavi *et al.* (2003) through a hybrid methodology that includes, *Goal programming* with a *Genetic algorithm*.

On the student based sectioning approach, Tripathy (1984) summed up an `administrator-oriented` decision support system, and Carter (2000) with a decomposition based approach, have directed our view towards, a focused study of the students conflicts. Consequently, the impact of those conflicts on space and utilisation, have helped inspire our design of an enrolment generator.

Note, besides the aforementioned references, our research has been driven by two

essential and primary goals of (1) developing an understanding of the factors that lead to low utilisations, and (2) to investigate methods to chose risk-free, safety margins that are more cost-efficient for real-world teaching space, as underlined in the space reports, McCollum and Roche (2004); McCollum and McMullan (2004).

2.6 Decision support systems

The nature of course timetabling problems, constraints and objectives, is such that, a single solution to a specific problem might not be of interest. Instead, a group of solutions, preferably with tunable parameters, gives university administrators more flexibility in making future decisions.

Therefore, various papers have suggested decision support systems `DSS`, to tackle timetabling in academic universities. Generally motivated by the presence of conflicting objectives, satisfying an objective at the expense of another becomes a matter of institutional choice.

Most famously, Dinkel *et al.* (1989) suggested a decision support system to assign teachers and subjects to timeslots and rooms, modelled as an efficient network problem. Their approach is quite similar to our approach, including soft competing constraints like timetable, space usage and faculty preferences. They study the trade-off between faculty/subject assignment and Utilisation[10], and provide a weighted preference for sections of a given department to be allocated before other sections. The model includes the choice to select subjects to be allocated disregarding *No Partial Allocation*[11]. Dinkel *et al.* approach

[10]Defined as unassigned and assigned courses to rooms

[11]When assigning teaching events, No Partial Allocation penalty would force the solver to allocate all events, meaning that allocation is fully performed, not partially

differs from ours, mainly in the fact that no splitting of subjects was involved, and also they considered course splitting as already provided by administrators. So the number of sections and the `Section-Size` are fixed in their approach.

Ferland and Fleurent (1994) also suggested heuristics and exact methods within a decision support system for both the timetabling and grouping sub-problems. For other decision support systems, (see Boronico, 2000; Harwood and Lawless, 1975; Schniederjans and Kim, 1987; Badri, 1996; Dimopoulou and Miliotis, 2001, 2004; Glassey and Mizrach, 1986; Ferland and Roy, 1985).

2.7 Classroom/teacher assignment

The classroom assignment problem focuses on the assignment of rooms to teaching events considering a fixed timetable. Carter and Tovey (1992) thoroughly studied the different formulations and analysed their complexities. They proved which cases of the variants are tractable and which are NP complete[12]. They classified the problems as interval and non-interval according to whether classes meet once or many times, in a week. Ferland and Roy (1985) solved the classroom assignment sub-problem by reducing it to a quadratic assignment problem while Abdennadher (1998) used constraint logic programming to tackle the same problem.

Fizzano and swanson (2000), used a matching algorithm to tackle a variant of the classroom assignment problem and provided optimal schedules to plan classes in classrooms. There was no splitting involved, but an interesting aspect in their study was assessing the utilisation levels of rooms in *Puget Sound* university. For other relevant problems, like the

[12]Details on complexity and intractability can be found in (Garey and Johnson, 1979; Karp, 1986).

"Teacher Assignment Problem", (see Santos *et al.*, 2005; Tillett, 1975).

2.8 Mixed Integer and integer programming

Integer Programming have been abundantly applied to sectioning and timetabling problems. An integer program IP seeks the maximisation or minimisation of a linear function subject to linear constraints and decision variables taking integer values only. (see Bosch and Trick, 2005; Boyd and Vandenberghe, 2004; Dantzig and Thapa, 1997; Nemhauser and Wolsey, 1988).

An IP with n variables and m constraints, has the following form

$$\min \quad z^* = C^T x$$

$$s.t. \quad Ax \leq B \quad \forall x \in \mathbf{Z}$$

with A an $m \times n$ matrix, $A \in \mathbf{R}^{m \times n}$, and vectors $B \in \mathbf{R}^m$, $C \in \mathbf{R}^n$

With the following cases (Achterberg, 2004; Hooker, 2005, 2002; Nemhauser and Wolsey, 1988):

- if $x \in \{0,1\}^n$ the problems becomes a *Binary Integer Program* (BIP).
- if **all** variables are real, $\forall x \in \mathbf{R}^n$, it will become a *Linear Programming problem* LP. Linear programming can be solved efficiently using the simplex method. Nemhauser and Wolsey (1988) method.
- If some of the x variables, **NOT all**, are real the problem falls under the *Mixed Integer Programming* MIP area .

IP provides a flexible tool to model intricate timetabling problems, but is faced with the difficulty of solving large instances. For state-of-the-art approaches to sectioning using IP (see Papoutsis *et al.*, 2003; Dimopoulou and Miliotis, 2001; Mirrazavi *et al.*, 2003; Daskalaki *et al.*, 2005; Avella and Vasil'ev, 2005; Mulvey, 1983; Aubin and Ferland, 1989; Cheng *et al.*, 2003; Miyaji *et al.*, 1988; Mirhassani, 2006). For example, Daskalaki *et al.* (2004) recently proposed a novel *Binary Integer program* on a course timtebling formulation at the *University of Patras*, but was able to solve only a small instance taken from a single department (72 courses and 211 teaching periods and 12 classrooms).

Techniques and methods that combine Mixed-Integer and Constraint Programming CP can be found in (Rodošek *et al.*, 1999; Milano *et al.*, 2002; Milano and Wallace, 2006; van Hoeve, 2001, 2005; Hooker, 2005, 2002).

2.8.1 Goal programming

Goal Programming GP is a multi-objective mathematical programming technique. In many complex problems, decision making involves maximising more than one well defined objective function. In fact there is a growing interest in considering more then one specific goal (target) to be achieved when tackling timetabling problems

A GP problem can formulated as follows (see McNamara, 1971; Tamiz *et al.*, 1998) :

$$z = \sum_{i=1}^{k} (u_i n_i + v_i p_i) \tag{2.1}$$

$$\text{s.t. } f_i(X) + n_i - p_i = b_i \quad i = 1 \dots k$$

$f_i(X)$: a linear function of X with X as the vector of nonnegative decision variables and

unknowns.

b_i the target value for that objective.

u_i and v_i, the weights corresponding to objective function deviation. n_i and p_i represent the negative and positive deviations from this target value.

k the number of objectives. A survey on GP can be found in Tamiz *et al.* (1998); McNamara (1971).

Goal programming has been successfully adapted to timetabling and sectioning problems. The ease of use and the flexibility in weighting and prioritising constraints has contributed to making GP an ideal tool for decision support systems. Some approaches are detailed in **sec.** 2.4 and 2.5, and can be found also in (Harwood and Lawless, 1975; Schniederjans and Kim, 1987; Badri, 1996; Diminnie and Kwak, 1986; Ritzman *et al.*, 1979; Lee and Clayton, 1972; Miyaji *et al.*, 1988).

2.8.2 Phase transition

Phase transition is a common behaviour, in physical and combinatorial systems, where a system is transformed from one phase to the other. In physics, for example a transformation from solid to liquid at a given critical point, the "melting point", is an example of that transition. The critical point itself depends on a special combination of temperature, pressure and material composition, and it's around that point, that the system undertakes the transformation. Phase transitions themselves exhibit, in general, a sudden change in one of the properties of the system, and it is an interesting problem to understand and quantise the probability and extent of such a transition.

Phase transition in complexity and combinatorial optimisation has recently at-

tracted many researchers. Transitions around a critical point has been identified in many NP-Hard problems like SAT, Hamiltonian Cycles (see Cheeseman *et al.*, 1991; Coarfa *et al.*, 2000; Parkes, 1997; Smith and Dyer, 1996, and Others).

Phase transition has also been studied on number partitioning (NP) and Multi-Way Number Partitioning (MNP) (see Gent and Walsh, 1995, 1996). Parkes (1997) studied the phase transition in satisfiability problems.

In timetabling and graph colouring, Ross *et al.* (1996) showed that there is a phase transition for some evolutionary algorithms tailored to timetabling problems. They showed that, increasing the number of constraints made the problem harder to solve until a point where it became easier. The approach considers the behaviour of the algorithm with the increase in constraints but does not tackle the case with an increase in events or rooms and is not set as a decision or planning problem. Erben (2001) has identified phase transition regions in a class of graph colouring problems with randomly generated graphs. They extend the colouring problem to the grouping problem using a genetic algorithm. For other works on thresholds and phase transition (see Coarfa *et al.*, 2000; Diminnie and Kwak, 1986; Smith and Dyer, 1996; Gent *et al.*, 1996).

2.9 Neural networks in timetabling

Artificial neural networks have also been widely used in solving combinatorial optimisation problems. They have proved to be successful in tackling complex problems, including the full range of Timetabling and sectioning problems. (see Carrasco and Pato, 2004; Kovacic, 1993; Tsuchiya and Takefuji, 1997; Corr *et al.*, 2006) for application in timetabling problems.

2.10 Constraint programming in timetabling

Constraint Programming is a programming paradigm where variables are related in a way that can be stated in the form of constraints.

The basic motivation in *Constraint Programming* is that some user defines the constraints that models a specific real-world problem, and a general purpose solver is used to solve those constraints. For example, in trying to schedule a set of activities, decision variables might be the available timeslots, or even resources needed to accomodate them, and the constraints might act on the availability of those timeslots, the conflits between activities in some given time etc. Hence, constraint solvers use such problems, consider their variables and constraints, and find given assignements for those variables that satisfy all constraints. Different ways to search the solution space (through specific algorithms, like backtracking and branch and bound) can be used, but face the challenge represented by the computational complexity of constraint problems. Much care, is therefore needed in finding good formulations that can considerably reduce the search space.

Constraint Satisfaction Problems `CSP`[13], represent the process of solving a set constraints such that:

Define the triple $\langle X, D, C \rangle$:

X a finite set of variables,

D is a domain of values,

C is a set of constraints.

$\forall X_i \in \{X_1, \ldots, X_n\}$,

[13] see Frish (2002), Wallace (2004)

D_i domain of variables X_i.

Given a finite set of constraints, where each constraints restricts the values that the variables can take e.g. $X_1 = X_2$, $X_1 + X_2 < X_3$

A **Total Assignment** maps variables X_i to one element of their domain D_i. A **solution** to an instance of CSP will be a **Total Assignment** satisfying all constraints. For an in-depth study (see Milano *et al.*, 2002; Milano and Wallace, 2006; Hooker, 2002).

Applications of `CSP` to timetabling and planning can be found in Stamatopoulos *et al.* (1998); Milano and Wallace (2006), Abdennadher and Marte, (2000).

Constraint Logic Programming[14] `CLP`, on the other hand, combines logic and constraint programming, by extending logic programming to include constraints in the body of clauses. *Assignment-type* problems, with large number of complex constraints can be easily modelled as `CLPs`. Various powerful solvers are widely available for solving `CSPs`: CHIP, ILOG Solver and ECLiPSe and Prolog were developed for solving and modelling large scale `CSPs and CLPs` and are widely used by the scheduling and timetabling community (Cheng *et al.*, 1995; van Hoeve, 2000; Hooker, 2005, 2002; Stamatopoulos *et al.*, 1998; Kambi and Gilbert, 1996).

2.11 Other Approaches

Previous work relevant to space utilisation, and studies of demand for space in universities is abundant in the litterature. Fizzano and swanson (2000) reported the needs by their institutions (University of Puget Sound) to improve their space usage, and tried to assess the classroom requirements to meet all the demand. Also, early work, on space planning

[14]For a survey of Constraint logic programming see Jaffar and Maher (1994)

by Bullock (1974), studied the demand for teaching space from an architectural point view, stressing on the planning aspect for teaching space. They have considered conflicts between events in a discrete fashion and listed all possible timetabling and teaching patterns. Based on that, they tried to estimate the demand for rooms. Demand beeing measured by the probability distribution of events in weekly schedule.

2.12 Summary and conclusion

This chapter has surveyed research in splitting and sectioning and discussed it's relevance to our work. We sought to clarify some of the main terminologies used in the course timetabling literature. We looked at different forms of splitting performed in course-centred timetabling, where the teaching events are generally split disregarding individual student assignments, and compared different methodologies under this approach.

We also surveyed some of the literature in large scale timetabling, where focus is on assigning students to available sections/groups, and discussed the different approaches proposed.

Finally, we exposed some of the main methodologies used in the field, and provided references to relevant work. The next chapter will formally introduce the foundation of the work undertaken in the thesis.

Chapter 3

Initial Investigation of Short-term Space Management

'Rooms are used half of the time, and when used are half full'

HEFCE

3.1 Introduction

This chapter introduces the lecture-based - *splitting free* - pure teaching space allocation. Motivated by the need to improve space usage, we carry out an initial investigation of the real world factors and constraints that affect space usage. We try to develop an understanding of how good is space utilised through one aspect of our problem, the **lecture** case: "Lectures" ideally should not be split. From an optimisation perspective, we use Pareto fronts in the presence of different constraints and provide quantitative evidence that constraints arising from timetabling and location have the potential to explain low utilisation

seen in reality. A multiobjective problem arises, when constraints related to timetabling and location are conflicting with the main goal of optimising space usage. This chapter will introduce some of the methodologies seen in coming chapters, as well as the exact and heuristic solvers used to tackle the problem.

3.2 Aims and motivation

As discussed in **chap.** 1, Utilisation figures in academic institutions are very low. Attempts to remedy this situation, and so to improve space planning are hampered by the lack of a qualitative understanding of why utilisation is generally that low.

First consider a simple "pure" event[1] allocation problem in which we optimise utilisation by assigning teaching events (for example lecture classes as explained in chapter 2) from a given pool to available timeslots. On the data-sets we have available, this immediately gave utilisation of 85-90%. This is far too high to match reality, and so indicates that a model based purely on space issues, and given free choice of events, is inadequate to model teaching space allocation in real-world universities.

Based on administrators recommendations, to extend the model we introduce some realistic constraints, that map the event allocation problem to a real world problem. Therefore, since event allocation (e.g. lectures) usually takes place within the context of many constraints on locations and timings of events, accordingly we include within our model, objectives that are intended to provide an approximation/abstraction of real timetabling issues.

[1]Events, in this chapter represent "lectures" as explained in chapter 2. "lectures" needs to be allocated and do not require splitting. We call them "events" to simplify and generalize the approach which could apply to other type of teaching events.

The underlying problem that we consider is the event allocation problem. This problem generally arises as the classroom assignment problem (Carter and Tovey, 1992; Schaerf, 1999; Burke and Petrovic, 2002) within timetabling problems, and consists of selecting feasible room assignments for events. However, in our work it is not a hard constraint that all the events must be allocated, but rather the decision as to which events are considered, becomes a part of the problem. To the best of our knowledge, this differs from all the course timetabling literature in which it is a hard constraint that every event must be allocated. This also meant that we had to implement a new solver rather than attempt to use an existing one.

3.3 Lecture based, teaching space allocation model

In the following subsections, we give a description and a mathematical formulation for the teaching space allocation problem (TSA). Next we define the terminology used in this chapter.

3.3.1 Description

For each teaching *room*, we are given:

1. **Capacity** : the maximum number of students that the room can accommodate.

2. **Timeslots** : the number of timeslots for which the room is available during the week (or other relevant scheduling time period).

3. **Department** : the department that owns, or is most closely associated with, the room.

An *event* requires the following information:

1. **Students** : the number of students that must be accommodated.

2. **Department** : the owning/associated department. The primary task is to assign *events* to rooms so as to satisfy the following *hard constraints*:

1. The size of an event (students) must not exceed the room capacity

2. The number of events allocated to a room must not exceed the number of timeslots, as events cannot share room timeslots.

Notice that there is no constraint saying that an event must be allocated to some room (like is often the case in timetabling). Instead, finding a set of events that are to be allocated such that utilisation is maximised, is part of the problem.

3.3.2 Real-world Location and Timetabling penalty

In this section, we describe the objective and some real world constraints that mostly influence administrators in seeking better timetables.

3.3.3 Utilisation

In **chap.** 1 we have detailed the main motivation behind studying utilisation levels in academic institutions. Our objective is to increase room *Occupancy* and *Frequency of usage* by filling rooms with as many students as possible to directly improve utilisation. The utilisation measure in this thesis, is closely linked to the number of Bodies-On-Seats (BOS)[2] which is the sum of all students allocated to a given room in a weekly schedule. If a room

[2]A commonly used acronym within the Academic Planning community.

is used many times in a given schedule, then U is the sum of all students that have been allocated to this room in that schedule over the total available seats.

$$U := \frac{\text{BOS used}}{\text{total BOS capacity } (BOS_C)} \tag{3.1}$$

Usually we refer to U as a percentage. Then $U = 100\%$ iff every seat is filled at every available timeslot. One can also think of U as the product of the frequency of usage by the occupancy of a room. Our reasoning goes as follows:

The frequency of usage represents the number of timeslots used in a given room j over the total timeslots available p:

$$F_j = \frac{\text{timeslots used}}{\text{timeslots available}} \tag{3.2}$$

and the Occupancy (O_j) is the fractional usage of room j at any time, that is the number of seats used over total available seats:

$$O_j = \frac{\text{seats used}}{\text{seats available}} \tag{3.3}$$

The occupancy is generally averaged over all timeslots for a given room,

$$O_j^P = \left(\frac{\sum_P O_j}{p}\right) \tag{3.4}$$

O_j^P represents the occupancy of room j averaged over all timeslots P . Finally, the utilisation U_j for a single room j , is the product $F_j O_j^P$.

Therefore, the **total utilisation** U, is the weighted average of U_j over all rooms. Optimising

the BOS used will imply optimising U.

Location Penalty (L) : The intention of the location penalty is to model how well the rooms match the events allocated to them in terms of physical location. Here, this means that the allocation of an event E to a room R should receive a penalty depending on the estimated distance between E and R which is the distance between the department owning the room, and the department owning the event allocated to it. (We will treat minimisation of the `L` penalty as an objective; or specifically will be maximising `-L`.) In the absence of any data for physical distances we have arbitrarily selected a set of penalties based purely on the departments involved. When an event E is owned by a department, and this is assumed to be its natural home, then entries in the matrix $\mathcal{L}_{N\times M}(L_{ij})$ are equal to zero. The diagonal is therefore zero in the matrix. The `L` penalty is non-zero only if the event is owned by a department different from that which owns the room. Basically, this just encourages events to be placed within their owning department. (Such location penalties are also likely to be fairly natural, even if the allocation decisions are made be a central administration: lecturers and students will generally prefer to remain close to their "home" department.)

Notice that the location penalty depends only on the room and event, and so if we take a linear combination, then the goal would be to optimise BOS and `-L`; the `L` penalty and BOS score together will form a new objective.

Timetable Penalty (TT): Teaching space allocation is also constrained by timetabling needs, and this constraint has utmost importance in academic universities. In order to take

some account of the effects of timetabling we introduce a conflict graph between events. In the absence of data we have used various simple *randomised generators* for the TT conflict matrix. An enhanced real-world "student enrolment" generator is explained in **chap.** 6, and will replace the *randomized generator* when creating conflict matrices in future work. Our model is again based on the owning department for each event. Specifically, we generate conflict matrices denoted by $TT(p, q)$ according to:

- conflicts between events from the same department are generated randomly with probability p.
- conflicts between events from different department are generated randomly with probability q.

The case $p = q$ corresponds to ignoring the department, and we will refer to $TT(p, p)$ as simply $TT(p)$. Another simple case is $TT(100, 0)$: the conflict graph that has edges between any two events in the same department but none otherwise. This corresponds to expecting that events from the same department are more likely to have students in common, or simply that departments strongly prefer that their own events do not clash. We expect that the TT constraints will capture some of the broad effects of real timetabling problems. Note that the departmental owner of an event is fixed, and is not necessarily the same as its allocated location, though in cases of a mismatch the assignment would generate an `L` penalty as explained earlier.

3.3.4 Mathematical formulation

Let's call, first, each *(room,timeslot)* pair a `roomslot`, then we tend to assign events to roomslots, and to maximise the allocated seat-hours. In the **absence** of other constraints or objective functions, it is well-known that this is just a standard assignment problem, and reduces to a maximum weight matching problem in a bipartite graph (see Garey and Johnson (1979)).

Roomslots denote the available space to which events are allocated. So having r rooms and p timeslots per room, the number of roomslots would be rp. A roomslot represent a room taken at any given one time, and have therefore the same capacity as that room.

Model `EALLOC` :

given:

N : set of all events

n : total number of events, $n = |N|$

P : set of all timeslots

p : total number of timeslots, $p = |P|$

r : total number of rooms

M : set of all roomslots

m : total number of roomslots, $m = |M|$ and $m = rp$.

S_i : number of students enrolled in event i

C_j : capacity of roomslot j

$\mathcal{L}_{N\times M}(L_{ij})$: location matrix between event i and

roomslot j with entries L_{ij}.

$C_{i_1 i_2}$: Conflict matrix between different events i_1, i_2

B_L^{up} : upper limits on the location (L)

$T^z = \{j \quad | j \in M : z \in P\}$: represents the set of roomslots corresponding to a given timeslot z, with $M = \bigcup_{z \in P} T^z$ and $\bigcap_{z \in P} T^z = \emptyset$.

For every timeslot $z \in P$, T^z is the subset which includes all roomslots of timeslot z. T^z is the set of roomslots considered at a given one time.

We use the following binary decision variable:

$$y_{ij} = \begin{cases} 1 & \text{if an event } i \text{ is allocated to roomslot } j. \\ 0 & \text{otherwise} \end{cases}$$

Objective and Constraints The objective is to maximise the total BOS used:

$$Obj^* = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} S_i y_{ij} \right) \tag{3.5}$$

subject to the following constraints:

Capacity of an event should be less than or equal the capacity of a roomslot:

$$S_i y_{ij} \leq C_j \qquad \forall i \in N, j \in M \tag{3.6}$$

Only one event can be allocated to a given roomslot :

$$\sum_{i=1}^{n} y_{ij} \leq 1, \quad \forall j \in M \tag{3.7}$$

An event cannot be assigned to more than one roomslot:

$$\sum_{j=1}^{m} y_{ij} \leq 1, \quad \forall i \in N \tag{3.8}$$

Location constraint: The location penalty must be less than or equal the upper limit B_L^{up}:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} L_{ij} y_{ij} \leq B_L^{up} \tag{3.9}$$

When $B_L^{up} = 0$, hard location penalty is enforced.

One can the change the location constraint to an objective, by using a linear combination of location and total BOS used :

$$Obj^* = W(BOS).\Big(\sum_{i=1}^{n}\sum_{j=1}^{m} S_i y_{ij}\Big) - W(L).\Big(\sum_{i=1}^{n}\sum_{j=1}^{m} L_{ij} y_{ij}\Big) \tag{3.10}$$

$W(BOS)$ and $W(L)$ are weights, set to give preference to one objective over the other. This would be used in coming sections to plot the trade-off surfaces.

Timetabling constraints: Constraint 3.11 forbids two events with common students to be allocated to the same timeslots.

$$C_{i_1 i_2}(y_{i_1 j_1} + y_{i_2 j_2}) \leq 1, \quad \forall \theta \in P, \forall j_1 \in T^\theta, \forall j_2 \in T^\theta, \tag{3.11}$$

$$\forall i_1 \in N, \forall i_2 \in N \tag{3.12}$$

$$i_1 \neq i_2, j_1 \neq j_2; \tag{3.13}$$

The conflict matrix $C_{i_1 i_2}$, has been generated using the randomized generator for the `TT`

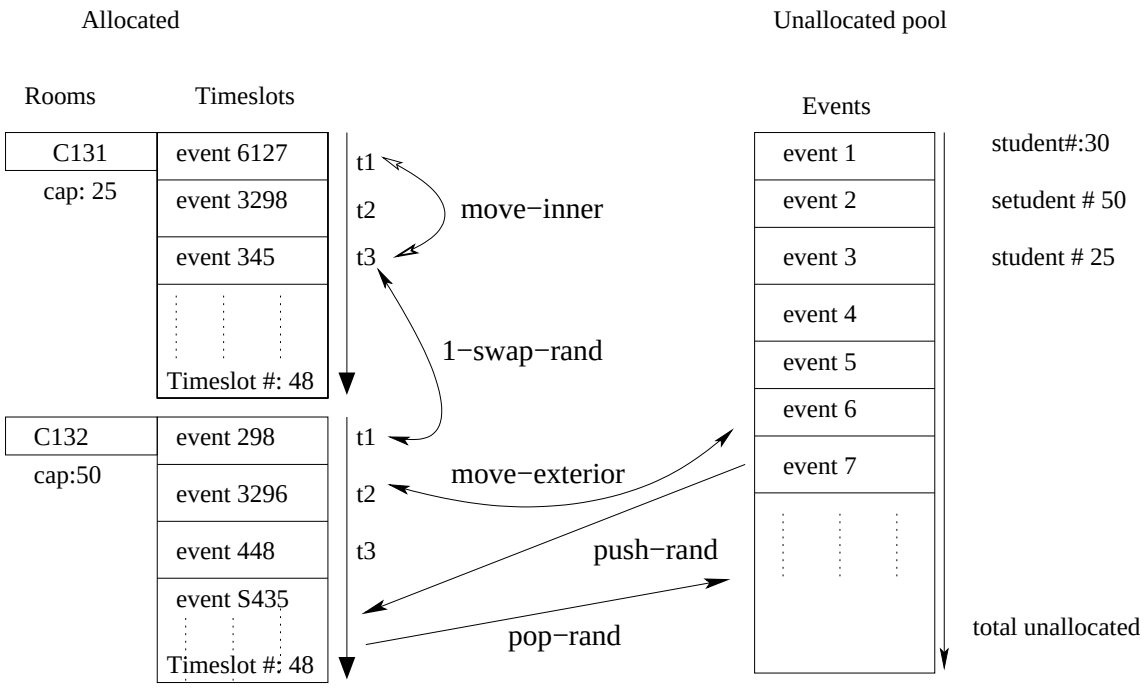

FIGURE 3.1: Schematic of the local search operators (except ***2-swap-rand***) for both `HC` and `SA`.

penalty.

3.4 Heuristic approach: optimisation problem

In this section, we present a heuristic approach to the problem. We use two local search algorithms (Hill Climbing `HC` and Simulated Annealing `SA`), with specialised operators to explore the search neighbourhood of solutions.

3.4.1 Local search operators

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 3.1

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.

Push-rand Randomly selects one event 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 unallocates it.

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

3.4.2 Meta-heuristics

We use Hill-Climbing `HC` and Simulated Annealing `SA` (Kirkpatrick *et al.*, 1983; Aarts and Korst, 1990) implementations in this chapter. The `HC` variant uses 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[3] than the incumbent, we commit to the move, but otherwise disregard it.

`SA` was used as the main component for overcoming local optima (mainly in the presence of the timetabling penalty). As it is known, simulated annealing, which evolved from an algorithm for Statistical Mechanics by Metropolis (Aarts and Korst, 1990), avoids getting stuck in local optima by probabilistically accepting, non-improving candidate solutions. The acceptance criteria depends on the current `temperature` of the algorithm. Given a fitness function $\mathcal{F}$, and $\Delta\mathcal{F}$ the change in fitness between two neighbouring solutions, then the probability P of accepting non-improving solutions is given by:

$$P = e^{\frac{\Delta\mathcal{F}}{T_k}} \tag{3.14}$$

where T_k is the current temperature of the system and is generally cooled according to a cooling schedule. In our algorithm the fitness function is the weighted linear combination of total utilisation U plus the Location (L) and timetabling (TT) penalties:

$$\mathcal{F} \quad = \quad W(U) \cdot U + W(L) \cdot (-L) + W(TT) \cdot (-TT) \tag{3.15}$$

where the W(*) are simply weights associated with each objective or penalty and change

[3]Represented by **Eq** 3.15.

according to a preference or used to derive the pareto fronts as will be discussed in next section.

A geometric cooling schedule was used in our algorithm, specifically temperature $T_{i+1} \rightarrow \alpha T_i$ every 650 iterations with $\alpha = 0.998$. We generally used 6 million iterations and initial temperature $T_i = 0.6$. Such a slow cooling over a large number of iterations was chosen to err on the side of safety. Tuning the SA parameters, such that we gain confidence on the quality of solutions provided was compared with the `IP` solver (CPLEX) on a smaller dataset. (Details shown in figure 3.2).

3.5 Pareto fronts

In the following set of experimental results we show, quantitatively, the effect of adding the L and TT penalties on the utilisation U. Adding those penalties will unlikely increase utilisation levels, instead they will drive it down. However, the main issue is the magnitude of such an effect, and in particular, whether the effects can be large enough to be a substantial cause for the low values of utilisation seen in practise. Hence, we are interested in reducing the upper estimates on utilisation, and will enable the solver to select those events that will be allocated so that U is maximum. We will treat the system as a multi-objective problem using the utilisations, and (the negatives of) the location penalty L, and timetable penalty, TT, and determine the appropriate (approximate) Pareto fronts (**Ref.** Deb (2005); Steuer (2003) for descriptions of the concepts of Pareto optimisation).

Let's first consider the simpler two-objective sub-cases.

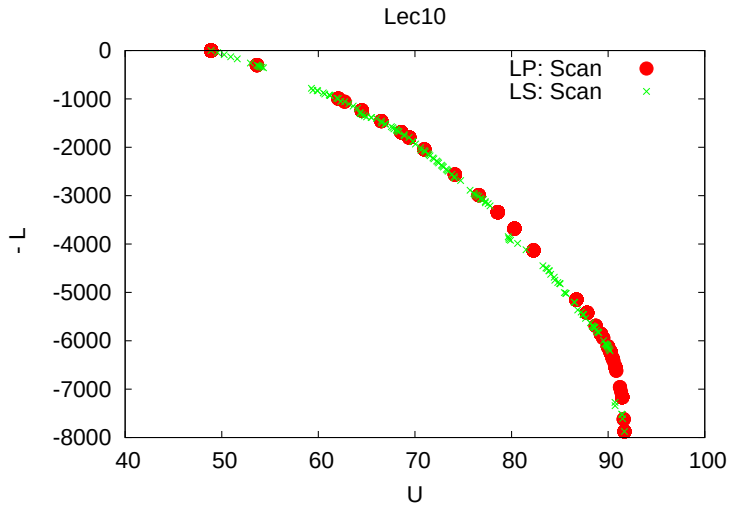

FIGURE 3.2: L versus U. with 10 timeslots. LP scan uses the exact solver (CPLEX) and LS uses the local search method

3.5.1 Bi-objective: Utilisation versus Location

In finding trade-off surfaces, we follow the simple, standard procedure of generating points by taking many possible linearisations of the problem. That is, data points are generated by running the solver(s) for different discretized values of the objective weights. The specific weights of every objective were altered by increasing it by a constant factor (chosen in a way proportional to the scale of the objective (the estimate is by trial and error)) such that the change in the specific weights would allow a wide exploration of the pareto front.

Figure 3.2 shows the results of such an exploration. Note that in fact we plot `-L` rather than the location penalty L itself, merely so that we meet the convention that all axes correspond to maximisation problems, i.e., "better" means towards to the top right. Also, for clarity, in all such plots we remove all dominated points.

The first set of points are obtained by taking a linear combination of the weights

(for U and L), solving model EALLOC using CPLEX 10, (with objective defined in equation 3.10). Each such solution is hence Pareto optimal. However, some Pareto optimal solutions might not be reachable in this way ("unsupported" solutions, Deb (2005); Tuyttens *et al.* (2000)), and this leaves gaps in the experimental Pareto Front, hence, those gaps were reduced by using an alternate method: by setting a given threshold, on the location penalty, then optimising the utilisation, we can plot the different optimal U levels for various location thresholds. Using this method the plots were clearer with less gaps in between different runs. The final set of points in Figure 3.2 are obtained using our local search method (SA). It gives points that are also very close to optimal, the difference being small enough so as to not significantly change the shape of the curve. This gives us confidence that our SA method will not be distorting later results. SA was run many times, with every run correponding to one point in the graph. Problems uses dataset Lec in the appendix. Note that even if the location penalty has the potential to decrease utilisation to near 50% value, the real world case is actually between 92% (which the max in the absence of L) and 50%, since one might relax this penalty a bit, not driving it as low as 0. Indeed, one can notice, from this figure, that if we relax L by 1/8th of it's whole range (that is allow -L to be 1000), one can gain nearly 12 % in utilisation. In an environment where space is very scarce, such a gain can be justifiable.

3.5.2 Bi-objective: Utilisation versus Timetabling Penalty

Figure 3.3 shows the trade-off between utilisation U and timetabling TT. Introducing the timetabling penalty can drastically decrease the utilisation to near 35% level. Timetabling penalty in our model and in real world cases is a crucial player, rightly affecting how rooms,

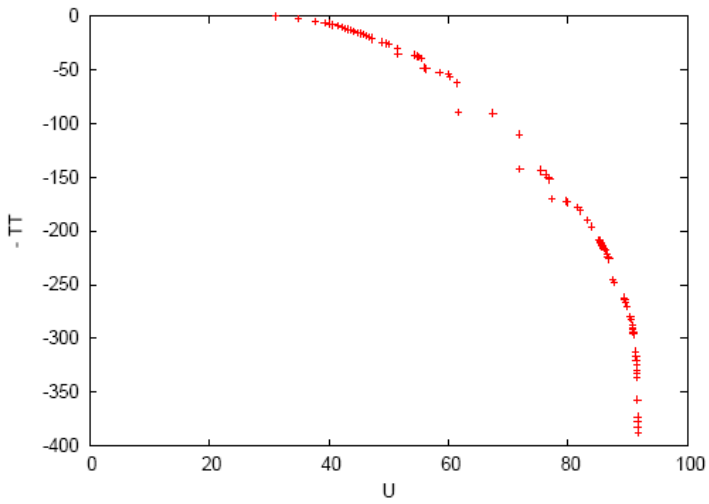

FIGURE 3.3: The trade-off between Utilisation,U, and Timetable Objectives, TT, dataset with 10 timeslots.

timeslots and events interact between each other. While one can generally relax other constraints (location for example) it's generally an imperative requirement that timetabling penalty be obeyed and no clashes occur between events. For example, a 35% U given TT=0 is much different scenario than 50% U given L=0.

3.5.3 Three Objective Problem: U vs L vs TT

In this set of experiments we combine the three objectives together and study their interaction and their effect on the utilisation. We fix the weight for the U objective and produce points by scanning many different weights for L and TT (meaning TT(100,0)), and optimising each one separately. The results of this for rooms with 10 timeslots are given in Figure 3.4 .

As explained in the caption, the 3 dimensional data is converted to 2-dimensional by giving U and -L to the x and y axes. The timetable objective is represented by grouping

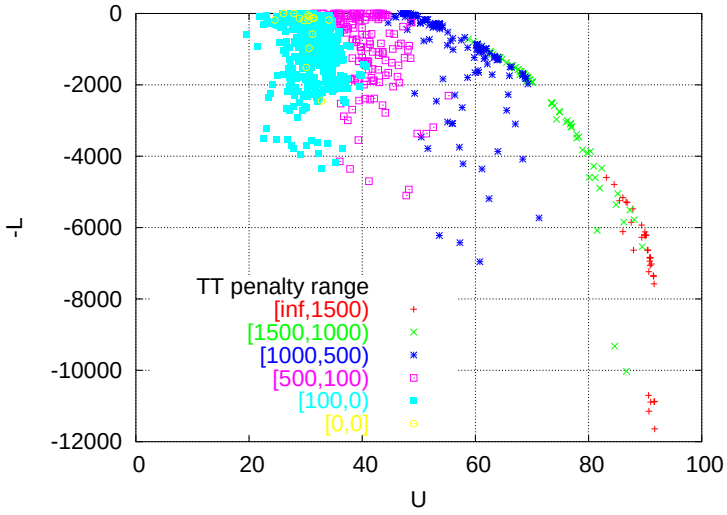

FIGURE 3.4: U vs L vs TT for dataset with 10 timeslots. X and Y axes are for Utilisation `U` and Location `L`. Different colors represent different values for the timetabling penalties. e.g [200,100) means $200 \geq TT > 100$ and `inf` meaning infinity.

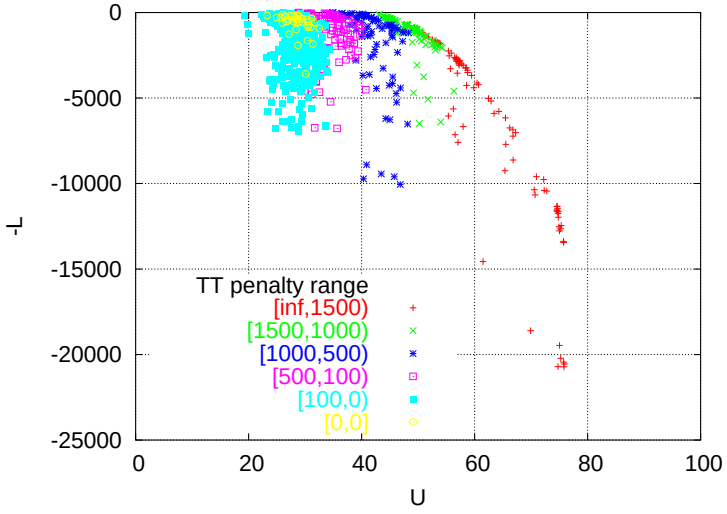

FIGURE 3.5: Same as Fig 3.4 but with 18 timeslots.

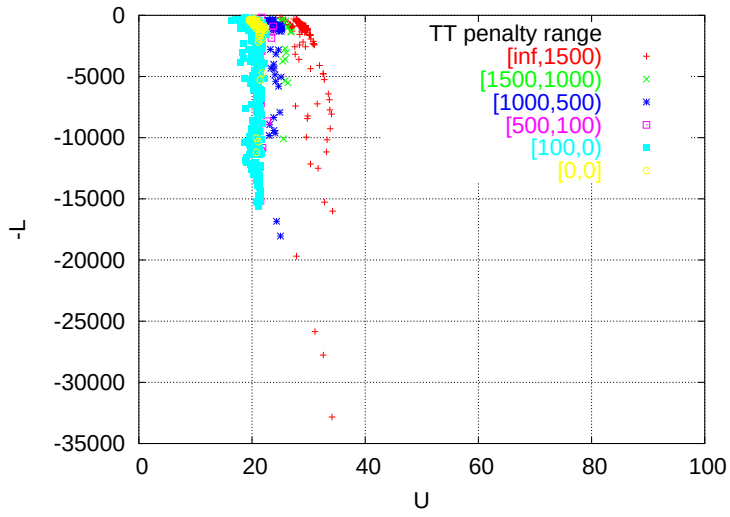

FIGURE 3.6: Same as previous figures but with 50 timeslots.

points into different sets according to their TT penalty, and plotting using different point patterns (and shades). Sets further down the key correspond to being better for the TT. We see that the maximal U is 92% but this is reduced to 29% if the L and TT penalties are forced down to zero.

Figures 3.5 and 3.6 gives results for data-sets with 18 and 50 timeslots. Primarily, note that by increasing the numbers of timeslots, utilisation decreases. This is merely because the seat hours needed by the events is less than the available seat-hours as soon as the number of timeslots is greater than 18. Also, with larger number of timeslots, the chance of having more roomslots under filled (e.g. lower occupancy) is higher. Notice that L and TT are having a significant effect even for the case with 50 timeslots.

Also, in figure 3.6, there are sufficient seats available that in the absence of any L and TT constraints, all events can be allocated, this leads to the value of U=37% on the righthand edge. However, such complete allocations have large L and TT penalties. If

the L and TT penalties are driven down then U is also driven down significantly. That is, even when the available seat-hours is twice the requested number, L and TT can provoke that some events must remain unallocated.This supports our hypothesis that the location and timetable penalties have the potential to dramatically drive down utilisation, and so are a reasonable candidate to explain low utilisations in the real world. furthermore, if we request any utilisation beyond 40% then we are very likely to be able to satisfy about 95-98the request, but not the 100% we would expect in a safe region. We suspect that this indicates that there is a mismatch between rooms and events.

Note, the nature of the data-sets used are such that if we use 50 timeslots per room, lectures become substantially under-subscribed, in the sense that the total demand for seat hours and timeslots from the events is much smaller than the supply of seat-hours and timeslots from the rooms. In order to explore a wider range of these supply-to-demand ratio we need to do one or more of (i) add more events, (ii) reduce the number of rooms, or (iii) have fewer timeslots per room. We opt against creating more events, as it would make the problems unnecessarily large. The options of reducing rooms or reducing timeslots are similar in that they reduce the available seat-hours. also the variety of room sizes has probably a role to play in setting utilisation levels, and so do not want to change any room profile at the moment (this issue is investigated in **chap.** 7). Instead we uniformly reduce the timeslots for all rooms. Hence we created instances with 10, 18, and 50 timeslots.

3.6 Impact of Location and Timetabling penalty

As mentioned previously, L and TT are having a significant effect on utilisation. There are sufficient seats available that, in the absence of any of these 2 constraints all events can be allocated, and this leads to the value of U=30% as a nominal, average value across many institutions. Clearly, if the L and TT penalties are to be forced down, then U will significantly improve. Also, even when the available `seat-hours` is twice that requested, L and TT can mean some events must remain unallocated. Perhaps, in the context of achieving a given utilisation level, our aim should also be oriented towards looking at achieved utilisation versus specific random requests. For example, what would be an achieved level of U, given a requested notional level, under L and TT constraints. The effect of those 2 constraints can be studied indirectly rather then explicitly varying their intensity and looking at the changes in the utilisation levels. Let alone, a deterministic change in U and L, or U and TT, is highly indicative but still insufficient in determining future evolution or maybe current space utilisation in many institutions. The results in this chapter support our hypothesis that the location and timetabling penalties have the potential to dramatically drive down utilisations, and so are a reasonable candidates to explain low utilisations in the real world. But we are yet to prove the notion of achieving a given utilisation level and the ability to meet different random requests, under different space requirements and constraints. We will delve deeper in this aspect in later chapters, by analysing the stochastic nature of the problem in hand, and show that they match the conclustions extracted from this chapter.

3.7 Summary and conclusions

In this chapter, we looked at the pure event allocation problem from an optimisation perspective and aimed to understand the factors that have the potential to explain low utilisations observed in academic institutions; this is a first, preliminary step in an attempt to improve utilisation in practice. We have devised exact and heurstic methods that have provided an initial, unsurprising insight to major constraints affecting space usage. We have shown that constraints that mimic timetabling and physical locations, reduce the utilisation levels in universities to the more realistic and observed levels. This chapter provides the foundation for the coming work, in including the timetabling and location penalty and using similar methodologies on other more important aspects affecting teaching space allocation issues. We used Simulated Annealing and Hill Climbing algorithms when dealing with datasets intractable by exact IP solvers. The constraints used here do not cover the case where "non-lecture" teaching activities require allocation, (e.g. tutorials) where most of the events do not fit in the available rooms and therefore require splitting which is the subject of the next chapter.

Chapter 4

Splitting in Teaching Space Allocation: The Optimisation Approach

That which is static and repetitive is boring. That which is dynamic and random is confusing. In between lies art

John A. Locke

4.1 Introduction and motivation

In this chapter, we introduce and describe the teaching space allocation problem with splitting, where the focus is on teaching activities that do not generally fit into available space and need to be broken up into multiple groups (e.g. sections). We propose an algorithm to perform "dynamic splitting" in the presence of different constraints and compare it to a static[1] way of splitting, widespread in many institutions. A multi-objective optimisation

[1]Refer to **sec.** 2.4 for details on different types of splitting.

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. We will explore such trade-offs in this chapter using a dynamic splitting strategy embedded in the local search.

Outline of the chapter Section 4.2 gives the basic description of the problem constraints and objective functions, and a brief description of the data sets. We study the splitting algorithms in section 4.3. In section 4.3.1 we outline a form of local search that does not include splitting, but which forms a good basis for the algorithms for splitting (Dynamic splitting) presented in section 4.3.2. In Section 4.4 we compare the performance of the various algorithms. In Section 4.5 we move to the exploration of the solution space itself, presenting results for the trade-offs between the various objectives.

4.2 Problem description

Teaching space allocation with splitting, is concerned with allocating teaching events (classes/course offerings, tutorials, seminars) to rooms and times. We follow, in this chapter the same terminology described in **sec.** 2.2. We introduce some of the constraints and objectives that arise when splitting is involved.

4.2.1 Classes, groups and rooms

Most of the work in this chapter is centred around the `class` and `group` levels of our model (see **sec.** 2.2). Hence, for classes we have:

1. **Size**: Number of students.

2. **Type**: type of teaching activity (Lecture, Seminar, Tutorial, etc.) of the `class/group`.

3. **Department**: Department offering/managing the class.

Classes will generally be split into groups, which should be assigned to a single room and timeslot. In this model, classes have the same information as modules except that each takes only a single timeslot.

For every *room* we have:

1. **Capacity**: Maximum number of students in the room.

2. **Timeslots**: The number of timeslots per week.

3. **Spacetype**: type of teaching activity (Lecture, Seminar, Tutorial, etc.) for which the room is more suitable.

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 a `class/group` cannot exceed the room capacity

2. **No-sharing constraint**: At most one `class/group` is allowed per roomslot.

In this chapter, we will also apply the condition that the spacetype of a `class/group` 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 this is studied in **chap.** 7. So, henceforth, in descriptions of the algorithms we will ignore spacetypes.

4.2.2 Penalty and objective functions

Merely allocating teaching events to roomslots 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 described in **chap.** 3, and also from considerations of what a good allocation is likely to mean in the presence of splitting, we use the following:

Utilisation (U)

The primary objective is that we want to make the best 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 roomslot. Utilisation is defined in **sec.** 3.3.3

Timetabling (TT)

Teaching space allocation is also constrained by timetabling needs, details of which has been explained in **sec.** 3.3.2. Hence we use here a timetabling penalty (TT) that is just a standard conflict matrix between events; a set of pairs of events that should not be placed at the same timeslot. We will simply use randomly generated conflict graphs for now. 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%.

Location (L)

In **sec.** 3.3.2, we have detailed the motivation behind the location penalty, merely reducing the physical travel distances for students between taught events, seeming likely that students and faculty would prefer that the events they attend will be close to their own department. As in **sec.** 3.3.2, 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 roomslot. Specifically, if a `class/group` 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, $Y(D(i), D(r))$. All `classes` 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.

Group Size (GZ):

For teaching events such as tutorials or seminars it is standard practice that these events are taught in small groups. Various institutions sets out to have group size preferences for such kinds of events. Hence when splitting, we need to be able to control the sizes of groups. In this chapter, we use a simple model in which we take a target size for the groups, and simply penalise the deviation from that target. Given an allocated group i, let the number of students be s_i, the total number of allocated groups be I, and the target group size T. The group size penalty GZ that we use is

$$GZ = \sum_{i=1}^{I} |s_i - T| \tag{4.1}$$

Disregarding this penalty, would generally condition the Group Size to be bounded by the room capacity. The target group size generally varies with different spacetypes. Specific values are empirically estimated from previous years average group sizes.

Group number (GN) :

Every group will need a teacher, and so the total number of groups allocated will have a cost in terms of teaching hours, and should not be allowed to become out of control. The penalty GN is simply the total number of allocated groups. Pressure to minimise GN will tend to discourage class splitting. When splitting, a class transmits all conflicts to it's groups: say class A, conflicts with B and C, then upon splitting the new formed groups of this class , A_1 and A_2 will both conflict with B and C. In coming chapter we will discuss the `partial inheritance` aspects in more detail.

No Partial Allocation (NPA) :

The context in which we do the search is that we have a large pool of classes available and we should choose the best subset that can be allocated. However, if a class is broken into groups, then the class as a whole ought to be allocated or not. The `NPA` penalises those cases in which some of the groups of a class are allocated, but other groups from the same class remain unallocated. Enforcing `NPA` as a hard constraint would disallow partial allocation: for every class, either all its groups are allocated, or not.

4.2.3 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 objective or fitness F, which can be represented as follows:

$$\begin{aligned} F &= W(U) \cdot U + W(L) \cdot (-L) + W(TT) \cdot (-TT) + \\ & \quad W(GZ) \cdot (-GZ) + W(GN) \cdot (-GN) + W(NPA) \cdot (-NPA) \end{aligned} \tag{4.2}$$

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. Using weights is also intended to allow modelling of the way that administrators will relax some penalties and tighten others.

4.3 Splitting algorithms

In this section, we describe the splitting heuristics that are incorporated into the hill-climbing (HC) and the simulated annealing (SA) approaches (**sec.** 3.4.2). Two strategies are implemented: a) constructor-based splitting, and b) dynamic local search-based splitting. In the first case, the group size is calculated during the construction of an initial solution

and remains fixed for all teaching events throughout the local search. In dynamic splitting, the group size is calculated as the local search progresses according to the size of the class (and room capacity) that is being allocated. Hence, we will have:

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

4.3.1 Static splitting

In static splitting we select a target group size (generally based on room profiles) and then split all the classes of size larger than that target size, into as many groups 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 (hill-climbing or simulated annealing) to improve the initial solution. So, in this strategy, splitting happens within the constructor and this provides no flexibility in changing the group size during the local search.

There can be many ways to calculate and fix the target group size. Here we compare three variants which are based on the notion of a "target room capacity". This means that the target group size is calculated based on the capacity of the rooms that are available for allocating groups. Specifically, the target group 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 group 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 compare it to the case in which splitting is carried out during the local search (dynamic) which is described in the next subsection.

4.3.2 Dynamic splitting operators

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

Dynamic splitting is embedded in the local search in such a way that there is freedom and diversity in the choices of group sizes. Thus the dynamic splitting operator considers not only the room capacities (as in the case of the of static splitting) but also

the location (L), timetabling (TT), group number (GN), group size (GZ), and no partial allocation (NPA) constraints. Note that, at the current stage, the operators themselves do not directly respond to an increase/decrease in penalties, this is tackled directly 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 issue in future work.

It is important to note that in this dynamic splitting approach, the "pool of unallocated classes" is a pool of the portions of classes that are not yet allocated. The unallocated portions contain no information about how they will be split. That is, it is not a pool of groups waiting to be allocated, but instead the groups are created during the process of allocation. The main characteristic of the dynamic splitting operators lies in the fact that when a split occurs, we actually select a fraction of a class and allocate it. When a group is unallocated, we merge it back with the associated class without keeping track of previous group splits.

We detail below, the neighbourhood operators used in the dynamic splitting (ordered roughly by their degree of elaborateness):

1-swap-rand-grp: This operator works as ***1-swap-rand*** described in (chapter 3, section 3.4.1) but the move is carried out between 2 groups (not necessarily of the same class) from different rooms, and swap them.

Move-inner-grp: This operator works as ***move-inner*** described in (chapter 3, section 3.4.1) but the move is carried out between 2 groups (not necessarily of the same class).

Push-rand: This operator works as ***push-rand*** described in (chapter 3, section 3.4.1) but note that the events being 'pushed' to the allocation are groups of a class that are smaller than the chosen room, and so no splitting was needed.

Pop-unsplit. This operator is used to remove groups from their allocated room and unsplit/rejoin groups with their unallocated parent class. 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 group from a randomly selected room. In the case that the chosen class is a group, the operator unallocates the group and merges it with its unallocated parent event. If the class is not a group it is simply added to the unallocated pool.

Split-push: This operator is used to handle classes whose unallocated portion is larger than the chosen room, and is the main operator that is used to create new groups. 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 class 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 roomslot t_j
5 Create group S_i with size s
  and resize the remainder P_i
6 Set that S_i inherits all conflicts from class P_i (see **chap.** 6)
7 Generate candidate move by allocating S_i to room R_j in timeslot t_j

Note that $rand(\delta, 1)$ means a number selected uniformly at random from the interval $[\delta, 1]$ and the parameter δ is described below. After randomly selecting a roomslot and unallocated class, the main step in this operator is to decide how to split the class to create a new group. Assuming that the capacity of the room is smaller than the size of the remainder of the class, the new group size, s, is calculated by multiplying the capacity of the room by a randomly selected factor. The factor depends on a *"group 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 group size, s, will be between 40% and 100% of the selected room's capacity. The intention of this randomised selection of group size is that it enables the search to discover group sizes that match the penalties such as group size and group number. The new group inherits all of the conflict information from its parent class – see the discussion of the "Conflict Inheritance Problem" in **chap.** 6. The new group is then allocated to the chosen room. The remaining part of the parent class is left in the unallocated list of classes with its size reduced appropriately.

Split-max: This operator is a version of ***split-push*** with δ=1 and is designed so that classes with size larger than the chosen room are split into groups with the maximum size allowed to fit in the chosen room.

4.3.3 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 4.1. Two classes C1 and C2, of sizes 120 and 40

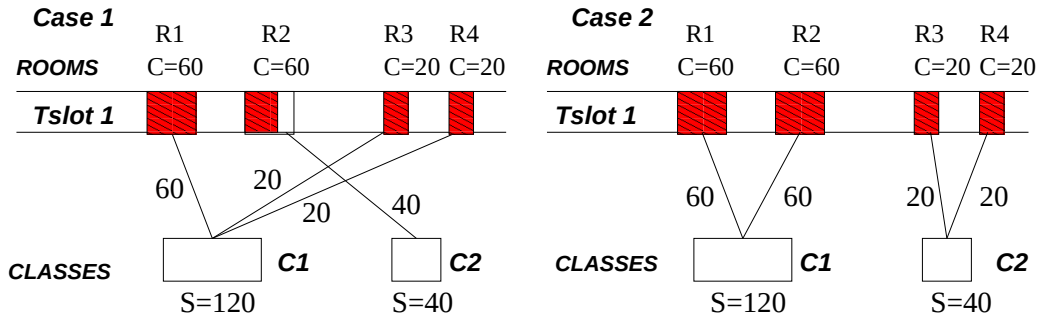

FIGURE 4.1: Example in which applying operators to split classes has different effects. In case 1, class C2 first receives a ***push-rand*** into room R2, and then applications of ***split-push*** to C1 are able to allocate only 60+20+20=100 students rather than the needed 120. However, in case 2 we see that reversing the order allows all of both classes to be allocated.

respectively, are to be allocated to the four rooms available. We have selected capacities so that total size of classes equals the total capacity of the rooms. In the first case, the smaller class C2 is allocated first to room R2 via a ***push-rand*** move without a split. This inevitably means that 20 spaces within room R2 are wasted, and so later it becomes impossible to fully allocate all of class C1. However, in the second case, the larger class C1, is first split using ***split-max*** and then we end up with a perfect fit of class C1 to rooms R1 and R2, and class C2 to rooms R3 and R4. The operator ***split-max*** with its implicit "maximum size groups first" is often better at maximising the utilisation; though there are other cases in which ***push-rand*** is necessary, like when we are required to achieve a target group size, for example. For this reason, and also because of preliminary experimentation, we tend to give the operator ***split-max*** more probability of being selected than the operator ***push-rand***.

4.3.4 Controlling the search

We have also observed, that the effectiveness of each operator varies during the search. As an example, suppose we are just doing non-splitting local search as described in section 4.3.1.

We start with an empty allocation, and then the ***Push-rand*** operator is most important and successful in the early stages as classes need to be allocated. But due to limitations in the capacity of rooms, this operator stalls for 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.

4.4 Experimental comparison of splitting algorithms

In this section, we first investigate the "static splitting" method in which only the constructor does splitting followed by local search in the form of Simulated Annealing, CONS-SA (CONS-HC is not presented as, unsurprisingly, it performs no better than CONS-SA). We find that it is far inferior to the dynamic splitting approach. Moving to the dynamic splitting itself we then compare the HC and SA variants, and will see that the DS-SA variant is the better.

Considering the datasets in Appendix A (table A.1), we illustrate the importance of splitting in our scenario. Table 4.4 compares some examples of the utilisation percentages obtained, and the number of events allocated, without any splitting (not even static splitting from the constructor) and compares them with those obtained by DS-SA:

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

We clearly see that splitting is essential for the tutorials and seminars as, otherwise, virtually nothing is allocated. For the workshops, some classes can be allocated, but we still lose a lot compared to when splitting is allowed. So from now on we always permit splitting. While, in the results above, utilisation figures seem a bit higher than those found in the

real world cases (30-40%) we show later that constraints drive the utilisation down to more practical levels.

In general, our results present trade-off curves which are approximations to Pareto Fronts. These curves represent the compromise between two of the objective functions: We select a wide range of combinations of weights associated for the two chosen objectives, and 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, the gaps are generally 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).

4.4.1 Dynamic vs. static splitting

Figure 4.2 shows the trade-off curves between utilisation and location for the three different methods from the static splitting, MAXCAP, MINCAP, AVGCAP (see subsection 4.3.1), and compares them to the results from the dynamic splitting method, DS-HC.

We see that for the static case, splitting based on the average room capacity (AVG-CAP), outperforms the other two methods (MINCAP and MAXCAP). This is reasonable because when splitting by the smallest room capacity there is capacity wastage in larger rooms and when splitting is based on the larger room capacity there is also wastage caused by violating room capacities, since we cannot allocate a group to a room with smaller

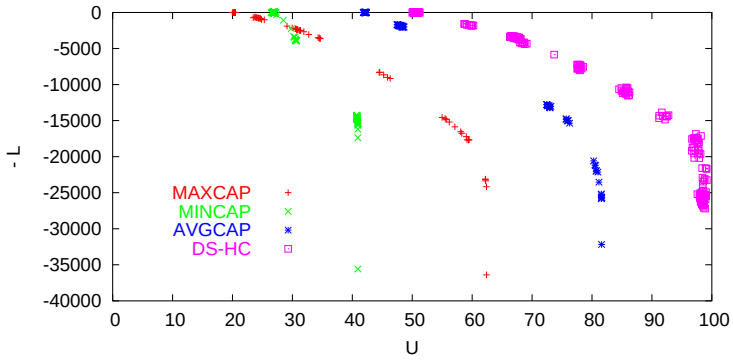

FIGURE 4.2: Comparison of dynamic and static (constructor-based) splitting for the Wksp data set. Plots give the trade-offs obtained between utilisation and Location; all the other objectives being disregarded (W_{TT}=W_{GZ}=W_{SN}=W_{NPA}=0). The first three sets of points are from the three static splitting methods and the last set from the dynamic splitting with hill-climbing.

capacity.

However, it is also clear that all our constructor-based splitting methods are easily outperformed by the dynamic splitting. This is unsurprising, as it is 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 chapter we will henceforth consider only dynamic splitting.

4.4.2 Dynamic splitting: HC vs. SA

Figure 4.3 illustrates the different performances of DS-HC and DS-SA on the `Wksp` problems in the presence of timetabling. Figure 4.4 is the same except that it is for a tutorials dataset (see Appendix. A). 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

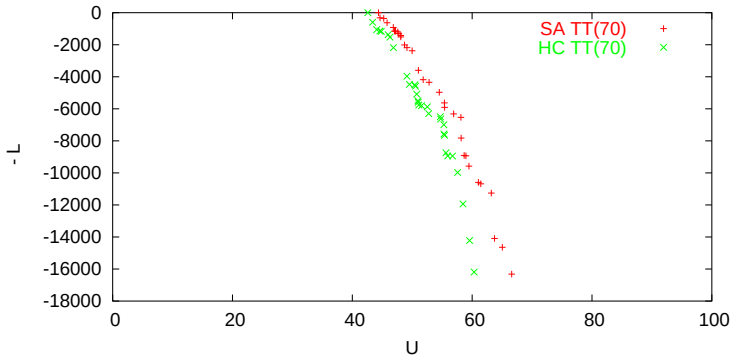

FIGURE 4.3: Trade-off of utilisation and location as obtained with dynamic splitting, and using the hill-climbing (HC) and simulated annealing (SA) algorithms. For the Wksp data, and in the presence of TT(70), and no other constraints beside U, L, and TT.

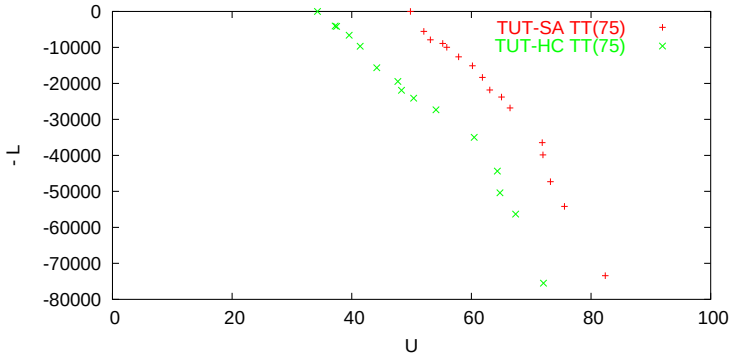

FIGURE 4.4: Same as for Figure 4.3 but instead using the tutorials dataset, `Tut-trim`, and with TT(75).

outperform the HC, as SA can escape local minima but HC cannot. Perhaps more surprising is that the performances in the absence of TT are often very similar. Presumably, without the TT, the search space is rather well-behaved.

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).

4.5 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 almost entirely on the solution space. In particular, we present some preliminary and partial results on how the various objectives interact, and in particular the magnitude of their effect on the utilisation.

4.5.1 Interaction of Group Size Penalty (GZ), Location Penalty (L), and Utilisation (U)

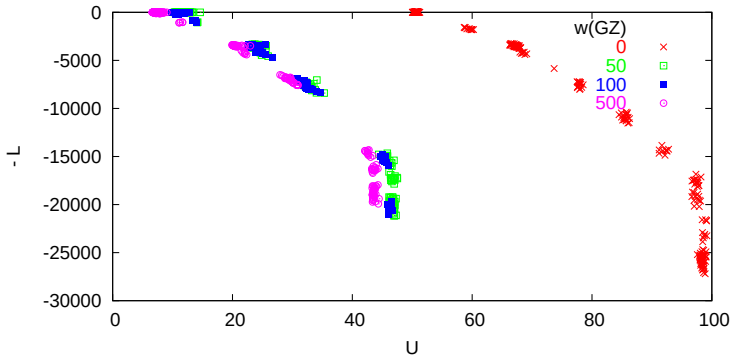

FIGURE 4.5: Trade-off surfaces for the given values of the weight W(GZ) for the group size policy. On the Wksp data-set, with a target group size of 25; and optimising only utilisation U, location L, and group size GZ.

Figure 4.5 gives plots of the trade-off between utilisation (U) and location (L), in the presence of various weights, $W(GZ)$, for the group size penalty (GZ), with a target group size of 25, but with no other penalties. Note that the case $W(GZ) = 0$, was seen previously as the best line in figure 4.2, and illustrates that, even without group 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 $W(GZ)$ 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.

4.5.2 Trade-offs arising from Group Size penalty and Utilisation

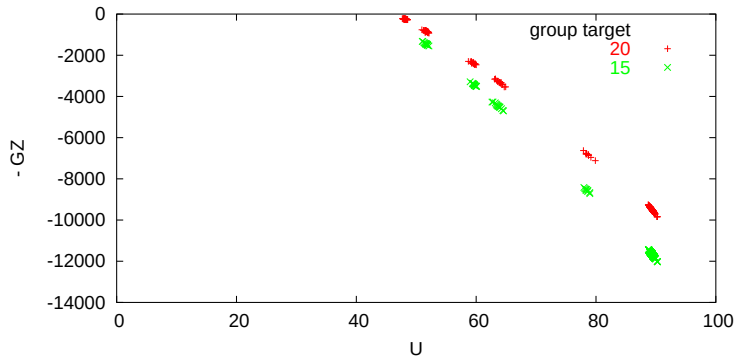

FIGURE 4.6: Utilisation vs. group size penalty, GZ, for the `Wksp` data set, and for two values (15 and 20) of the target group size.

So far, we have only looked at trade-offs between Utilisation and Location. But now, in Figure 4.6 we show the trade-off between utilisation, U, and group size penalty (GZ). This happens to be with a small weight given to the group 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

group size becoming a hard constraint. We also see that reducing the target for the group size reduces utilisations though by a lesser amount.

Part of this effect is possibly because our current group size penalty does not allow a range of values for the group size, and because it penalises under-filling a group just as much as overfilling. In future work, we intend to allow more relaxed and flexible versions of the group size penalty. However, intuitively, it still seems very likely that group size requirements are going to have a strong negative effect on utilisation, and crucially, the methods that we are developing will still allow one to quantify these effects.

Generally, enforcing a soft group size penalty is more realistic than a hard one since flexibility in the group size is quite reasonable; groups aren't always standardized towards a fixed, unchangeable target size and it ought to be possible to vary the target size to suit other constraints.

4.5.3 Effects of timetabling constraints

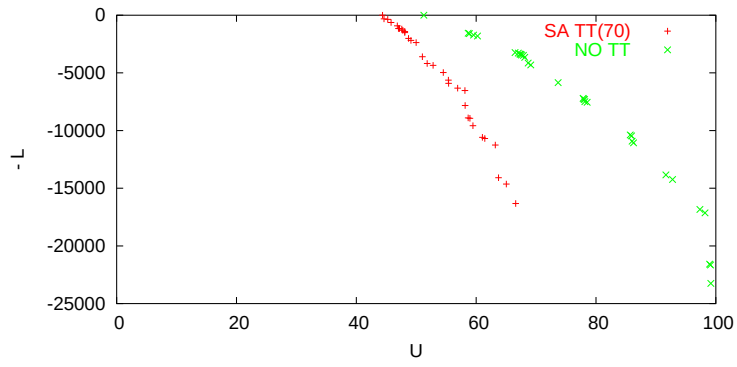

FIGURE 4.7: Trade-offs between Utilisation and Location for the Wksp dataset. "No TT" means that no objectives besides L and U are weighted, in particular W(TT)=0. In contrast, "TT(70)" means that a timetabling constraint, with a density of 70% is enforced as a hard constraint.

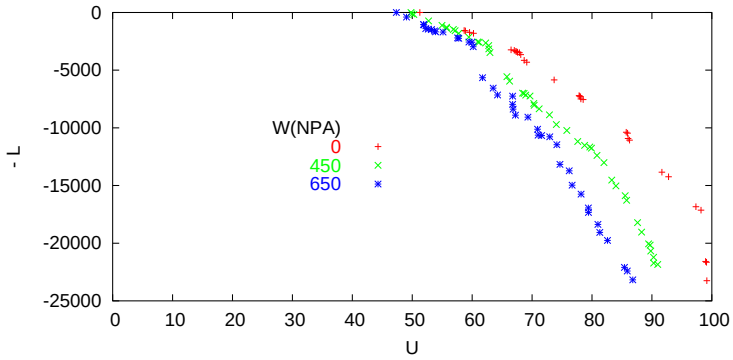

FIGURE 4.8: Trade-offs between Utilisation and Location, in the presence of various strengths of the "No Partial Allocation" (NPA) penalty, but with no TT or other penalties.

Figure 4.7 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. In this case, as we split, all conflicts from parent events are transfered to group sub-events. This meant that timetabling constraints have significantly affected the utilisation, but in fact, as we will show in **chap.** 6, full inheritance is not a realistic aproach to transmitting conflicts, and *partial inheritance* will be discussed further. For now, we can assume that utilisation, under full inheritance, is driven down by the timetabling penalty.

4.5.4 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 groups from a class 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 reduce the utilisations obtained.

The magnitude of this effect is seen in Figure 4.8, We see that giving NPA high weights can further reduce the utilisation by about 10-20%. This is a significant effect, though it is somewhat smaller than the effects seen in the trade-offs with the timetabling and group size objectives. It is also interesting that the effect of the NPA becomes very small when selecting solutions with small location penalty.

4.5.5 Impact and Approach Objectives

Our previous empirical approach of studying the effects of different constraints, stems from the absence of a real-world model that incorporates important factors affecting utilisation. A naive historical way to perform capacity planning, based on utilisation estimates, would have been simply to ensure that the supply exceeds the demand. However, it is very rare that it is possible to use all of the available seats.

Naturally, excess capacity within space 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 between various constraints, mainly timetabling, Location, Group size and utilisation.

If we were interested in doing splitting under a form of "pre-timetabling", as the early stages of timetabling, then we lack the essential information of available students, and will have to rely on enrolment estimates. Effectively, the consideration of the group size constraints will very much depend on those estimates, and so does most critically the timetabling penalty. For space planning, we need to understand which utilisations

are achievable, under varying random requests from year to year. Also how those achieved measures, impact the decision criteria, when it comes to choosing group sizes, in the presence of constraints, such as location and timetabling.

4.6 Summary

In this chapter we looked at the optimisation problem of maximising utilisation while satisfying preferences on splitting and considering constraints like event location and timetabling conflicts. The problem has a multiobjective nature, and we showed some of the characteristics or trade-offs between the different constraints. 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. We acknowledge that other factors apart of those considered in this study are also likely to have an impact on the utilisation inefficiency in real-world problems. The work presented here, forms a first step into a deeper investigation of this issue.

However, this chapter investigated the *optimisation approach* and did not consider the decision problem, of: "given some random subset of classes, can we split and allocate all of them in the available space". Investigating this decision problem (*Fixed choice*) will provide a deeper insight into the nature and behaviour of splitting under a changing number of events and is the subject of the next chapter.

Chapter 5

Threshold Effects on Teaching Space Allocation

Know most of the rooms of thy native country

before thou goest over the threshold thereof

Thomas Fuller (1608-1661)

5.1 Introduction

This chapter looks at the decision problem for the teaching space allocation with splitting: "whether it is possible or not, to split and then fully allocate a given random set of modules within the available teaching space". Using an Integer Programming model for the TSA with splitting, we study the probability that different levels of utilisation are achieved. We show that sharp transitions exist between phases in which splitting and allocation is "almost always possible" to ones in which it is "almost never possible". More importantly, we provide quantitative evidence of the computational cost increasing rapidly as we approach

the *Phase Transition*. The point of this chapter is that even though it's possible to estimate utilisation Levels, the computational cost of achieving them in practice, might be quite high. This will in turn affect the true utilisation levels that can be claimed as achieved. We use Integer Programming and the stochastic (SA) solver of **chap.** 4 to illustrate the point, and consider the splitting case in the presence of location and Group-Size penalties. 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.

5.2 Thresholds and motivation

In the area of graph theory, the study of thresholds has focused on the random graphs Bollobas (1985). 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 **?**.

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. In AI work, hard thresholds were studied for their own intrinsic interest, however, their actual usage was almost exclusively[1] 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 in this chapter, 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] 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. **?**)

5.3 Description and formulation

In this section, we present the terminology used, followed by a Mathematical Programming formulation.

5.3.1 Problem parameters

We have slightly extended the model of **chap.** 4 to become centered around **Module-Class-Group** levels defined in **sec.** 2.2. We now explain the parameters[2] associated with `modules`, `classes`, and rooms :

For every `module`, $k \in \{1, \ldots, q\}$ we associate the following:

1. **size** S_k : Number of students in `module` k.
2. **timeslots** T_k : Number of timeslots required by the `module` in a weekly schedule.
3. **department** d_k: Department administering `module` k.

Other aspects, belonging to `modules`, like special `module` features, or `module` preferences, can be thought of, yet we are not considering them in this study. Also, as explained in **sec.** 2.2, `classes` will carry the same information as their respective `modules` except for only having one time-slots, and also having an associated "type". That is, for a `class` $i \in \{1, \ldots, n\}$ we associate the following:

1. **size** S_i: number of students of the `class` (equal to the number of students of the respective `module`).
2. **type** SP_i: Lecture, Workshop, Tutorial, etc.

[2]Some of these parameters have been explained in previous chapters

3. **department** d_i: department offering/managing the `class`.

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 per week.

3. **spacetype** SP_j: space for Lecture, Workshop, Tutorial, etc.

4. **department** d_j: the one that owns/administers the room.

5.3.2 Penalty and objective functions

The constraints and objectives that have been introduced in (**chap.** 4, **sec.** 4.2), will still apply to this model, except for the timetabling penalty. In chapter 6, we will give proof, that splitting will relax conflicts between classes, and make timetabling effects negligible on the utilisation. The *standard* constraints of **capacity** and **no-sharing**, will be included.

However, more importantly, we also introduce the idea of the solvers having two different modes, free and fixed choice.

Fixed Choice The solver is not given the freedom of choosing which `classes` to allocate to rooms. Instead, the question that we address is "can we fully allocate all the `classes` to the rooms?"

Free Choice The solver is allowed to select which `classes` to allocate so that it maximises utilisation. The question that we address: "which set of `classes` in Q, to allocate to rooms in J, given timeslots number T, so that we get a maximum utilisation?".

The free choice mode allows the solver to choose `classes` that better fits in available roomslots. In a set of available `classes`, which subsets provides the best utilisation is a characteristic question in this mode. As discussed in **chap.** 1, in order to study utilisation we allow the sets of *allocated* events to vary. The free choice mode directly allows the allocated events to vary and corresponds to the (extreme) case in which `classes` are allowed to be chosen solely to maximise utilisation. In the fixed choice mode, events are varied by solving problems with many different and varied sets of events.

5.3.3 Mathematical formulation : IP model

The following parameters and variables are used for modeling the problem as an Integer Program (IP). Roomslots, as defined previously, denote the available space to which `classes`/`groups` are allocated. So having r rooms and p timeslots, the number of roomslots would be rp.

Model TSA-SPLIT :

Given:

Q : set of all `modules`

q : total number of `modules`, $q = |Q|$

N : set of all `classes`

n : total number of `classes`, $n = |N|$

P : set of all timeslots

p : total number of timeslots, $p = |P|$

M : set of all roomslots

m : total number of roomslots, $m = |M|$ and $m = rp$

r : total number of rooms

In this chapter, we only consider tutorial and workshop spacetypes, and furthermore we consider them individually and do not mix, that is we will force that `classes` of a given type are assigned to room with the same spacetype. Spacetype mixing will be studied in **chap.** 7

Further parameters are:

S_i : number of students enrolled in `class` i

C_j : capacity of roomslot j

$\mathcal{L}_{N\times M}(L_{ij})$: location matrix between `classes` i and

roomslot j with entries L_{ij}.

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 occupancy allowed, i.e. minimum fraction of room seats to be filled

We also have parameters to impose non-negative upper limits on the penalties:

B_L^{up} : upper limits on the location (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 no location penalty, i.e. that all locations are perfect matches.

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

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

This variable is an integer variable that will ultimately govern the size of the group when assigned to a timeslot.

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

y_{ij} will is a binary variable that determines whether a group will be assigned to a timeslot or not.

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

Objective and Constraints : The objective is to maximise the overall seat-hours used

$$Obj^* = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} \right) \tag{5.1}$$

subject to the following constraints.

Given that no partial allocation is allowed, we enforce

$$\sum_{j=1}^{m} v_{ij} = S_i x_i \qquad \forall i \in N \tag{5.2}$$

Room capacities cannot be exceeded, and so we impose

$$v_{ij} \leq C_j y_{ij}, \quad \forall i \in N, j \in M \tag{5.3}$$

This also links the v and y decision variables.

Only one `group` can be allocated to a given roomslot:

$$\sum_{i=1}^{n} y_{ij} \leq 1, \quad \forall j \in M \tag{5.4}$$

The location penalty must be less than the upper limit B_L^{up}:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} L_{ij} y_{ij} \leq B_L^{up} \tag{5.5}$$

When $B_L^{up} = 0$, hard location penalty is enforced.

To limit the group size (GZ) penalty we impose

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \left| v_{ij} - G_i^t y_{ij} \right| \leq B_{GZ}^{up} \tag{5.6}$$

Note that only occupied roomslots will contribute to this penalty, and the net effect is to drive group sizes to the target size. When B_{GZ}^{up} is 0, all `groups` must have a size equal to G_i^t :

In some cases we also impose upper and lower limits on the group sizes, as university

managers would like group sizes in a given range. we therefore use the following :

$$v_{ij} \leq G_i^{up} y_{ij} \qquad \forall i \in N, j \in M \tag{5.7}$$

$$v_{ij} \geq G_i^{low} y_{ij} \qquad \forall i \in N, j \in M \tag{5.8}$$

We impose upper limits on the number of `groups` per `class`, e.g a `class` can be offered in only 2 groups :

$$\sum_{j=1}^{m} y_{ij} \leq G_i^{nb}, \quad \forall i \in N \tag{5.9}$$

Note that if $G_i^{nb} = 1$ the problem becomes the pure teaching space allocation, without splitting.

If a roomslot is used then the given fraction O^l of room seats needs to be filled:

$$v_{ij} \geq O^l C_j y_{ij}, \quad \forall i \in N, j \in M \tag{5.10}$$

In this chapter, we use $O^l = 0.3$ (we investigated other values and found that smaller values, or turning off this constraint altogether, do not change the results we present here).

The following constraint is entailed by the other constraints

$$\sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \leq rp \tag{5.11}$$

but we added it as it makes the formulation tighter, leading to a considerable reduction in computation times.

Finally, note that in the fixed choice mode, we simply enforce $\forall i.\ x_i = 1$ giving

$$\sum_{j=1}^{m} v_{ij} = S_i \qquad \forall i \tag{5.12}$$

The value of the objective Obj^* is then fixed, and the problem is simply that of feasibility. That is the problem will no longer be an optimisation problem, but rather overconstrained, in a bid to explore wether all the demand can be satisfied by available space.

5.4 Achievement curves

This methodology of generating random subsets, widespread in the literature, is detailed in the pseudocode of **proc: subset-scan**, and will be used throughout this chapter.

Given a fixed set of rooms and a set of `modules` Q of a given spacetype, and their respective `classes`, representing an estimate of the demand for students enrolled, BOS_e, one can ask the following:

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

The point of "Achievement Curves" is to study how the probability of the answer being "Yes, all are allocatable" varies with the demand. More specifically, since we are given a fixed set of rooms and a set of `classes`, then we can compute the utilisation if they were all to be allocated, we call this the "requested" utilisation U_R, (section 3.3.3). U_R will be generated, as explained in the coming pseudocode, by selecting random subsets of `classes` and summing over all available students, then dividing by the total available room capacities. That will provide a "requested" value that could be fed to the solver for

further analysis. We can then run a solver to find the set of `classes` that can be allocated, and these then give the "Achieved" utilisation U_A. If all `classes` can be allocated then $U_A = U_R$, however in general U_A is lower than U_R.

The intent of an achievement curve is to find the likely values of U_A as a function of U_R. We determine the achievement curves experimentally simply by taking many different sets of `classes`, over a wide range of values of U_R. In this chapter, we use the following simple procedure:

Proc: subset-scan

1 Given a set of `modules` Q of a given spacetype, with `module` $k \in \{1, \ldots, q\}$.
2 For ν= 1 to 3000
3 For s= 1 to q
4 Randomly select a subset of s different `modules` from set Q, represented by U_R
5 Run the solver in free choice mode on the subset to get U_A
6 END
7 END

In this **subset-scan** procedure, every randomly generated subset generates a point, with coordinates (U_R, U_A). An example is given in figure 5.1(a), in which we give the achievement curves with two different upper limits on the location penalty ($B_L^{up} = 800, 1200$).

Alternatively we can present the "fractional achievement", or "fill factor"

$$K_f = \frac{U_A}{U_R} \tag{5.13}$$

against U_R. An example is given in figure 5.1(b). The results were obtained using CPLEX 10.0 as the solver. CPLEX is given an "MIP gap" of 1% meaning it only stops when it

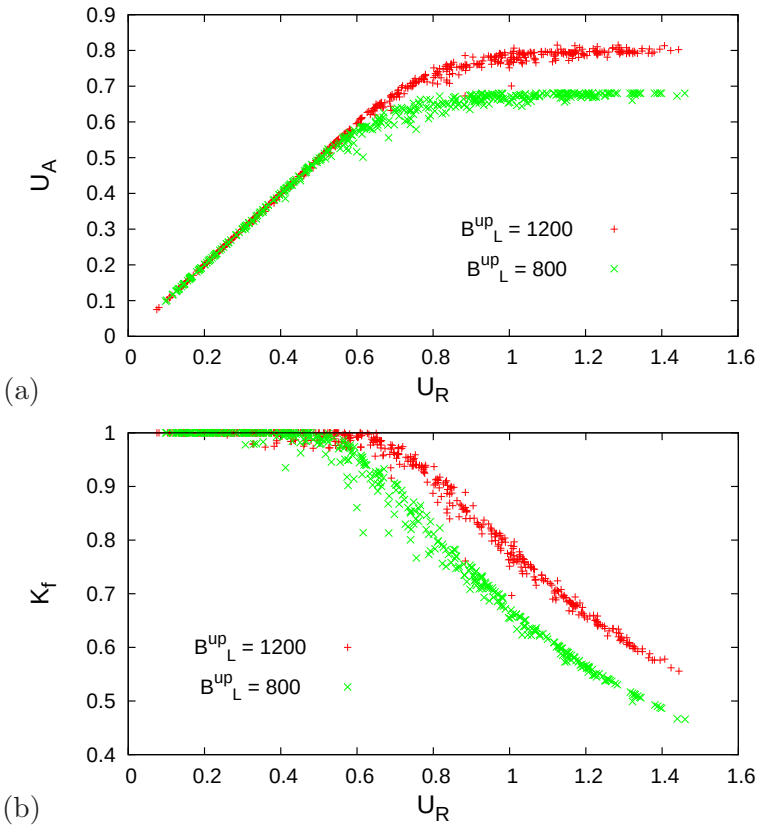

FIGURE 5.1: Safety graph, representing requested utilisation (U_R) versus achieved utilisation (U_A), with 2 different upper limits on the location penalty (for the WKSP data-set)

proves the current solution is within 1% of optimal. Hence we can be sure that the fall-off in achieved utilisation is a not a result of any deficiency in the solver.

As observed in figures 5.1(a) and (b), 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 them into three distinct areas:

1. "`Under`", $U_R < U_C\%$. Almost all instances are "satisfiable", that is, the requested utilisation almost surely be achieved. That is, $Pr(U_R = U_A) \sim 1$. In figure 5.1(b) it corresponds to the area where $K_f = U_A/U_R = 1$

2. "`Critical`", $U_R \approx U_C$. In this region there is mix of satisfiable and unsatisfiable instances.

3. "`Over`", $U_R > U_C$. In this region almost all instances are unsatisfiable, that is, it is impossible, or at least very unlikely that full allocation is possible.

We note that such sharp thresholds as seen above are common behaviour, in physical and combinatorial systems. In physics, they correspond to a "phase transition", for example, melting is a transformation from solid to liquid. Phase transition in complexity and combinatorial optimisation have attracted many researchers. For example, transitions have been identified in many NP-hard problems such as SAT, Hamiltonian cycles (HC) (Cheeseman *et al.*, 1991).

In terms of space management we might regard the "under" region $U_R < U_c$ as the **SAFE** region because the requests are very likely to be satisfied. The critical region and over are generally **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.

We are therefore interested in the quantity

$$Pr[K_f = 1] : \text{probability of achieving } K_f = 1 \text{ on random choice of } \texttt{classes} \text{ given } U_R \tag{5.14}$$

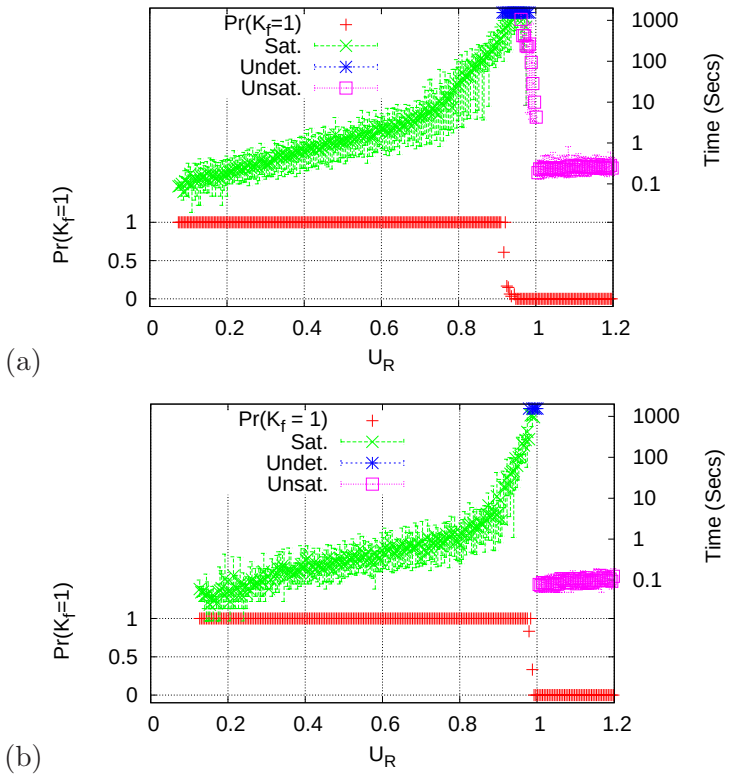

FIGURE 5.2: The x-axis is the requested utilisation U_R. The left-hand "y1-axis" is the probability of full allocation, Pr(K_f=1). The right-hand "y2-axis" is the run-time (on log scale). The average run-times are plotted together with "error bars" representing the lower and upper quartiles of the run-time. For the run-times, the instances are separated into: "Sat", satisfiable; "Unsat", unsatisfiable; and "Undet" for undetermined by the solver. 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.

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

5.5 Phase transition and computational hardness

In this section we give experimental results for the probability of full allocation, $Pr[K_f = 1]$ and also for the computational time needed for solving the allocation problems. In particular, we will see thresholds, indicating that the search costs (measured in run-time)

are much higher in the phase transition region, i.e. the existance of an `Easy-hard-easy` transition (Cheeseman *et al.*, 1991; Erben, 2001; Gent and Walsh, 1996; Ross *et al.*, 1996; Smith and Dyer, 1996).

Experimental Procedure. In order to explore "easy-hard-easy threshold" results we follow the following method:

1. Produce many varied subsets of the `classes` using a procedure similar to **subset-scan** in the previous section. For each such subset:
   (a) run the solver (CPLEX in this case) using model **TSA-SPLIT** and record whether the instance is
      i. "Sat.": a full allocation is found, i.e. the instance is satisfiable.
      ii. "Undet.": the problem reaches the time limit (we use 1500 seconds) and so remains undetermined.
      iii. "Unsat.": it is proved no allocation is possible.
2. Take the data and put into bins[3] according to the requested BOS for each instance; selecting bin-sizes appropriate to give a reasonably fine-grained plot, but with enough data points in each bin. Compute for each bin:
   (a) the probability of the instance being satisfiable.
   (b) for the runtime analysis, further divide the instances into "Sat.", "Undet.", and "Unsat." and for each division compute the average runtime, and the lower and upper quartiles (the 25th and 75th percentiles).

[3]A standard way to find the probability, is to group data points into "categories" or "bins" and then plot the frequency distribution (e.g probability) over all bins

The results of such an analysis are given in Figure 5.2 for the WKSP data set, and not imposing any limits on the location and group size penalties. The results for the probability of full allocation show a sharp threshold with $U_C \approx 92\%$. We also see that the runtimes are much larger in the neighbourhood of the threshold. Note that the run-time axis is logarithmic: instances at $U = 90\%$ take 10-100 times as long as those at $U = 80\%$. The observed `easy-hard-easy` behaviour is reasonable. Below the threshold 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.

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.

Figure 5.2 (b) gives the results of a similar investigation of the TUT dataset. The outcome is similar to the Workshop case of figure 5.2 (a), but we can notice that threshold is closer to 100%. We believe this is because the smaller rooms help achieve near optimal allocations compared to larger one in WKSP dataset. However both databases exhibit the same behaviour in terms of time requirements.

5.5.1 Thresholds with location constraints

In the same fashion as figure 5.2, in figure 5.3 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. The transition becomes

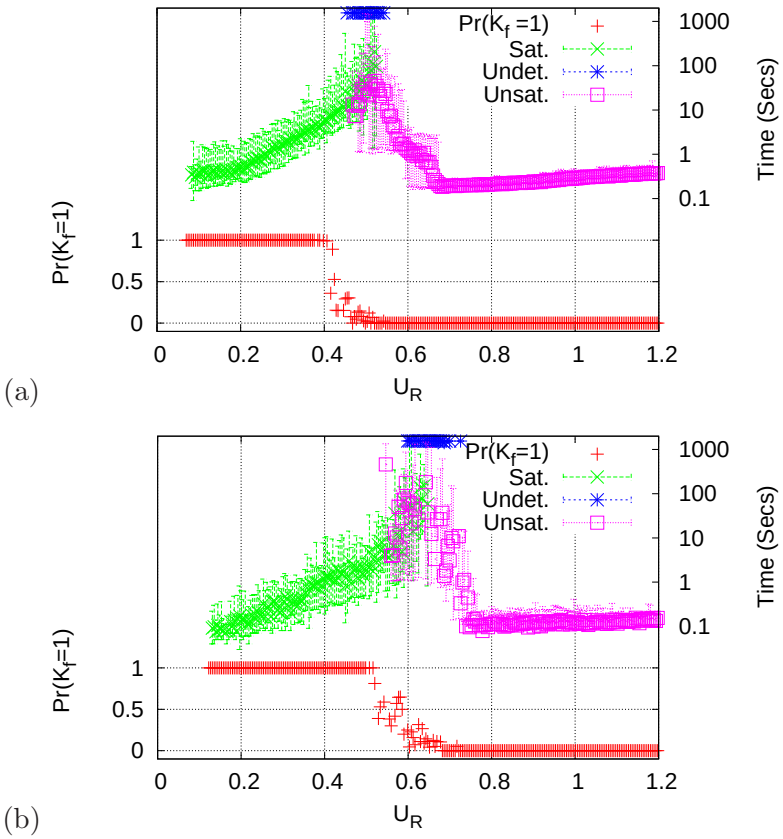

FIGURE 5.3: Threshold results, as Figure 5.2, but with an upper limit on location penalty $B_L^{up} = 800$. Data-sets: (a) WKSP (b) TUT

slightly less sharply defined. These are consistent with the results of **chap.** 3, that the need to reduce location penalties has the potential to significantly drive down utilisations. The computational hardness, and easy-hard-easy pattern, also peak at the threshold, in line with the case of pure splitting.

5.5.2 Thresholds with location and group size constraints

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.

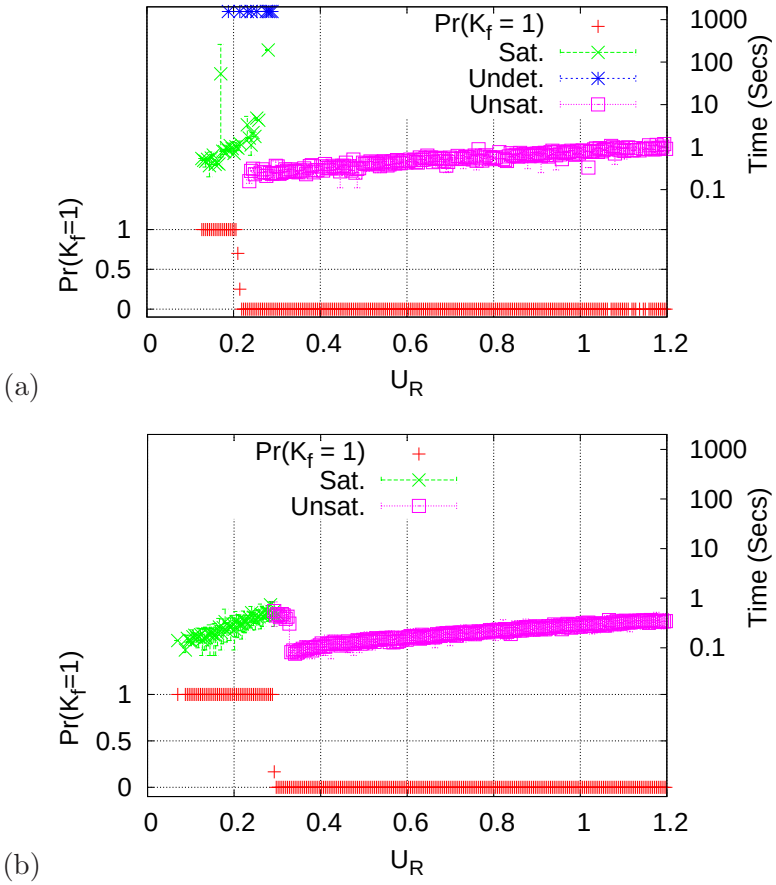

FIGURE 5.4: TUT Data-set $B_L^{up} = 800$ (a) Group size target enforced by $B_{GZ}^{up} = 150$ $G_i^t = 15$ (b) Group size strictly constrained in the range 12,...,16.

In figure 5.4(a) we take a target size of 15 and impose an upper limit on the resulting total `group` size penalty. There is again a threshold, but now the critical utilisation plummets to 20%. Also, the computational hardness remains in that many instances still cannot be solved.

As an alternative to the overall penalty, in Figure 5.4(b) 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. (remark that it is well-known that whilst thresholds are often associated with hardness peaks, it is that not all thresholds have a hardness peak).

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 room sizes profiles are part of our ongoing research, and will be reported elsewhere.

5.6 Hardness peaks using a stochastic (SA) solver

In this section, we reuse the solver of chapter 4, in a `Fixed-Choice` mode, using the `Simulated Annealing` algorithm with reheat, to study the *Threshold Behaviour* and *Phase Transition*. As shown later, using another algorithm strengthens our findings by suggesting that the same characteristic behaviour of the problems occurs across other different algorithms.

5.6.1 Runtimes comparison between (SA) solver and CPLEX

Figure 5.5 a) plots the run-times required by CPLEX and SA on the pure splitting case, NOT imposing any limits on the location or group size penalties. After 80% U_R, (SA) appear to exhibit a "False Threshold", hardly coping with the required change of regime, where no free roomslots are available, and given this set of roomslots, the goal becomes to maximize the total number of students allocated. While CPLEX adapts better, in the hard region, it still finds it difficult to fully allocate all courses. Specialised algorithms might be

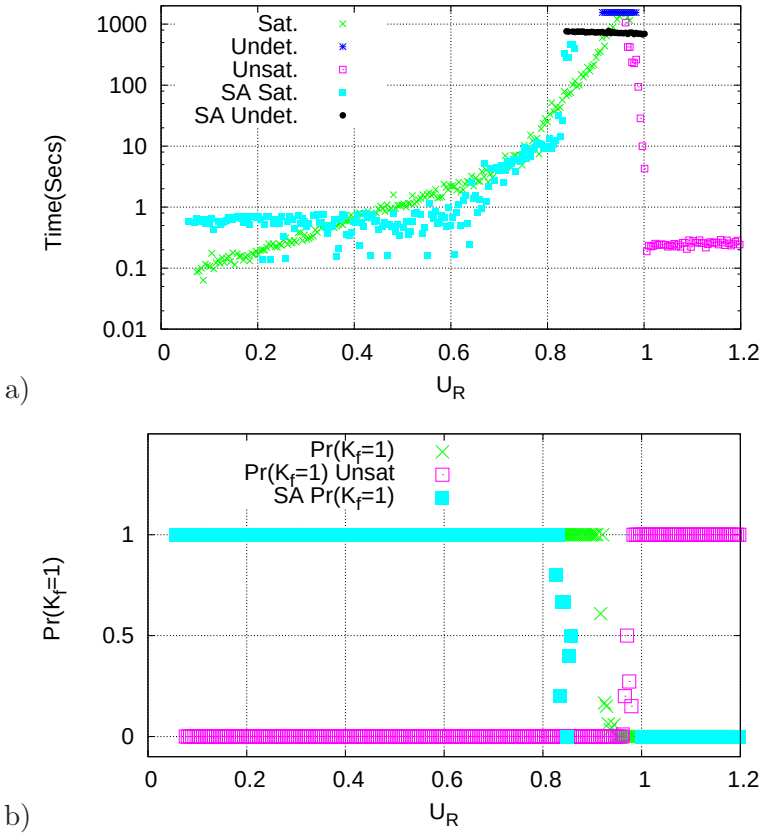

FIGURE 5.5: The average run-times, of SA and CPLEX plotted together. For the run-times, the instances are separated into: CPLEX: "Sat", satisfiable; "Unsat", unsatisfiable; "Undet" for undetermined, and Simulated Annealing (SA) : "SA Sat" satisfiable and "SA Undet" for undetermined.

required in this area of the search, however this study stands at the level of commonly used algorithms adaptable to most search types, e.g. CPLEX branch and bound and Simulated Annealing. Figure 5.5 b) represents the probabilities that $k_f = 1$ is achieved using CPLEX and the (SA) solver. Notice how the local search under-performes CPLEX on instances of the same size. The transition of $\mathbf{Pr}(K_f = 1)$ from 1 to 0 coincides with the respective peaks (for (SA) and CPLEX) in runtime shown in figure Figure 5.5 a). We also plot that $\mathbf{Pr}(K_f = 1)$ is Unsat. The small area in the graph for $\mathbf{Pr}(K_f = 1)$ between Sat. and Unsat represents the Undetermined (Undet.) region.

Figure 5.6 a) plots the runtimes required by CPLEX and SA in the presence of the location `L` penalty. The SA solver adapts well in the threshold area and even solves some instance quicker then CPLEX. Experiments with `SA` in presence of the location constraint, are performed with solver in an optimisation mode, the outcome is therefore either Sat. or Undet.

Figure 5.6 b) represents the probabilities that $k_f = 1$ is achieved using Cplex and (SA) solver in the presence of the location penalty. Results are very similar to Figure 5.5 b) but the transitions smoother and less sharp then the pure splitting case.

5.7 Summary

Still in the "Space Management" Perspective, the splitting problem , as mentioned in chapter 1, is of considerable importance for many institutions. In this chapter we looked at the decision problem of: "Is it possible to allocate all classes into the available space and achieve a target utilisation U_R?". We found that, even if some utilisation levels can be achieved

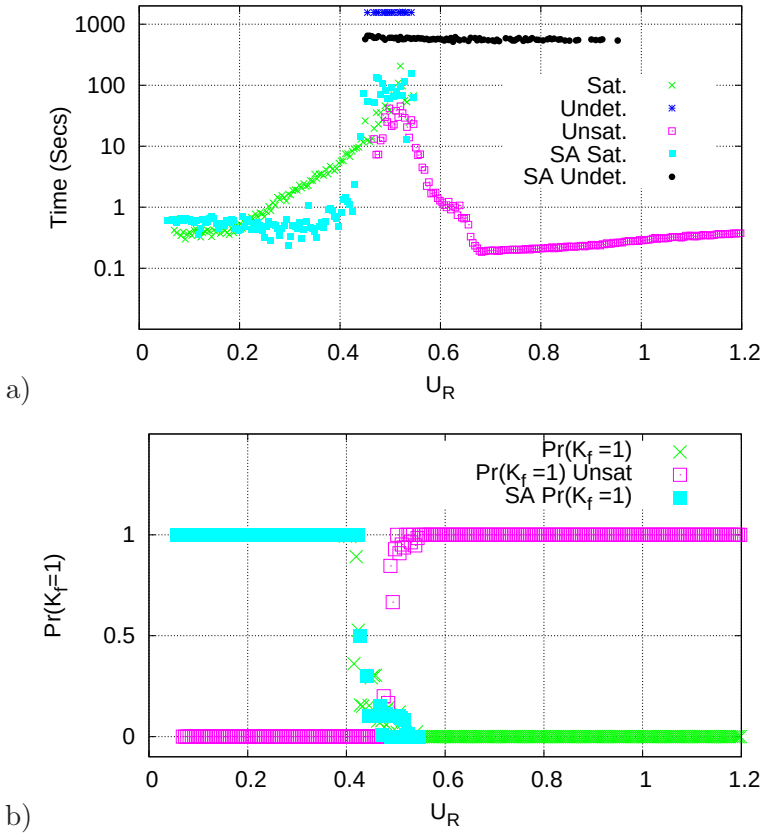

FIGURE 5.6: The average run-times, of SA and CPLEX in the presence of upper limit on the location penalty $B_L^{up} = 800$, plotted together. For the SA solver run-times, the instances are separated into: "Sat", satisfiable; and "Undet" for undetermined by the solver

with a good splitting, they won't be of much use if solvers are incapable of achieving them. We have shown that the computational effort of achieving some desired utilisation levels can be very high, and that happens naturally at the *Threshold*. This chapter, proposes a case in which, *Phase Transition* systems can have a practical impact on studying space usage. We used an integer programming formulation and a Simulated Annealing Meta-heuristic to illustrate the point.

So far in the thesis, we have only considered spacetypes separately. We have enforced that the type of a class must be the same as that of the room. However realistically allocation happens in a mixed spacetype environment where e.g. tutorial groups can be assigned to a lecture theatre etc. We will discuss spacetype mixing in chapter 7. On the other hand, In the proposed model of this chapter, we have omitted the timetabling penalty, and considered that conflict inheritance (**chap.** 6) is zero. In the coming chapter we briefly illustrate how good is our approximation of a zero-valued conflict inheritance by considering a model that incorporates student enrolments.

Chapter 6

Partial Inheritance in Teaching Space Allocation

The finest inheritance you can give to a child is to allow it to make its own way

Isadora Duncan

6.1 Introduction

In chapter 5, we studied the splitting problem not considering constraints arising from timetabling conflicts. When splitting and allocating a class "A" which conflicts with "B" and "C", we define the "partial inheritance" as a measure of how conflicts are transmitted from "parent" classes to "child" groups as splitting takes place. It has always been known that sections generally relax the timetabling constraints (Schaerf, 1999; Carter, 2000). Indeed, part of this work is to quantify and discuss the extent to which the timetabling conflicts are reduced and provide cases where they sometimes get reduced to zero. This chapter actually

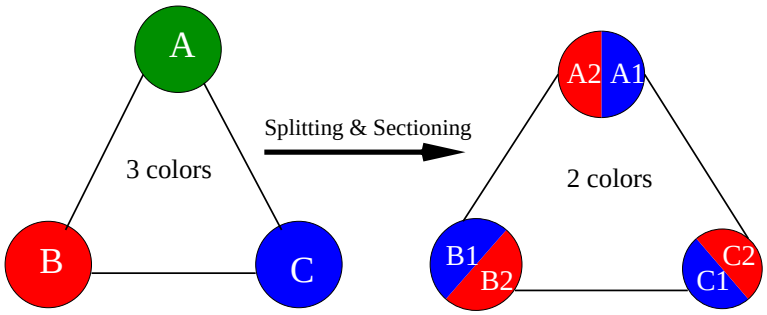

FIGURE 6.1: Effects of splitting and sectioning which reduces the timetabling conflicts

differs from existing work in the area in that it's `student-centered`, that is decision are made on assigning individual students to groups as well as constructing a class timetable.

6.2 Motivating partial inheritance

So far, the models in this thesis, have focused on splitting and allocating `classes` to available roomslots, disregarding how individual students are assigned to respective groups. We took a `course-centred` approach (**Ref. chap.** 2). In chapter 3, when assigning teaching events to roomslots, we have studied the timetabling penalty as a major real-world constraint affecting space utilisation. This penalty took into account the conflicts between `classes`, as students had to enrol in a multitude of them in a given semester. It was represented by a conflict graph. For example, three classes "A", "B", and "C" in which at least one or more students are simultaneously enrolled, cannot be allocated to the same timeslot, necessarily requiring three timeslots[1]. When those classes get split into groups e.g. A^1, A^2, B^1, B^2 and C^1, C^2, then if every group should inherit all the conflicts from its parent, then they will require three or more timeslots. However, there are cases where splitting and

[1] in figure 6.1, an edge between any two vertices, will represent two conflicting classes and colors represent timeslots

reallocating students to groups could eventually reduce the number of conflicts (two colors in figure 6.1). Consequently, even if conflicts cannot be completely reduced, their impact on the timetabling penalty would be negligible post-splitting. The main assumption here, is that grouping students, in a proper manner, would relax the timetabling penalty.

For us to support this assumption, we are bound to use a `student-centred` model of the teaching space allocation, capable of assigning individual students to groups and timeslots, under well defined conditions. Moreover, a valid student model, requires information on the enrolment of students into modules and classes, that is a provision of a real-world conflict graph, generated by the student enrolments. This provides our motivation: (1) to design, a new enrolment generator, that would capture as much as possible, the factors that mimic those enrolments, and (2) to support the claim that when splitting occurs, a *course-centred* approach is a credible and valid approximation that allows timetabling conflicts to be relaxed.

Note, we do not try to provide an exhaustive study of conflicts and conflict graphs in timetabling (as with a graph theoretical approach), that stands beyond the scope of this thesis.

6.3 Enrolment Generator: genRol

This section describes the enrolment generator (***genRol***) that will be used in this chapter, to generate student enrolments in classes and their respective conflict matrices. Although, ***genRol*** is fully randomised, class sizes has been determined by class profiles available in datasets of Appendix. A: that is, given a set of real-world, fixed class sizes, ***genRol*** selects

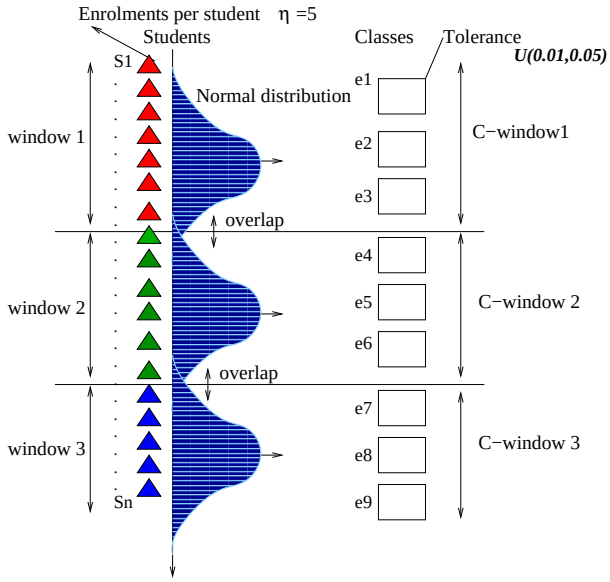

FIGURE 6.2: Schematic of the workings of ***genRol***, showing the window based structure, and a random selection based on a Normal distribution

students and fills in classes such that the total number of students assigned to a given class is less then or equal to the fixed class size multiplied by the class tolerance γ[2].

Procedure **proc: genRoll**, describes the algorithm behind ***genRol***. Using the set of class profiles and the set of all students, we group students and classes into windows and assigns students to classes by sampling from a uniform or normal distribution.

[2] γ allows a small variation in the class size, as generally classes have different sizes from year to year.

6.3.1 Window based selection

Proc: genRol

```
1  Initialize sorted student vector B.
2  Initialize sorted class vector Q.
3  Initialize class sizes φ_i.
4  Set η upper limit on student-enrolments.
5  Set γ: Course tolerance.
6  Set σ: Window size
7  Set ϵ: window overlap
8  Set δ: number of windows
9  Set ζ_b: current student-enrolment for student b
10 Set φ_i: current class capacity for class i
11 initialise students window center-position W_C[w]
12 select probability distribution Dist = (N|U)
13 LOOP w = 1 to δ
14     LOOP k over all φ_i in window w
15         WHILE φ_k ≤ φ_k × γ DO
16             IF Dist == N
17                 set Dist = N(W_C[w], σ + ϵ)
18                 select student b according to Dist
19                 IF ζ_b ≤ η THEN
20                     add student to class φ_k
21                     ζ_b = ζ_b + 1
22                 ELSE
23                     remove student s from window (student-enrolments for s satisfied)
24                 END IF
25             ELSE IF Dist == U
26                 set Dist = U(W_C[w] + (σ + ϵ)/2, W_C[w] − (σ + ϵ)/2)
27                 select student b according to Dist
28                 IF ζ_b ≤ η THEN
29                     add student to class φ_k
30                     ζ_b = ζ_b + 1
31                 ELSE
32                     remove student s from window (student-enrolments for s satisfied)
33                 END IF
34             φ_k = φ_k + 1
35         END WHILE
36     END LOOP
37 END LOOP
38 Return set of student enrolments, conflict graphs and new class profiles
```

The schematic diagram of figure 6.2 illustrates our approach to ***genRol***: Students and classes are used as two separate, sorted vectors. Students s_1 to s_n can be thought as occupying aligned positions 1 to n. Windows which can be thought of as departments in a given faculty, or faculties in a large university, are groupings of students or `classes`, where every window is a subset of the `classes` or students respectively. E.g. students of `window 1` are to be randomly selected and assigned to `classes` of the corresponding class window `C-window 1`. Although student windows can only have students assigned to the corresponding `classes` window, when overlap is allowed, a small fraction of students of a given window can be assigned to `classes` of another window.

Initially, all `classes` are empty with capacity dictated by the profile of `classes` (**See** Appendix. A). In every window, students are selected, at random, from a given distribution (*Normal* or *Uniform*) and assigned to `classes`.

In proc: genRol, the *student-enrolment* number (η) is the upper limit on the number of classes a student enrols in, per semester. Generally, the higher the limit, the denser the conflict graph. The class tolerance parameter γ controls the number of students assigned to any given `class` as a fraction of the given `class` capacity, This, in turn, controls the overall requested `BOS`. It is set as a random parameter controlling the increase or decrease in the total number of students in classes from year to year. Also, σ is the size of the window and δ is the number windows. So, if b is the total number of students, $\sigma = b/\delta$.

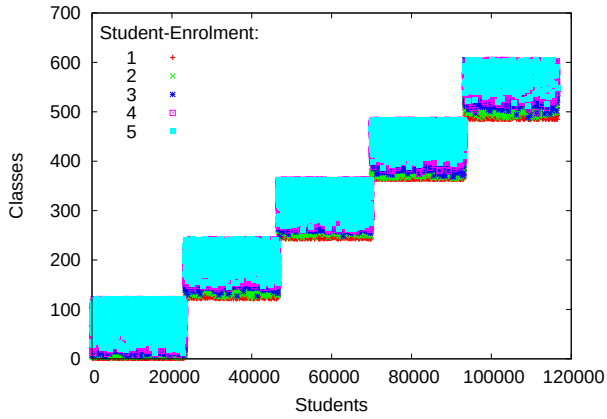

FIGURE 6.3: Selection based on a uniform distribution with with $\delta = 5$, $\eta = 5$, $\epsilon = 0.05$

6.3.2 Enrolment histograms

In the following histograms, we present the outcome of enrolments produced by *genRol*. The `X` axis represent the students, and `Y` axis represent the `classes`. One single student can enrol in more then one class, and therefore every *student-enrolment* is represented by a different color.

Figure 6.3 shows the enrolment diagram using a selection based on a `Uniform` distribution. We allow 5% of overlap between windows, that is $\epsilon = 0.05$, allowing ***genRol*** to select student-enrolments and fill `classes` from other windows.

In figure 6.4, student selection is based on a normal distribution with a standard deviation larger the window size, allowing more students to cross-enrol in a neighbouring window. Similarly, figure 6.5 uses a normal distribution, but with a tighter standard deviation ($std = \sigma$). the histogram is less fuzzy than figure 6.4 and less students are allowed to register across windows.

Our approach to ***genRol***, is to try to mimic real world enrolments where students

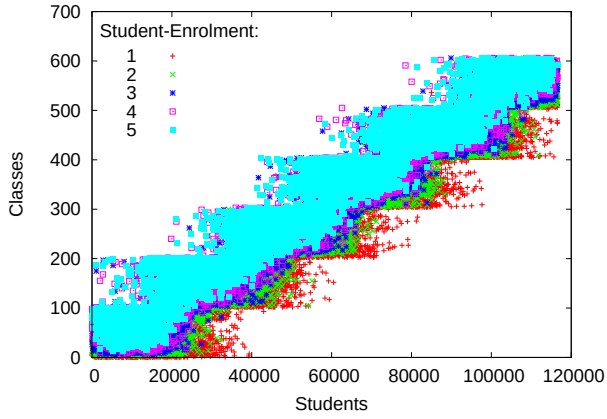

FIGURE 6.4: Selection based on a normal distribution with with $\delta = 6$, $\eta = 5$, standard deviation std $= 1.8 \times \sigma$ and $\epsilon = 0.01$

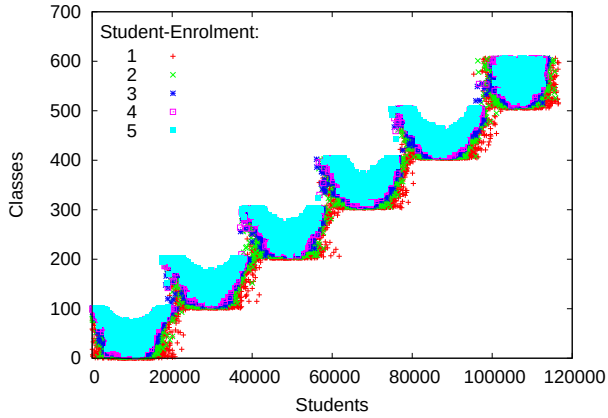

FIGURE 6.5: Selection based on a normal distribution with with $\delta = 6$, $\eta = 5$, standard deviation $= \sigma$ and $\epsilon = 0.01$

in large universities can choose to subscribe to courses from other departments (windows). This enrolment will generally affect the structure of the timetabling conflict graph. To this end, on a separate research track, studying the clustering coefficients in conflict graphs, and analysing real-world student enrolments[3], we found a very close match between our histograms and histograms revealing the clustering of students into classes. This actually strengthens our belief that a departmental/window grouping is a reasonable strategy for ***genRol***.

6.4 Student-based model formulation (EN-st)

In the following formulation every student is able to enroll in more than one `class`. We will call the single enrolment of a student in a specific `class`, a *student-enrolment*, the number of *student-enrolments* for a given student is bounded by η, which is the upper limit on the number of classes a student is allowed to register in. On the other hand, roomslots denote all available space to which `classes` are allocated. Having r rooms and p timeslots per room, the number of roomslots is rp.

Model EN-st :

given :

B : the set of all students

N : set of all *student-enrolments*,

n : total number of *student-enrolments*, $n = |N|$

M : set of all roomslots

[3]From the university of Toronto

m : number of roomslots, m=$|M|$ and $m = rp$

K : the set of all `classes`

P : the set of all timeslots

p : the number of timeslots, $p = |P|$

$T^z = \{j \quad | j \in M : z \in P\}$: represents the set of roomslots corresponding to a given timeslot z, with $M = \bigcup_{z \in P} T^z$ and $\bigcap_{z \in P} T^z = \emptyset$.

T^z is the set of roomslots considered at a given one time.

Given a partition of students in a given `class`, we define the following :

$S^i = \{s \quad | s \in N : i \in K\}$: represent group of students which belong to different `classes`, with $N = \bigcup_{i \in K} S^i$ and $\bigcap_{i \in K} S^i = \emptyset$. s_i :number of students enrolled in class i, $s_i = |S^i|$

$C^\beta = \{d \quad | d \in N : \beta \in B\}$ represents the set of conflicts such that every subset represents "student-enrolments" for a single student, with $N = \bigcup_{\beta \in B} C^\beta$ and $\bigcap_{\beta \in B} C^\beta = \emptyset$.

For every student β, C^β represents all *student-enrolments* for that student.

C_j : capacity of roomslot j

W_i: weight of `class` i, represents a weighted preference for a given `class`.

L_{ij}: location matrix between `classes` i and room 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 `group`s of `class` i

O^l : minimum occupancy allowed, i.e. minimum fraction of room seats to be filled

τ: Upper limit on section size for all `classes`, $G_i^{up} \leq \tau \quad \forall i$

ν: Upper limit on section number for all `classes`, $G_i^{nb} \leq \nu \quad \forall i$

μ: Upper limit on allowed sections taught by a single teacher

B_L^{up} : upper limits on the location (L)

B_{GN}^{up} : upper limit on `group` number (GN) penalty

Decision variables: We use the following binary decision variables:

$$x_{bj} = \begin{cases} 1 & \text{if } \textit{student-enrolment } b \text{ is assigned to roomslot } j \\ 0 & \text{Otherwise} \end{cases}$$

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

Objective: to maximise the total `BOS` (total *student-enrolments*):

$$Obj^* = max(\sum_{b=1}^{|N|}\sum_{j=1}^{|M|} x_{bj}) \tag{6.1}$$

Subject to the following constraints:

The location penalty must be less than the upper limit B_L^{up}:

$$\sum_{i=1}^{|K|}\sum_{j=1}^{|M|} L_{ij}y_{ij} \leq B_L^{up} \tag{6.2}$$

When $B_L^{up} = 0$, hard location penalty is enforced.

The group number penalty could also be preferred to be less than an upper limit B_{GN}^{up}, set by university policies:

$$\sum_{i=1}^{|K|}\sum_{j=1}^{|M|} y_{ij} \leq B_{GN}^{up} \tag{6.3}$$

B_{GN}^{up}, will represent an upper bound on the total group number required.

6.4.1 Student constraints

This set of constraints enforces proper assignment of students to groups and roomslots. In constraint(6.4) a given *student-enrolment* cannot go in more than one roomslot:

$$\sum_{j=1}^{|M|} x_{bj} \leq 1, \quad \forall b \in N; \tag{6.4}$$

In constraints (6.5) (6.6), the number of all students assigned to a roomslot should be less than the roomslot capacity C_j:

$$\sum_{b=1}^{|N|} x_{bj} \leq C_j, \quad \forall j \in M; \tag{6.5}$$

and with τ is the upper limit on the group size, then we also have:

$$\sum_{b=1}^{|N|} x_{bj} \leq \tau, \quad \forall j \in M; \tag{6.6}$$

In constraint (6.7) the sum of all students of a `class` is less than the size of the given `class` s_i.

$$\sum_{b=1}^{|S^i|} x_{bj} \leq s_i, \quad \forall i \in K, \forall j \in M; \tag{6.7}$$

Group sizes should not exceed the initial roomslot capacity. This constraint links also variables x to y.

$$\sum_{b=1}^{|S^i|} x_{bj} \leq C_j y_{ij}, \quad \forall i \in K, \forall j \in M; \tag{6.8}$$

If a roomslot is used then the given fraction O^l of room seats needs to be filled:

$$\sum_{b=1}^{|S^i|} x_{bj} \geq O^l s_i y_{ij}, \quad \forall i \in K, \forall j \in M; \tag{6.9}$$

Classes/Group constraints

Constraints (6.10) (6.11) specify upper limits on the number of groups. Constraint (6.10) ensures that the number of groups per class be less then ν and constraint (6.11) does not allow assigning more then one group to any roomslot.

$$\sum_{j=1}^{|M|} y_{ij} \leq \nu, \quad \forall i \in K, \tag{6.10}$$

$$\sum_{i=1}^{|K|} y_{ij} \leq 1, \quad \forall j \in M, \tag{6.11}$$

Timetabling constraints

Constraint (6.12) represent student conflicts, such that any two `classes`, where a given student is simultaneously enrolled, cannot be allocated to the same timeslot.

$$x_{b_1 j_1} + x_{b_2 j_2} \leq 1, \quad \forall \gamma \in P, \forall j_1 \in T^{\gamma}, \forall j_2 \in T^{\gamma}, \tag{6.12}$$

$$\forall \beta \in B, \forall b_1 \in C^{\beta}, \forall b_2 \in C^{\beta} \tag{6.13}$$

$$b_1 \neq b_2, j_1 \neq j_2; \tag{6.14}$$

the C^{β} set, hold the *student-enrolments* for student β

Constraint (6.15) ensures that no two groups of a given `class` be allocated to the same timeslot (assuming for example that a given teacher teaches both groups).

$$\sum_{j_1}^{|T^p|} y_{ij_1} \leq 1, \quad \forall p \in P, \forall i \in N; \tag{6.15}$$

$$x_{bj}, y_{ij} \in \{0, 1\} \quad \forall b \in N, \forall i \in K, \forall j \in M; \tag{6.16}$$

6.5 Decision making and effects on Partial Inheritance

We have discussed, so far, a practical way to generate student enrolments and class conflicts using **genRol**. In the following sections, the focus is on the student-centered model (**EN-st**)

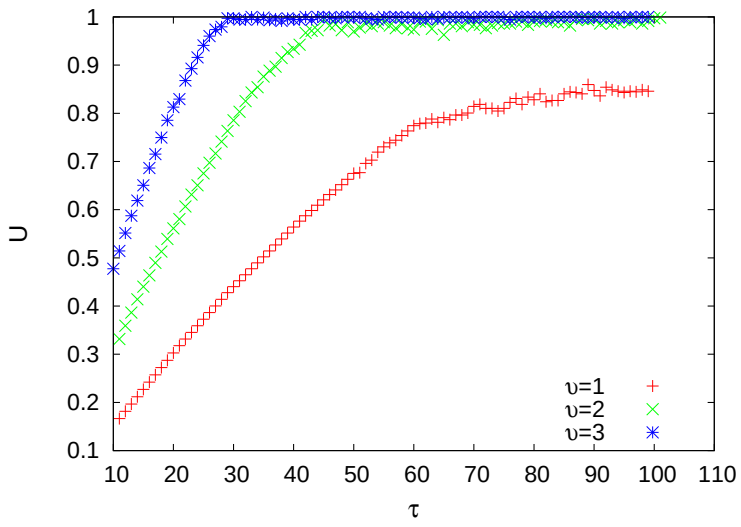

FIGURE 6.6: Utilisation (U) versus upper limit on group size (τ) for different upper limit on the group number (ν)

of **sec.** 6.4, whose objectives and constraints are similar to the model of **chap.** 5, with the exception that decisions here are made based on individual student assignments. Indeed, we give evidence of conflicts being resolved, when splitting occurs under certain conditions and parameter settings. We show that achieving or not full allocation given a set of class conflicts is dependant on a set of parameters of which, the allowed number of groups per `class` (ν) and the group size (τ).

Partial inheritance $\mathcal{I}$: Let us define the *partial inheritance* to be a function $\mathcal{I}$, depending on the following variables: window number (δ), enrolments per student (η), timeslots number (p), upper limit on group number (ν), upper limit on group size (τ), and denoted by $\mathcal{I}(\delta, \eta, p, \nu, \tau)$[4].

[4]The dependence of $\mathcal{I}$ on those variables, will be supported by coming experiments on utilisation and not by theoretical analysis. For detailed studies of conflict resolution in graph theory refer to **chap.** 2.

$$\mathcal{I}(\delta,\eta,p,\nu,\tau)\begin{cases} = 0 & \text{if all conflicts are resolved, } U = 100\% \\ \neq 0 & \text{Otherwise} \end{cases}$$

Note that the approach has been centred around deciding on conflict resolution based on the effects on the utilisation rather then on countable conflicts, yet one can argue that the latter is more expressive, however we have opted for this approach in line with work on utilisation.

We have no interest at this stage in knowing what values such a function would take. However, we are interested in knowing under what circumstances (e.g. parameters), $\mathcal{I}$ is not null.

Our approach is as follows: starting from: a fixed instance of `classes` and rooms, a given student enrolment, a fixed number of windows, and a fixed timeslot number, we vary the upper limit on the *group size* τ and plot the utlisation versus τ for different values of upper limit on group number ν. We vary τ such that it scans different group sizes up to sizes larger then the maximum class size: $\tau \in \{0,\ldots,C^{max}\}, \quad C^{max} = max(C_j), \quad \forall j.$

Results from this experiment are presented in figure 6.6. When ν is one, there is no full allocation for any values of τ, even as τ becomes equal to C^{max}. Hence conflicts will remain unresolved and $\mathcal{I}(\delta,\eta,p,1,\tau) \neq 0$. As ν becomes larger than one, two for example, after a given value of τ (≈ 42) full allocation becomes possible $\mathcal{I}(\delta,\eta,p,2,42) = 0$. Same applies when ν is set set to 3, $\mathcal{I}(\delta,\eta,p,3,30)$=0, Clearly for $\nu = 3$, all values of $\tau \geq 30$ make $\mathcal{I} = 0$.

These experiments gave almost the same results when we changed the window number (δ) and the number of enrolments per student (η), in this case with $\delta \in \{3 \dots 8\}$ and $\eta \in \{3 \dots 7\}$. Changing values of δ and η caused values τ to be shifted for $\mathcal{I}$ to become equal to zero. Administrators that have to make decisions on splitting, are interested in the first values of τ and ν which make $\mathcal{I} = 0$ or $U = 100\%$. Then, one might consider the smallest pair (τ, ν) (e.g. smallest τ and ν) which makes the `partial inheritance` zero, mainly because it's sometimes preferred to have group sizes as small as possible, or possibly the smallest number of groups.

As figure 6.6 dealt with one instance of `classes` and enrolments, in the coming section we study the behaviour of the partial inheritance under changing student enrolments. In a sense we study the robustness of `partial inheritance` at the pair (τ, ν), in the case of figure 6.6, at the points $(30, 3)$ or $(42, 2)$.

Note, the obvious cases where *partial inheritance* can never be zero: for example, if the number of *students-enrolments* is larger than the number of timeslots *(which is highly unlikely, as in a given teaching week with 45 timeslots, students are not usually allowed to enrol in more then 4-7 classes).* Also when the number of windows is greater than the number of courses or when there are less than 2-3 available rooms, or fewer than 4-5 timeslots, $\mathcal{I} \neq 0$. Those cases rarely happen in practice.

6.6 Robustness and solution quality

6.6.1 Free versus Fixed timetable

The student model used in this chapter, (model **EN-st**), involved decisions on which classes are assigned to which roomslots (**y** variables), and also which students are assigned to which groups (**x** variables). Hence, there is the *timetabling* component and the *sectioning* component repectively. Studying the robustness of the *Partial Inheritance*, leads to understanding the behaviour of utilisation levels at the *special* pair (τ, ν), for a fixed timetable[5], under changing enrolment. For this purpose, in this section, we will make use of both procedures, ***genRol*** and ***Enroll-Robust***.

Proc: Enroll-Robust
1 Given a set of `classes` K of a given spacetype.
2 Use model **EN-st** on single instance of classes and rooms for a given pair (τ, ν)
3 Extract the timetable (y variables): y_{ij}^{FIX}
4 For z= 1 to 3000
5 Run **genRol** and get U_R
6 Run modified model **EN-st** with $y_{ij} = y_{ij}^{FIX}$
7 Get U_A
8 END
9 END

6.6.2 Proc: Enroll-Robust

Procedure ***Enroll-Robust*** explains the methodology used. For a fixed timetable and under varying enrolments and class tolerance γ, the main purpose is to study the utilisation level at the given "special" pair (τ, ν), e.g at $(30, 3)$ or $(42, 2)$. For example, at the pair $(30, 3)$

[5] fixed values of y_{ij} variables become y_{ij}^{FIX}

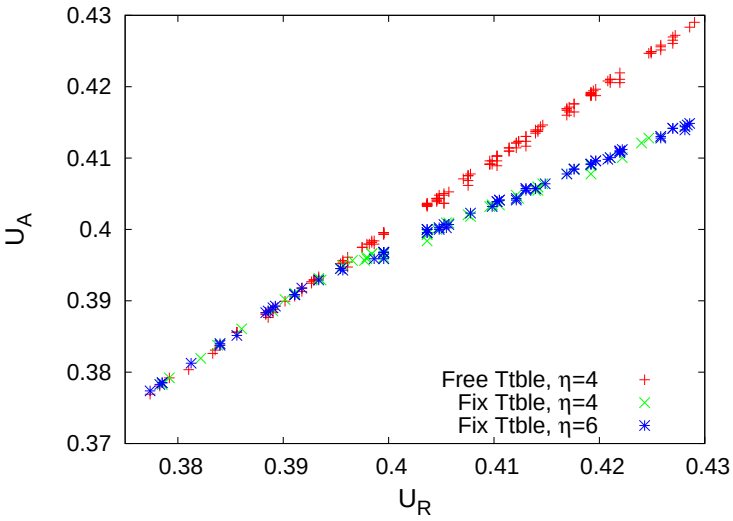

FIGURE 6.7: Robustness of the Partial Inheritance at the special pair (τ, ν). $\tau = 42$ and $\nu = 2$. Plots with different values of number of enrolments per student (η=4 and 6), for free and fixed timetable.

of figure 6.6, which corresponds to a single solution provided by model (**EN-st**) for fixed values of τ and ν, we extract the timetable assignments (**y** variables). Then for every single run, using procedure **genRol**, we vary the enrolments and the class tolerance γ, ($\gamma \sim \mathcal{U}(0.02.0.05)$). We record the requested utilisation U_R versus the achieved U_A. We repeat this procedure and collect the points shown in figure 6.7. Note, this procedure is different than ***proc:subset-scan*** used in chapter **chap.** 5, as ***Enroll-Robust*** does not select different subsets, but rather changes the enrolments and the class sizes, to generate requested utilisations U_R.

In figure 6.7, Every point in the graph represents a single solution. The first plot, leaving the timetable free, shows that almost every requested utilisation has been achieved. Changing the timetable for every instance helps resolve all the conflicts. Besides, by using a reasonably higher value for τ, allows enough flexibility for the solver to make $\mathcal{I} = 0$.

For the other plots (`Fix Ttble`), we fix the timetable and find that after a given critical utilisation level, not all requests can be achieved. However, the difference is quite small, for example, while a requested value U_R of 0.39 can be easily achieved, a U_R of 0.43 can only lead to 0.41 achieved. One can generally deduce that, depending on how risk averse universities are, a timetable fixed from the outset, is robust enough that differences between requested and achieved utilisations do not exceed the 5% mark. This is also supported in figure 6.8, where we plot the requested utilisation, governed by a change in enrolment and class tolerance, versus the fill factor K_f[6], for a fixed timetable. One can notice that the change induced by the requested utilisation do not drive the fill factor by more than 5% (from 0.95 to 1) and for U_R between 0.38 and 0.4 the fill factor is always 1.

6.6.3 Research directions

This section has lead us again to the *timetabling paradox* that was discussed in **chap.** 2. That is, should we timetable post enrolment or pre-enrolment. Understanding space planning and the effects of this concepts on space utilisation is beyond the scope of the thesis and remains, up to our knowledge an open-ended question. Many approaches, tend to provide an initial timetable, then after enrolments occur, the initial timetable is altered to meet the new demands. Yet decisions as to the group sizes and group numbers would greatly affect this alteration and more complex constraints tend to complicate changing a timetable post-enrolment. Figure 6.9 represents the stochastic approach of a global decision making process, where in a first stage a timetable is constructed and decision on upper limits of group size and group number are made, then as time unfolds and enrolment occurs, the

[6]The "fractional achievement", or "fill factor" was defined in **sec.** 5.4 of **chap.** 5, as $K_f = \frac{U_A}{U_R}$.

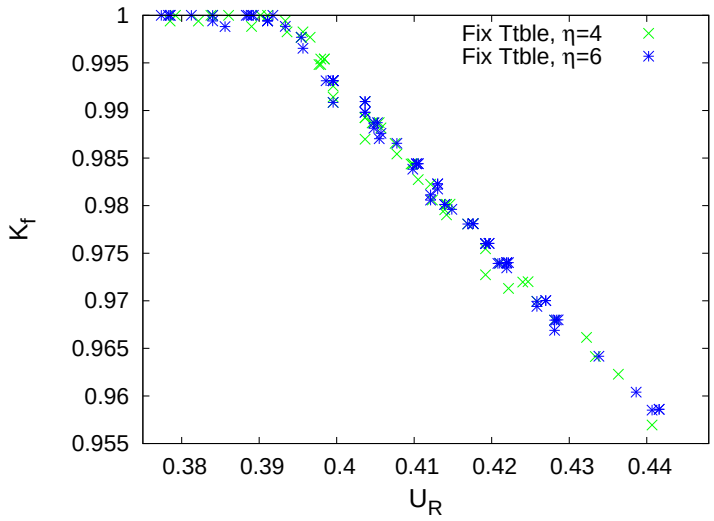

FIGURE 6.8: Achievement curve, representing K_f versus U_R, For different values of (η=4 and 6), at the special pair (τ, ν). $\tau = 42$ and $\nu = 2$.

second stage or "recourse" will decide on the student assignments.

6.7 Summary

This chapter has studied and introduced the *partial inheritance* concept which was defined as to how conflicts are transmitted from classes to group splits when splitting and allocation

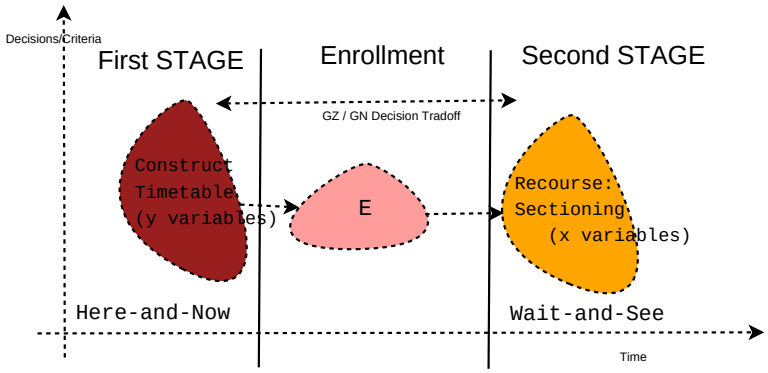

FIGURE 6.9: Stochastic approach

occurs. We have designed a purpose-fit student enrolment generator **genRol** that tries to mimic real world enrolments and generates realistic class sizes. Based on that generator we showed cases where most of the conflicts got resolved as more group splits were allowed. We have also studied the robustness of the ***partial inheritance***, and showed that changing enrolments could cause a 5% drop in the requested utilisation, under a fixed timetable.

It is well known that sectioning will relax the timetabling conflicts (Schaerf, 1999; Carter, 2000), but this chapter was needed to support this claim in a splitting framework, which is a close variant of the well known *Sectioning* problem. Moreover, as we will deal with spacetypes in the coming chapter, when a given spacetype requires splitting the timetabling penalty is omitted but when no splitting is needed (e.g Lectures) class conflicts are considered and the penalty is switched on automatically.

Chapter 7

Spacetype Mixing in Teaching space allocation : Long Term Planning

If you are planning for a year, sow rice;

if you are planning for a decade, plant trees;

if you are planning for a lifetime, educate people

Chinese proverb

7.1 Introduction

So far in this thesis we have considered spacetypes separately; e.g. the allocation of tutorial activities to tutorial rooms, lectures to lecture theatres etc. However, in practice, allocation happens in a mixed spacetype environment in which, teaching events of a given type can be allocated to rooms of another spacetype, e.g. tutorials can borrow lecture theatres or workshop rooms, when they're available. The "preference" for which events can be

placed in which spacetype is of great practical interest for the academic sector in general and RealTime Solutions in particular. This chapter will provide a more general model for the teaching space allocation problem with spacetype mixing and look at altering the spacetypes of rooms so that the critical utilisation can be improved. We address the problem of finding what better "mix" of spacetypes within the physical space is achievable, and what minimal change in space profiles would best improve the critical utilisation?. We provide two methodologies: a heuristic and an exact approach to alter the room spacetypes.

7.2 Aims and motivation

As discussed in chapter 1, the limited availability of teaching space, constantly motivates institutions to achieve optimal space usage. Our study of teaching space, has focused so far on the short-term space management perspective, " how to split and allocate given demand to the available teaching space". Teaching space is constituted of (not restricted to) rooms of different sizes, providing a range of facilities, e.g. large tiered rooms also known as lecture theatres, small tutorial rooms, laboratory rooms, seminar rooms, etc.

Rooms are therefore associated with a given spacetype (e.g. lecture, tutorial, workshop, seminar etc.) and generally host academic activities of a similar type, for example, lectures are offered in lecture theatres, tutorials in tutorial rooms, etc.

Room characteristics (sizes and spacetypes) representing a set of given rooms, will form the space profile of the available space. Therefore altering one room's spacetype, will affect the "space profile".

Space planning in academic institutions is closely related to maintaining a given

space profile so that future demand for space will be satisfied. The demand will be represented by teaching events (like lectures, tutorial, seminars etc.), that need to be assigned to the available space.

In practice, mixing happens more frequently in institutions, and from informal discussions with school managers, events are often assigned rooms of different spacetypes, e.g. Tutorials can be offered in lecture theatres if they're available, or seminar and workshop rooms etc.

Consequently, we give evidence that matching spacetypes when performing activities allocation, using current space profile, has a detrimental effect on the utilisation, that in sum, justifies mixing which occurs in many institutions.

On the the other hand, the "space profile" that took shape, when the estate was first built and made available for teaching, was generally tailored to the institution needs at the time, not considering its future growth and the increase in teaching events and activities. Later, as time evolves, and institutions grow bigger, varied sized events (lectures, tutorials) are assigned to rooms that don't really match their type, nor their size, therefore affecting utilisation. A new space profile would then be required to match the new teaching activities requirements. Remodelling space for purpose-fit accommodation is sometimes restricted by cost or the method of construction.

In this chapter, we address all these facts as follows: using a typical instance of current events, and two distinct methods, we try to alter the space profile in a way that best matches that instance and show that this newly created space profile potentially improves the expected utilisation.

The methodology goes as follows:

- First, given the initial (current) space profile, and an initial IP model, we plot the achievement curve allowing an approximation of the critical utilisation, which happens to be around 20-25 % (pretty low as we impose a hard penalty on the group size).
- Second, using 1) an extended IP model, or 2) a local-search-like method, and a typical instance of teaching events (e.g. with BOS equal to available seats), we alter the current space profile and generate a new profile that optimises utilisation for that specific instance.
- Third, using the newly created space profile, we plot another achievement curve that proves the new critical point has improved by around 10-15%.

Outline of the chapter: in section 7.3 we describe the mathematical models, in section 7.4 we expose the methodology to generate new space profiles, in section 7.5, we expose the effect of altering space profile on the critical utilisation.

7.3 Description and Formulation

As mentioned in **sec.** 2.2, the course timetabling terminology, is varied, and depends on the countries and the academic institutions. For a detailed overview refer to chapter 2.

A course, in this model, generally lasts multiple years and represent the type of study, students chose to join any specific institution. Course requirements let students enrol in `modules`, which are taught once a term, but several times in a week ; e.g. the `module` "Programming for CS" is taught on Monday 10:00 am, Wednesday 1:00am, Friday

10:00 pm. Student enrolled in that `module` should attend **"all of"** those taught meetings. Meetings of a `module`, where a student is required to attend **"all of"** them: `classes`. A brief description is presented here.

For every `module`, $k \in \{1, \ldots, q\}$ we associate the following:

1. **size** S_k : Number of students in `module` k.
2. **timeslots** T_k : Number of timeslots required by the `module` in a weekly schedule.
3. **department** d_k: Department administering `module` k.

Other aspects, belonging to `modules`, like special `module` features, or `module` preferences, can be thought of, yet we are not considering them in this study.
`classes` will carry the same information as their respective `modules` except for the timeslots and type.
For a `class` $i \in \{1, \ldots, n\}$ we associate the following:

1. **size** S_i: number of students of the `class` (equal to the number of students of the respective `module`).
2. **type** EP_i: lecture, workshop or tutorial, etc.
3. **department** d_i: department offering/managing the `class`.

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 per week

3. **spacetype** SP_j: space for lectures, workshops, tutorials, etc.

4. **department** d_j: the one that owns/administers the room.

The hard constraints that are always enforced:

1. **Capacity constraint**: `class`/`group` size cannot exceed room capacity.

2. **No-sharing constraint**: at most one `class`/`group` is allowed per "room-slot", where by room-slot we refer to a (room, timeslot) pair.

For the definition of the utilisation and the soft constraints that are used in this model, we refer the reader to chapter 5. In this chapter, however, we have introduced an additional constraint of spacetype matching:

Spacetype Penalty: We use the notion of *spacetype* penalty, to account for the desire of matching teaching events of a certain type to rooms with the same given spacetype. E.g. tutorials would need to be assigned to tutorial rooms. We simply set a penalty if the type of the class does not match that of the given room. E.g. given class i with type EP_i, is assigned to room j with spacetype SP_j , then there is a penalty matrix represented by $\Gamma_{ij}(EP_i,SP_j)$ where all entries are non-negative. If EP_i=SP_j then, $\Gamma_{ij} = 0$. The total spacetype penalty of a given assignment is the sum of this penalty over all rooms and all classes. In sum, this could be considered as a count of mixing violations as we allocate classes and groups to available rooms.

7.3.1 Mathematical formulation

The following parameters and variables are used for modeling the problem as an Integer Program (IP), with roomslots also here, denoting the available space to which `classes`/`groups` are allocated.

MODEL `SPM`**:** Given :

Q : set of all `modules`

q : total number of `modules`, $q = |Q|$

N : set of all `classes`

n : total number of `classes`, $n = |N|$

R : Set of all rooms

r : total number of rooms, $r = |R|$

P : set of all timeslots

p : total number of timeslots, $p = |P|$

M : set of all roomslots

m : total number of roomslots, $m = |M|$ and $m = rp$

D : set of all types/spacetypes $D = \{1, 2, 3\}$

1 : for Lectures; 2: for Workshops; 3: for Tutorials.

EP_i: type of `class` i.

SP_j: Spacetype of roomslot j.

$T^z = \{j \quad | j \in M : z \in P\}$: represents the set of roomslots corresponding to a given timeslot z, with $M = \bigcup_{z \in P} T^z$ and $\bigcap_{z \in P} T^z = \emptyset$.

For every timeslot $z \in P$, T^z is the subset which includes all roomslots of timeslot z. T^z is the set of roomslots considered at a given one time.

$A^b = \{j \quad | j \in M : b \in R\}$: represents the set of roomslots corresponding to a given room b, with $M = \bigcup_{b \in P} A^b$ and $\bigcap_{b \in P} A^b = \emptyset$.

For every room $b \in P$, A^b is the subset which includes all roomslots of room b.

$F^k = \{i \quad | i \in N : k \in Q\}$: represents the set of `classes` belonging to a given `module` k, with $N = \bigcup_{k \in Q} F^k$ and $\bigcap_{k \in Q} F^k = \emptyset$.

For every `module` $k \in Q$, F^k is the subset which includes all classes of `module` k.

$E^d = \{i \quad | i \in N : EP_i = d, d \in D\}$: represents the set of `classes` belonging to a given type d, with $N = \bigcup_{d \in D} E^d$ and $\bigcap_{d \in D} E^d = \emptyset$.

For every type $d \in D$, E^d is the subset which includes all classes of type d. In this model, we consider lectures, tutorial and workshop spacetypes,

$H^d = \{j \quad | j \in R : SP_j = d, d \in D\}$: represents the set of rooms belonging to a given spacetype d, with $R = \bigcup_{d \in D} H^d$ and $\bigcap_{d \in D} H^d = \emptyset$.

For every spacetype $d \in D$, H^d is the subset which includes all classes of spacetype d.

S_i : number of students enrolled in `class` i

C_j : capacity of roomslot j

$\mathcal{L}_{N\times M}(L_{ij})$: location matrix between event i and roomslot j with entries L_{ij}.

Γ_{ij}: spacetype matrix between `classes` i and 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 `group`s of `class` i

O^l : minimum occupancy allowed, i.e. minimum fraction of room seats to be filled

$C_{i_1 i_2}$: conflict matrix between 2 lectures i_1 and i_2.

B_L^{up} : upper limits on the location (L)

B_{GZ}^{up} : upper limit on `group` size (GZ) penalty

B_{SP}^{up} : upper limits on the spacetype penalty (Γ)

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

Decision variables:

Let: v_{ij} be the number of students of `class` i allocated to roomslot j, with $v_{ij} \geq 0$, and v_{ij} integer $\forall i \in N, \forall j \in M$.

And the decision variables:

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

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

$$z_k = \begin{cases} 1 & \text{if } \texttt{module}\ k \text{ is allocated.} \\ 0 & \text{Otherwise} \end{cases}$$

7.3.2 Objective function

The objective is to maximize the overall seat-hours used:

$$Obj^* = max\Big(\sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij}\Big) \tag{7.1}$$

Subject to :

Given that partial allocation is not allowed, we enforce

$$\sum_{j=1}^{m} v_{ij} = S_i x_i \qquad \forall i \tag{7.2}$$

Room capacities cannot be exceeded, and so we impose

$$v_{ij} \leq C_j y_{ij}, \quad \forall i \in N, j \in M; \tag{7.3}$$

This also links the v and y decision variables.

Only one `group` can be allocated to a given roomslot:

$$\sum_{i=1}^{n} y_{ij} \leq 1, \quad \forall j \in M \tag{7.4}$$

The location penalty must be less than the upper limit B_L^{up}:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} L_{ij} y_{ij} \leq B_L^{up} \tag{7.5}$$

When $B_L^{up} = 0$, hard location penalty is enforced.

To limit the group size (GZ) penalty for workshops we impose

$$\sum_{i=1}^{|E^2|}\sum_{j=1}^{m} \left| v_{ij} - G_i^t y_{ij} \right| \leq B_{GZ}^{up} \tag{7.6}$$

Constraint 7.6 is not required for tutorials because hard limits are imposed on their group size and also not required for lectures since they do not split .

To enforce the spacetype mixing penalty we impose:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \Gamma_{ij} y_{ij} \leq B_{SP}^{up} \tag{7.7}$$

We also impose an upper and lower limits on the group sizes for tutorials using:

$$v_{ij} \leq G_i^{up} y_{ij} \qquad \forall i \in E^3, j \in M \tag{7.8}$$

$$v_{ij} \geq G_i^{low} y_{ij} \qquad \forall i \in E^3, j \in M \tag{7.9}$$

Given that partial allocation of `module` is not allowed, `classes` of a `module` should either all be allocated or none, we enforce:

$$\sum_{i=1}^{|F^k|} x_i = |D| z_k, \quad \forall k \in Q; \tag{7.10}$$

We impose upper limits on the number of `groups` per workshop `class`, as generally they are required to be taught in 3-4 groups only:

$$\sum_{j=1}^{m} y_{ij} \leq G_i^{nb}, \quad \forall i \in E^2 \tag{7.11}$$

A lectures do not split, the number of groups per class should be less than or equal to one, we impose,

$$\sum_{j=1}^{m} y_{ij} \leq 1, \quad \forall i \in E^1 \tag{7.12}$$

If a roomslot is used then the given fraction O^l of room seats needs to be filled:

$$v_{ij} \geq O^l C_j y_{ij}, \quad \forall i \in E^2, j \in M \tag{7.13}$$

This is relevant only for `Workshops`. In this paper we use $O^l = 0.3$. (We investigated other values and found that smaller values, or turning off this constraint altogether, do not change the results we present here.)

The following constraint is entailed by the other constraints but we added it as it lead to a considerable reduction in computation times.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \leq rp \tag{7.14}$$

Timetabling constraints:

Lectures, workshops and tutorials of a given `module` should not be allocated to the same timeslot:

$$\sum_{i=1}^{|F^k|}\sum_{j=1}^{|T^\theta|} y_{ij} \leq 1 \quad \forall k \in Q, \theta \in P; \tag{7.15}$$

Constraint 7.16 imposes that no two lectures with common students be allocated to the same timeslots.

$$C_{i_1 i_2}(y_{i_1 j_1} + y_{i_2 j_2}) \leq 1, \quad \forall \theta \in P, \forall j_1 \in T^\theta, \forall j_2 \in T^\theta, \tag{7.16}$$

$$\forall i_1 \in E^1, \forall i_2 \in E^1 \tag{7.17}$$

$$i_1 \neq i_2, j_1 \neq j_2; \tag{7.18}$$

The conflict matrix $C_{i_1 i_2}$ between different lecture classes i_1, i_2 has been generated using the enrollment generator of chapter 6.

Finally, note that in the fixed choice mode, we simply enforce $\forall i.\ x_i = 1$ giving

$$\sum_{j=1}^{m} v_{ij} = S_i \qquad \forall i \tag{7.19}$$

The value of the objective Obj^* is then fixed, and the problem is simply that of feasibility . Please refer to chap 6 for more details on the effect of splitting on the timetabling conflicts.

7.4 Adapting space profile

7.4.1 EXT-SPM model extension

EXT-SPM: Extended model The model of section 5.3 used a fixed space profile determined by parameters SP_j. An interesting approach would be to let the solver chooose the space profile that would maximimise utilisation. we extended model SPM by replacing SP_j and matrix Γ_{ij} by the main decision variable:

$$w_{\rho\sigma} = \begin{cases} 1 & \text{if room } \rho \text{ is assigned spacetype } \sigma. \\ 0 & \text{Otherwise} \end{cases}$$

and a derived variable :

$$u_{j\sigma} = \begin{cases} 1 & \text{if roomslot } j \text{ is assigned spacetype } \sigma \ . \\ 0 & \text{Otherwise} \end{cases}$$

and adding the following constraints:

if a room is assigned a given spacetype all of it's respective roomslots are assigned that spacetype:

$$\sum_{j=1}^{|A^b|} u_{jd} = pw_{bd}, \quad \forall b \in R, \forall d \in D; \tag{7.20}$$

Only one spacetype assigned to any room:

$$\sum_{d=1}^{|D|} w_{bd} = 1, \quad \forall b \in R \tag{7.21}$$

Linking variables u to y, imposes that if a roomslot is assigned a given spacetype

d, then that roomslot can only be assigned `classes` of type d.

$$y_{ij} \leq u_{jd}, \quad \forall d \in D, \forall i \in E^d, \forall j \in M; \tag{7.22}$$

Only one spacetype assigned to any roomslot:

$$\sum_{d=1}^{|D|} u_{jd} \leq 1, \quad \forall j \in M \tag{7.23}$$

When using model `EXT-SPM`, constraint 7.7 is not used since the matrix Γ_{ij} is dropped and replaced by decision variables u and w. All other constraints in `SPM` remain unchanged in the extended model.

Note, as the maximisation will still be focused on maximising the utilisation, the extended model will still decide on the spacetype assignments without directly including variable w part of the objective function. w will remain however, a structural variable part of the extended model.

7.4.2 `LS-SPM`: Local search in space profile

7.4.3 Adapting space profile

In space planning, an interesting goal for space managers is to alter the space profile so that the projected critical utilisation can be improved.

In this chapter adapting space profiles to a given fixed instance of events, is done in two ways:

First, by extending model `SPM` to become `EXT-SPM`.

Second, by using procedure `LS-SPM` and searching the spacetype neighbourhood of space profiles.

These methods would tailor the space profile to a single "real-world", fixed instance of teaching events. We try changing the space profile, using a local-search-like method, when large instances are in question. For smaller instances and for finding upper bounds on a given solution, the integer programming approach will be sufficient.

Local Search operators

The operators used in procedure `LS-SPM` are given below. They allow changes and alteration to the space profile. All operators maintain feasibility of the profile.

`Swap-type:` OP^1**,** Randomly select 2 different rooms with two different spacetypes, and swap their spacetypes: e.g. if selected room 1 is a lecture room and room 2 is a tutorial room, then after applying this operator, room 1 becomes a tutorial room and room 2 a lecture room.

`Alter-type:` OP^2**,** Randomly select a room and alter it's spacetype. Spacetypes can be one of (lecture, workshop or tutorial).

Proc: LS-SPM

1 Set $Obj_1 = 0$;
2 Load Rooms vector : Ro_i
3 load current Rooms vector : $Ro_c = Ro_i$
4 Load empty room vector Ro_d
5 Load modules vector E_i
6 LOOP 1 to `Tcriteria` or `No improvement`
7 Randomly select operator OP^X, with $X \in \{1, 2\}$.

8 apply to rooms vector(Ro_c): $Ro_d = OP^X(Ro_c)$
9 run CPLEX with Model `SPM` on (E_i,Ro_d)
10 get $Obj^*= Max(U)$
11 IF $Obj^* \geq Obj1$
12 $Obj_1 = Obj^*$
13 assign Ro_d vector to Ro_c: $Ro_c = Ro_d$
14 END IF
15 END LOOP
16 Return space profile: Ro_c

Procedure `LS-SPM` is a simple variant of local search, used to improve the current space profile. Starting with a fixed instance of events and rooms, the algorithm applies a perturbation/move using operators OP^{1-2} to rooms vector (static room profile). Then using the current event instance, it solves model `SPM` and finds the maximum utilisation Obj^*. If Obj^* is larger or equal to the incumbent then keep perturbation, otherwise, revert back to previous configuration. The algorithm loops until reaching a given termination criteria. This procedure is faced the difficult task of optimising model `SPM` so that the run-time is reduced. This procedure is performed over 6 hours of computation time, owing to the difficulty of quickly solving the IP model with the Integer Programming solver (CPLEX).

7.5 Results

7.5.1 New spacetype profile with procedure LS-SPM

Figure 7.1 is a histogram comparing the initial and adapted space profile using procedure `LS-SPM` without altering the room sizes (Operators OP^1 and OP^2 used).

One can notice how the algorithm "selects" small-sized room and 7.2 assigns them to tutorials. Workshops and Lectures are left with largest ones. The total number of room-

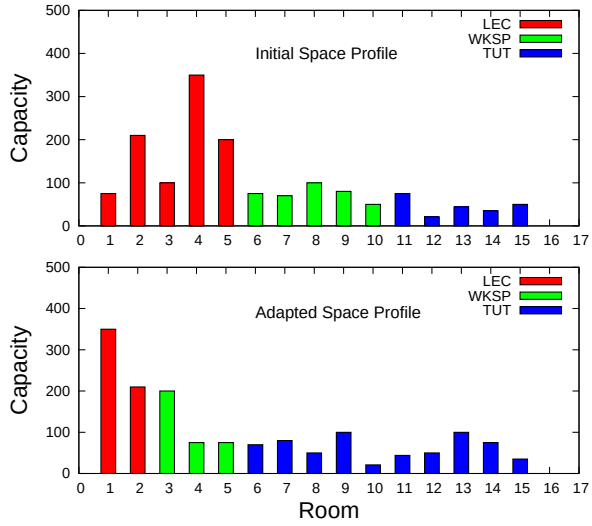

FIGURE 7.1: Adapting Space profile using procedure LS-SPM starting with a real world instance .

slots suffice to fit all lectures in just 2 larger rooms. The method is very slow, owing to running the SPM model in every iterations with the spacetype penalty enforced.

Similarly, in figure 7.2, we compare the initial and adapted space profile using the exact model EXT-SPM. The adapted profile is very similar to that of figure 7.1. In both our models the instance used has 20 modules having one lecture each.

7.5.2 Spacetype mixing effects on Utilisation

Figure 7.3 is an approximation plot of utilisation versus spacetype penalty for a typical "real-world" instance. The error bars represent the actual integral value of the utilisation (up to 2 % MIP gap) and the upper bound on the utilisation value resulting from the linear relaxation of model SPM. Enforcing a lower limit on mixing violations drives the utilisation by roughly 20%. The large number of tutorial groups generally require more tutorial room-slots than are available, and when we force groups to match their respective spacetypes,

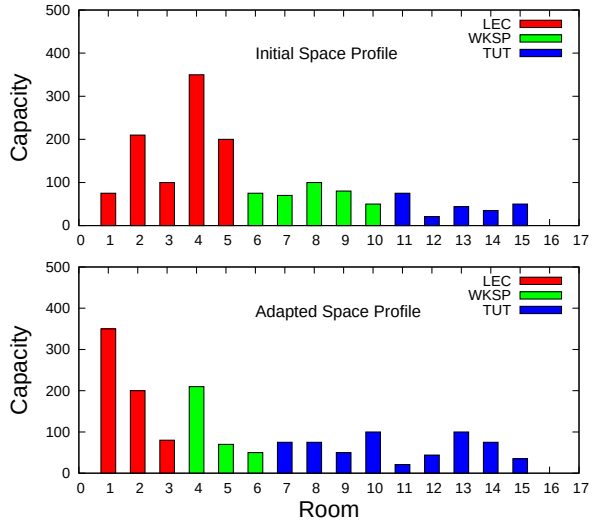

FIGURE 7.2: Adapting space profile using model `EXT-SPM` and same real-world instance as figure 7.1

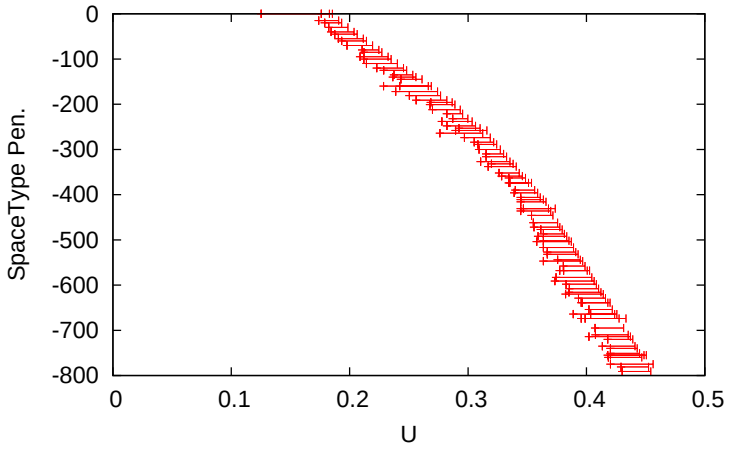

FIGURE 7.3: Spacetype mixing effects on utilisation. Error bars represent the integer solutions and their respective upper bound

there would be a knock on effect on the utilisation, as seen in this figure. In practise, most universities mix spacetype to counter balance this loss. In the next section we consider the effects of the spacetype penalty on the critical utilisation, for different projected demands [1]. *NOTE: We use a small dataset with only 15 timeslots, with large events and small rooms, creating a large number of splits per module, this justifies the low utilisation under sp-Penalty 0. Anyway 20-30 % utilisation is very typical of institution the average being 25% according to SMG*[2]

7.5.3 Safety in static, fixed-adapted models and dynamic-adapted profiles

In previous sections, we explored methods to alter the space profile. In this set of experiments, we plot the achievement curves (see **chap.** 5), to study the effects of altering the space profile, on the critical utilisation. The safety curves, as explained in **sec.** 5.4, make use of some model (e.g. `SPM`), a given set of rooms and some random subsets of teaching events, and looks for utilisation levels where it's "safe" to perform full allocation. In these coming experiments, we will compare critical utilisation levels for different sets of rooms, e.g. different space profiles.

We start by plotting the safety and achievemet curves:

First, using an initial "real-world" static profile, (`Static` case).

Second, perform this same procedure, but with a space profile adapted with procedure `LS-SPM` (`Fixed-adapted`, $B^{up}_{SP} = 0$).

[1] The spacetype penalty values are multiplied by -1 as we seek the maximum utilisation for the minimum spacetype penalty

[2] UK Higher Education Space Management Group. See www.smg.ac.uk

Third, we use model `EXT-SPM` to plot the achievement curves while adapting the space profile for every random subset generated (`Dynamic-adapted`, that provides an upper bound on the best possible utilisation.

Fourth, In all experiments we prohibit any mixing, this experiment, uses the adapted profile, but allows some mixing violations (`Fixed adapted` $B^{up}_{SP} = 200$).

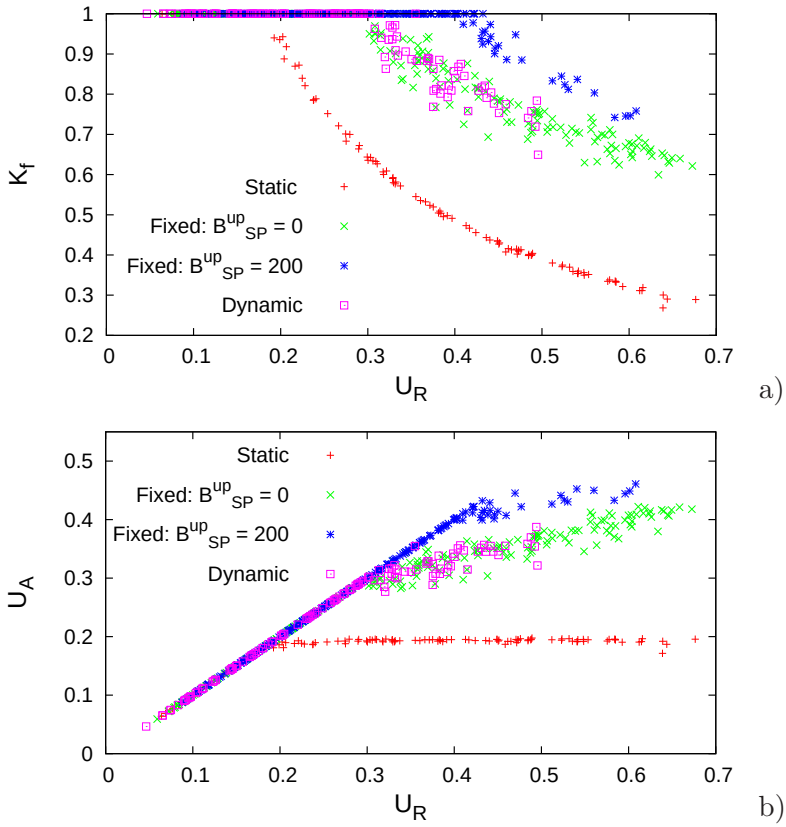

FIGURE 7.4: Safety and achievement curve of the requested utilisation U_R versus (a) K_f and (b) U_A, for `Static`, `Fixed-adapted` with $B^{up}_{SP} = 0$, `Fixed-adapted` with $B^{up}_{SP} = 200$, and `Dynamic-adapted` with $B^{up}_{SP} = 0$ models

Figure 7.4 presents the results of our experiments. In figure 7.4 (a), K_f or "Fractional Achievement Ratio", is the fraction of the requested utilisation that is achievable. The first observation, is that the values of achieved U_A for corresponding requests, are

grouped around the mean. Seemingly, small changes in U_A between points near to some value of U_R are small compared to the value of U_A itself, for (a) and (b). This implies that properties of the system are statistically predictable.

The adapted space profiles have improved the utilisation by more than 12-15 % over the static model. However, in order to reach the utilisation level where all mixing violations are allowed (around 42 % in figure 7.3 with $B_{SP}^{up} = 800$), setting a value of 200 as an upper limit for the spacetype penalty, can further improve the expected utilisation. Some mixing violations should generally be allowed, since when adapting the space profile, we alter the spacetype of all roomslots of a room (e.g. we can't consider a room to be of tutorial type half of the time and the other half a workshop room). Allowing that bit of mixing will overcome this small utilisation waste. Note that the fuzzy region of the adapted cases is indicative that for some requested utilisation, different levels can be achieved. For more details on the size of the problems and datasets please refer to the appendix (A) .

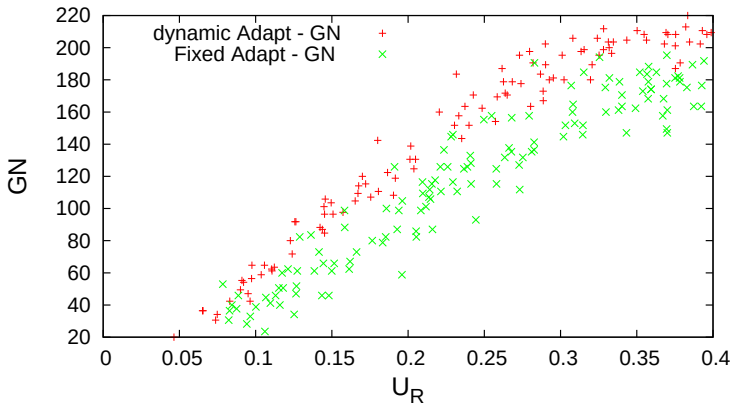

FIGURE 7.5: Total group number versus requested utilisation for the Fixed and Dynamic adapted cases.

	Lec Mean	**Wksp Mean**	**Tut Mean**	**Lec Std**	**Wksp Std**	**Tut Std**
Static	0.615	0.7621	0.515	0.060	0.063	0.032
Fixed Adapt	0.505	0.763	0.339	0.066	0.074	0.042
Dynamic Adapt	0.560	0.766	0.348	0.104	0.126	0.070

TABLE 7.1: Average occupancy and standard deviation, for `Static`, `Fixed` and `Dynamic` adapted cases

.

In figure 7.5, we plot the group numbers allocated versus U_R for the `Fixed-adapted` and `Dynamic-adapted` case. Clearly, the number of groups generated by the `dynamic-adapted` case, provides an upper bound on the `fixed-adapted`. We have used just one scenario when searching for the fixed adapted space profile. This would suggest that if the frequency of usage of rooms has the possibility of theoretically beeing improved, the utilisation achieved cannot in general. That in sum suggests that we should consider the effects of occupancy, which will be discussed in the next section.

7.5.4 Occupancy

Table 7.1 presents the average occupancy per spacetype, for the static, fixed and dynamically adapted cases, as well as the standard deviation for each of the three cases. The occupancy is measured for every solved instance as being the average over all rooms, of seats used over available seats. We first notice that tutorials, expectedly have the lowest occupancy of the three cases (0.55, 0.33), That is surely due to the constraint group size being in a range (15-20), less than the average room size.

However, unexpectedly, as we alter the spacetype profile, the occupancy tends to decrease, mainly in the case of tutorial and lectures. The results suggest that the spacetype profile improvement has dramatically improved the frequency of usage but not the room

occupancy. We believe that the best approach to fill rooms completely is to alter the room size profile and that is part of future work. Note that allowing a free split in the case of workshops did in fact improve the workshop occupancy as we alter the space profile.

7.6 General applicability of our approach

The achievement curves, exposed in previous sections, provide a view on the future quality of achievable allocations. In particular, within a given institutional context and set of resources and constraints, the critical utilisation gives a measure of the largest utilisation that can be safely achieved. Initial work on space planning focused on determining the critical utilisation for a given fixed set of (teaching space) resources. However, the natural goal in space planning is then to develop methods to adjust the sets of resources so as to increase the critical utilisation. Therefore, such an approach, has a rather global scope and is naturally extendible to encompass a wide spectrum of different academic settings and enrolements.

Furthermore, in contrast with our previous work, which used a single spacetype but added adjustements to the room-size profile, the work here was centered around altering the space profile (including spacetypes) such that better utilisation can be achieved.

However, for the purpose of result consolidation, an interesting, generalizable approach would be to use stochastic programming and simultaneously optimize with respect to multiple scenarios. This is part of future research being undertaken at the university of Nottingham.

7.7 Summary

This chapter is an initial study with small test cases, of the long-term planning and remodelling aspect of teaching space. We have proposed methods to alter the *space profile* (room spacetypes), and studied the spacetype mixing requirements and effects on space utilisation

when `module characteristics` don't really match the `space profile`. We have used a local-search based method and an Integer Programming formulation when altering room spacetypes, and found that this change has considerably improved the critical utilisation. This chapter lays the foundation for future work discussed in the next chapter, where good planning would consider the uncertain growth of universities, and matches it with decisions on space expansion.

Chapter 8

Conclusion and Future Work

8.1 Introduction

Space planning and management problems directly affect the academic sector. Problems related to space provisions are mostly over constrained by, first, the availability of space and, second, the need to satisfy timetabling and pedagogical constraints at all levels of the academic hierarchy. This thesis is an initial attempt, to provide an in-depth study of space utilisation in academic institutions and explore models and methodologies that could assess utilisation levels, in the hope of improving them in practice. Moreover, we have studied the problem of splitting, where large modules or classes, being too large to fit into available rooms, would need to be broken up in smaller groups, and we devised algorithms to study splitting effects on space utilisation.

8.2 Identifying real-world constraints and an initial investigation

In this thesis, we aimed at creating a better understanding of the factors required for an efficient management and planning of teaching space allocation. A fundamental stage

of capacity planning is to estimate the projected student enrolments and multiply it by the expected weekly student contact hours to obtain the total demand for `seat-hours`. Similarly, by summing up the room capacities and multiplying by the number of hours for which rooms are available, we can 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*; the fraction (or percentage) of available seat-hours that actually end up being used. In many institutions, the utilisation can be surprisingly low, commonly only 20-50%.

Attempts, therefore, to remedy this situation, and so to improve space planning are generally hampered because there does not seem to be an agreed or *qualitative understanding* of why utilisation is so low in the first place. Furthermore, the utilisation figures incorporated into space norms are obtained from standard sources, and are based more on statistical studies of available space and enrolments in some universities, and might as well be inappropriate for a modern module system. Hence, we initially, had to tackle two important goals:

1) Develop an understanding of the factors leading to low utilisation. To these ends, we considered a simple a pure event allocation problem in which we optimise utilisation by taking events and assigning them to available timeslots. On the data-sets we have available, this immediately gave utilisation of 85-90%. This is far too high to match reality, and so indicates that a model based purely on space issues, and given free choice of classes,

is inadequate to model the problem of managing teaching space allocation in real-world universities. To extend the model we have added extra constraints, where event-allocation usually takes place within the context of many constraints on locations and timings of events (timetabling penalty). Accordingly, we included within our model objectives intended to provide a simplified approximation/abstraction of real timetabling issues.

We found that the location and timetabling-based objectives do indeed have the potential to drive down the utilisation levels. We also found that if classes are selected in advance, then the reliably achievable utilisation can be much lower than when class selection is done by the optimiser.

2) Study factors leading to low utilisation in the splitting case. In this second goal, we had to consider the case where event sizes are larger than available rooms, and consequently would require splitting. Also, for ease of use of a multiobjective study, we have incorporated specially designed splitting algorithms as part of the local search. It was observed that splitting affects the solution quality and splitting could itself be affected by other penalties like *Location* and *Timetabling*. Whenever one mentions "timetabling", immediately, course timetabling comes to mind, and with splitting involved, one would consider the sectioning problem[1]. However, we have not studied such *immediate* space management problems as the sectioning problem. Instead, we were concerned with decision support for space capacity planning over a longer time frame. For space planning, we needed to understand which utilisations are achievable and how they depend on the decision criteria: such as group sizes, group number, and the constraints arising from location and

[1]See **chap.** 2

timetabling. We have therefore devised algorithms to do splitting together with event allocation. We also explored the trade-offs between the various objectives, so that we could 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 were used to investigate long-term space planning and not to address near-term space management which is associated to timetabling. Therefore, our approach was to: 1) Formulate or model the splitting problem such that it incorporates most of the main real world aspects - although it does not need to contain all the details. For example, we have covered the small group requirements by simply introducing objectives related to the group size or number. 2) Used local search and simulated annealing to explore the solution space and deal with the splitting problem. 3) Carried out experiments in order to visualise the trade-off surfaces. The specific contributions made were as follows. **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 classes, and can also rejoin them in order to suit the available rooms. **Preliminary trade-off surfaces :** where we presented results on the interaction of objectives such as location and timetabling, with preferences on group sizes and number.

8.3 Exact models and phase transition

In **chap.** 5 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, 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 utilisation. Accordingly, we extended the previous work by investigating the runtimes needed to solve the arising problems. Here, we have shown that there is a standard *Easy-Hard-Easy* pattern in the hardness, with problems near the threshold being much harder to solve. Whilst some utilisation levels are easily achievable, some levels require much higher computational effort. 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. We concluded that there is close link between improving utilisation and improving solver technologies. We have 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. Hardness peaks were found to occur when using two different algorithms, giving strength to believing that the hardness peaks are real. It may be that a specialised algorithm can make hard peaks disappear, but even if this was the case here, it is still important for the administrator or planner to be aware of effects of the choice of solver on the achievable utilisation. This provides a novel role for the *Easy-Hard-Easy* patterns that has been so extensively studied within the Artificical Intelligence community. Not only do they provide a source of challenging instances for algorithm development, but also they can have a direct interaction with practical issues in space planning and management.

8.4 Student-centred models and timetabling penalty relaxation

Conflicts between two teaching events generally happen when students simultaneously, enrol in both events, and therefore could not be assigned to the same timeslot[2]. Conflicts are represented by a conflict graph, where vertices are the events themselves and edges represent the actual conflicts. In standard timetabling concepts, the conflict graph will be fixed, and the "*timetabler*" is supposed to assign times to events so that no conflicting events are assigned the same times. However in the case of splitting, the conflict will surely change as those events are broken up into smaller groups. Hence, if a class has multiple groups, then not every group ought to have the same conflicts as the parent class. The question that naturally arises: "how do conflicts get transmitted from parent events to groups as splitting occurs ?". The main assumption being that, a proper assignment of students to groups can lead to dropping the timetabling penalty and relaxing the conflicts, resulting with a partial inheritance close to zero.

As the timetabling penalty was extensively used in this thesis there was a need to justify this claim by providing evidence that splitting, in our case, does relax class conflicts. For this purpose, we have studied the partial inheritance by introducing a *student-centred* model, and considered the behaviour of utilisation under changing student enrolment. We gave proof that when a class is divided into more then one group, and given specific group size and numbers, conflicts can be fully resolved. We have also studied the robustness of the partial inheritance under changing student enrolments. This also required the design of a new enrolment generator *genRol* which allowed us to simulate real-world student enrolments.

[2]Those events cannot be taught at the same time.

8.5 Long-term space planning and spacetype mixing

Finally, we have looked at the long-term space planning problem.

Attempts to study and improve utilisation levels were centered so far on the short-term space management aspect: " how to split and assign demand to roomslots".

In most academic institutions, teaching space is composed of different sized rooms, with different usages e.g. tiered/flat, tutorial rooms, laboratory, etc. Rooms are attributed therefore a given spacetype (e.g. lecture, tutorial, workshop, seminar etc.), and generally host academic activities of a similar type, for example, lectures are offered in lecture theatres, tutorials in tutorial rooms, etc.

Room characteristics (sizes and spacetypes) form the `space profile` of the available space. Therefore altering one room's spacetype, will affect the "space profile".

In day-to-day management practice, mixing spacetype occurs frequently, and, events are often assigned rooms of different spacetypes, e.g. Tutorials can be offered in lecture theatres if they're available, or seminar and workshop rooms etc.

This thesis considered matching spacetypes when performing activities allocation, using current space profile, and found that this mixing has a detrimental effect on the utilisation.

Moreover, the `Space Profile` that was generally seen in universities does not really match the class/module profile (which represent demand), since with time demand grows and curricula changes. Remodelling space for purpose-fit accommodation is highly restricted by cost.

We have therefore proposed two distinct methods that alter the space profile in a

way that best matches current demand, and showed that the newly created space profile can potentially improve the expected utilisation. Using the achievement curves and the notion of safety discussed in **chap.** 5, we gave evidence that altering the space profile have improved the critical utilisation by around 15-20 %.

8.6 Future work

This thesis represents an initial study into teaching space allocation, and lays the foundation for new upcoming research in teaching space. In this section, we look at some of the future research directions:

8.6.1 Proposed model

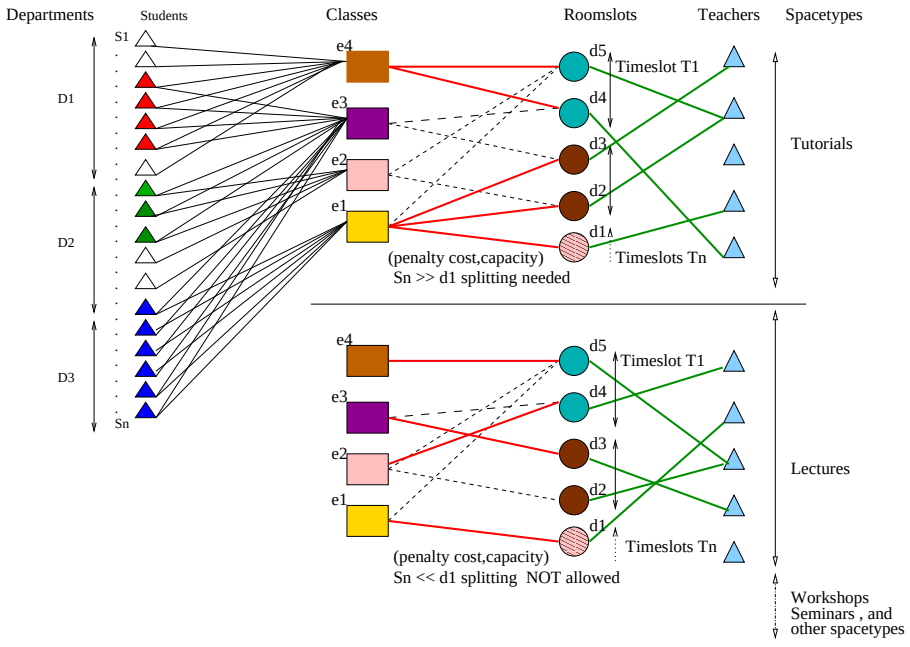

FIGURE 8.1: A global view at the model for Teaching Space Allocation.

Figure 8.1 provides the global picture of the model we're aiming at for the teaching

space allocation problem. This model extends the current one and groups allocation of modules and individual students to rooms, timeslots and teachers, all in a spacetype mixing environment.

From the pure event based allocation with timetabling, then to the splitting aspects and then to mixing spacetypes and altering space profiles (including spacetypes and room sizes), solving our model becomes a challenge at this point. The main question would lie in the extent to which, given this model, we can neglect student allocation and move from a pure *student-centred* to *course-centred* in order to study planning issues. We envisage a new simulation based approach, flexible enough to assist administrators in in their decision making on short-term and long-term space planning. The *Simulation Optimisation* would gather different specialised solvers and algorithms including exact methods like large scale integer programming and dynamic programming techniques, to solve subsets of the problem while delegating other subsets to heuristic solvers.

8.6.2 Phase transition under spacetype mixing

In **chap.** 5 we used the *Phase Transition* and *Easy-Hard-Easy* concepts, on teaching space allocation. We studied the computational hardness at the *threshold* for spacetypes separately, in the presence of different penalties and constraints (like location and group size) . We have shown that the hardness varies between spacetype and cross-penalties. The future work on this front will focus on studying the *threshold* behaviours under spacetype mixing and probably find new levels of difficulty as mixing can bring together different instances with different computational hardness patterns.

8.6.3 Room profile optimisation

In **chap.** 7 we looked at altering the spacetype profile of a room, and we studied the effects on the critical utilisation. Further work could explore altering the roomsizes rather then spacetypes such that expected utilisation can be improved. Modelling this requires a stochastic approach, for decisions made "here-and-now" will affect future growth and student enrolments.

References

Aarts, E. and Korst, J. (1990). *Simulated Annealing and Boltzman Machines*. Wiley.

Abdelaziz Dammak, Abdelkarim Elloumi, H. K. (2006). Lecture and tutorial timetabling at a tunisian university. In *Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 384–390, Brno, Czech Republic.

Achterberg, T. (2004). SCIP – A framework to integrate constraint and mixed integer programming. Technical report, Zuse Institute, Berlin, Berlin. ZIB : TR-04-19.

Al-Yakoob, S. M. and Sherali, H. D. (2006). Mathematical programming models and algorithms for a class-faculty assignment problem. *European Journal of Operations Research (EJOR)*, **173**(2), 488–507.

Al-Yakoob, S. M. and Sherali, H. D. (2007). 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.

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

AminToosi, M., Yazdi, H. S., and Haddadnia, J. (2004a). Feature selection in a fuzzy student sectioning algorithm. In *Selected papers from the Fifth International Conference for the Practice and Theory of Automated Timetabling (PATAT 2004)*, volume 3616, pages 147–160. Lecture Notes in Computer Science, Springer-Verlag.

AminToosi, M., Yazdi, H. S., and Haddadnia, J. (2004b). Fuzzy student sectioning. In *Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004)*, pages 421–424.

Aubin, J. and Ferland, J. A. (1989). A large scale timetabling problem. *Computers & OR*, **16**(1), 67–77.

Avella, P. and Vasil'ev, I. (2005). A computational study of a cutting plane algorithm for university course timetabling. *J. Scheduling*, **8**(6), 497–514.

Badri, M. (1996). A two-stage multiobjective scheduling model for [faculty-course-time] assignments. *European Journal of Operational Research*, **94**, 16–28.

Banks, D., van Beek, P., and Meisels, A. (1998). A heuristic incremental modeling approach to course timetabling. In *Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, volume 1418, pages 16–29.

Bloomfield, S. and McSharry, M. (1979). Preferential course scheduling. *Interfaces*, **9**, 24–31.

Boland, N., Hughes, B. D., Merlot, L. T., and Stuckey, P. J. (2006). New integer linear

programming approaches for course timetabling. *Computers and Op. Res.* In Press, Corrected Proof, Available online 11 December 2006.

Bollobas, B. (1985). *Random Graphs*. Academic Press, London, England.

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

Bosch, R. and Trick, M. A. (2005). Integer programming. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 69–96. Springer, New York, NY.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. CUP, Cambridge, UK.

Breslaw, J. A. (1976). A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Sciences*, **10**(6), 227–230.

Bullock, N. (1974). Modeling the demand for teaching space. *Environment and planning*, **1**, 69–103.

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

Burke, E. K. and Carter, M. W., editors (1998). *Practice and Theory of Automated Timetabling, Second International Conference, PATAT '97, Toronto, Canada, August 20–22, 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*, Berlin. Springer.

Burke, E. K. and Causmaecker, P. D., editors (2003). *Practice and Theory of Automated Timetabling, Fourth International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, volume 2740 of *Lecture Notes in Computer Science*, Berlin. Springer.

Burke, E. K. and Erben, W., editors (2001). *Practice and Theory of Automated Timetabling, Third International Conference, PATAT 2000, Konstanz, Germany, August 16–18, 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*, Berlin. Springer.

Burke, E. K. and Ross, P., editors (1996). *Practice and Theory of Automated Timetabling, First International Conference, PATAT '95, Edinburgh, U.K., August 29 – September 1, 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*, Berlin. Springer.

Burke, E. K. and Rudová, H., editors (2006). *Practice and Theory of Automated Timetabling, Sixth International Conference, PATAT 2006, Brno, The Czech Republic, August 30 – September 1, 2006, Proceedings*, Brno, The Czech Republic. Masaryk University.

Burke, E. K. and Trick, M. A., editors (2005). *Practice and Theory of Automated Timetabling, Fifth International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18–20, 2004, Revised Selected Papers*, volume 3616 of *Lecture Notes in Computer Science*, Berlin. Springer.

Busam, V. A. (1967). An algorithm for class scheduling with section preference. *Commun. ACM*, **10**(9), 567–569.

Carrasco, M. P. and Pato, M. V. (2004). A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem. *European Journal of Operational Research*, **153**(1), 65–79.

Carter, M. (2000). A comprehensive course timetabling and student scheduling system at the University of Waterloo. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling, Third International Conference, Konstanz*, pages 64–82. Springer.

Carter, M. and Laporte, G. (1998). Recent developments in practical course timetabling. In *Selected papers from the second International Conference on Practice and Theory of Automated Timetabling*, pages 3–19. Springer-Verlag.

Carter, M. and Tovey, C. (1992). When is the classroom assignment problem hard? *Operations Research*, **40**(1), 28–39.

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

Cheng, C., Kang, L., Leung, N., and White, G. M. (1995). Investigations of a constraint logic programming approach to university timetabling. In *Selected papers from the First International Conference on the Practice and Theory of Automated Timetabling (PATAT 1995)*, volume 1153, pages 112–129. Lecture Notes in Computer Science, Springer-Verlag, London, UK.

Cheng, E., Kruk, S., and Lipman, M. (2003). Flow formulations for the student scheduling

problem. In *Selected papers from the Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740, pages 299–309, Gent, Belgium. Lecture Notes in Computer Science, Springer-Verlag, London, UK.

Coarfa, C., Demopoulos, D. D., San Miguel Aguirre, A., Subramanian, D., and Vardi, M. (2000). Random 3-SAT: The plot thickens. In *Proceedings of 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894, page 143, Singapore. Lecture Notes in Computer Science, Springer Berlin / Heidelberg.

Corr, P., McCollum, B., McGreevy, M., and McMullan, P. (2006). A new neural network based construction heuristic for the examination timetabling problem. In *Lecture Notes for Computer Science*, volume 4193, pages 392–401, New York. Springer-Verlag.

Dantzig, G. B. and Thapa, M. N. (1997). *Linear programming I: Introduction*. Springer, New York, NY.

Daskalaki, S., Birbas, T., and Housos, E. (2004). An integer programming formulation for a case study in university timetabling. *Eur. J. Op. Res.*, **153**, 117–135.

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

Deb, K. (2005). *Search Methodologies: Introductory Tutorial in Optimization and Decision Support Techniques*, chapter Multi-objective optimization, pages 273–316. Springer.

Diminnie, C. B. and Kwak, N. (1986). A hierarchical goal-programming approach to reverse resource allocation in institutions of higher learning. *The Journal of the Operational Research Society*, **37**(1), 59–66.

Dimopoulou, M. and Miliotis, P. (2001). Implementation of a university course and examination timetabling system. *Eur. J. Op. Res.*, **130**, 202–213.

Dimopoulou, M. and Miliotis, P. (2004). An automated university course timetabling system developed in a distributed environment: A case study. *Eur. J. Op. Res.*, **153**, 136–147.

Dinkel, J. J., Mote, J., and Venkataramanan, M. A. (1989). An efficient decision support systems for academic course scheduling. *Oper. Res.*, **37**(6), 853–864.

Dyer, James S.; Mulvey, J. M. (1976). An integrated optimization/ information system for academic departmental planning. *Management Science,*, **22**, 1332–1341.

Erben, W. (2001). 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)*, volume 2079, pages 132–158, London, UK. Springer-Verlag.

Ferland, J. A. and Fleurent, C. (1994). Saphir: A decision support system for course scheduling. *Interfaces*, **24**(2), 105–115.

Ferland, J. A. and Roy, S. (1985). Timetabling problem for university as assignment of activities to resources. *Computers & OR*, **12**(2), 207–218.

Fizzano, P. and swanson, S. (2000). Scheduling classes on a college campus. *computational optimization and applications*, **16**, 279–294.

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

Gaspero, L. D. and Schaerf, A. (2002). Multi-neighbourhood local search with application to course timetabling. In *Selected papers from the Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002).*, volume 2740, pages 262–275, Gent, Belgium. Spriger-Verlag, London, UK.

Gent, I. P. and Walsh, T. (1995). The number partition phase transition. Research report, 95-185, Department of Computer Science, University of Strathclyde.

Gent, I. P. and Walsh, T. (1996). Phase transitions and annealed theories: Number partitioning as a case study. In *European Conference on Artificial Intelligence (ECAI-1996)*, pages 170–174, Budapest, Hungary. John Wiley & Sons.

Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1996). The constrainedness of search. In *AAAI/IAAI, Vol. 1*, pages 246–252.

Glassey, C. and Mizrach, M. (1986). A decision support system for assigning classes to rooms. *Interfaces*, **16**(5), 92–100.

Harwood, G. and Lawless, R. (1975). Optimising organisational goals in assigning faculty teaching schedules. *Decision sciences*, **6**, 513–524.

Head, C. and Shaban, S. (2007). A heuristic approach to simultaneous course/student timetabling. *Computers & OR*, **34**(4), 919–933.

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

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

Hertz, A. and Robert, V. (1998). Constructing a course schedule by solving a series of assignment type problems. *European Journal of Operational Research*, **108**, 585–603.

Hooker, J. N. (2002). Logic, optimization, and constraint programming. *INFORMS J. on Computing*, **14**(4), 295–321.

Hooker, J. N. (2005). A hybrid method for the planning and scheduling. *Constraints*, **10**(4), 385–401.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: A survey. *J. Log. Prog.*, **19/20**, 503–581.

Kambi, M. and Gilbert, D. (1996). Timetabling in constraint logic programming. In *Proceedings of the Symposium and Exhibition on Industrial Applications of Prolog (INAP-96)*, Tokyo, Japan.

Karp, R. M. (1986). Combinatorics, complexity, and randomness. *Comm. ACM*, **29**(2), 98–109.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

Kovacic, M. (1993). Timetable construction with markovian neural network. *European Journal of Operational Research*, **69**(1), 92–96.

Laporte, G. and Desroches, S. (1984). Examination timetabling by computer. *Comput. Oper. Res.*, **11**(4), 351–360.

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

Lee, S. M. and Clayton, E. R. (1972). A goal programming model for academic resource allocation. *Management Science*, **18**, 395–408.

McCollum, B. and McMullan, P. (2004). The cornerstone of effective management and planning of space. Technical report, Realtime Solutions Ltd.

McCollum, B. and Roche, T. (2004). Scenarios for allocation of space. Technical report, Realtime Solutions Ltd.

McNamara, J. F. (1971). Mathematical programming models in educational planning. *Review of Educational Research*, **41**, 419.

Milano, M. and Wallace, M. (2006). Integrating operations research in constraint programming. *4OR*, **4**, 1–45.

Milano, M., Ottosson, G., Refalo, P., and Thorsteinsson, E. S. (2002). The role of integer programming techniques in constraint programming's global constraints. *INFORMS J. on Computing*, **14**(4), 387–402.

Mingers, J. and O'Brien, F. A. (1995). Creating student groups with similar characteristics: A heuristic approach. *Omega*, **23**(3), 313–321.

Mirhassani, S. A. (2006). A computational approach to enhancing course timetabling with integer programming. *App. Math. Comp.*, **175**, 814–822.

Mirrazavi, S. K., Mardle, S. J., and Tamiz, M. (2003). A two-phase multiple objective approach to university timetabling utilising optimisation and evolutionary solution methodologies. *Journal of Operations Research Society (JORS)*, **54**(11), 1155–1166.

Miyaji, I., Ohno, K., and Mine, H. (1988). Solution method for partitioning students into groups. *European Journal of Operational Research*, **33**(1), 82–90.

Müller, T., Rudová, H., and Barták, R. (2005). Minimal perturbation problem in course timetabling. In Burke and Trick (2005), pages 126–146.

Mulvey, J. (1983). A classroom/time assignment model. *European Journal of Operational Research*, **9**, 64–70.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer Programming and Combinatorial Optimization*. Wiley, New York.

Papoutsis, K., C., V., and E., H. (March 2003). A column generation approach for the timetabling problem of greek high schools. *Journal of the Operational Research Society*, **54**, 230–238(9.

Parkes, A. J. (1997). Clustering at the phase transition. In *AAAI/IAAI*, pages 340–345.

Qualizza, A. Serafini, P. (2005). A column generation scheme for faculty timetabling. In *Selected papers from the fifth International Conference on the Practice and Theory of*

Automated Timetabling (PATAT 2004), volume 3616, pages 161–173, Pittsburgh, Pennsylvania. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg.

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

Ritzman, L., Bradford, J., and Jacobs, R. (1979). A multiple objective approach to space planning for academic facilities. *Management Science*, **25**(9), 895–906.

Rodošek, R., Wallace, M. G., and Hajian, M. T. (1999). A new approach to integrating mixed integer programming and constraint logic programming. *Ann. Op. Res.*, **86**, 63–87.

Ross, P., Corne, D., and Terashima, H. (1996). The phase-transition niche for evolutionary algorithms in timetabling. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling (PATAT 95)*, volume 1153, pages 309–324, London, UK. Springer-Verlag.

Rossi-Doria, O., Blum, C., Knowles, J., Sampels, M., Socha, K., and Paechter, B. (2002). A local search for the timetabling problem. In *Proceedings of the Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002)*, pages 124–127, Gent, Belgium.

Rossi-Doria, O., Sample, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., and Stützle, T. (2003). A comparison of the performance of different metaheuristics on the timetabling problem. In *Selected papers from the Fourth International Conference on the Practice and*

Theory of Automated Timetabling (PATAT 2002), volume 2740, pages 329–351, Gent, Belgium. Lecture Notes in Computer Science, Springer-Verlag, London, UK.

Rudová, H. and Matyska, L. (2000). Constraint-based timetabling with student schedules. In E. Burke and W. Erben, editors, *Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, volume 2079, pages 109–123, Konstanz. Hochschule Konstanz.

Rudová, H. and Murray, K. (2003). University course timetabling with soft constraints. In Burke and Causmaecker (2003), pages 310–328.

Sampson, S. E., Freeland, J. R., and Weiss., E. N. (1995). Class scheduling to maximize participant satisfaction. *interfaces*, **25**, 30–41.

Santos, H. G., Ochi, L. S., and Souza, M. J. (2005). A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. *J. Exp. Algorithmics*, **10**, 2.9.

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

Schniederjans, M. J. and Kim, G. C. (1987). A goal programming model to optimize departmental preference in course assignments. *Computers and Operations Research*, **14**, 87–96.

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

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

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

Socha, K., Knowles, J., and Sampels, M. (2002). A max-min ant system for the university timetabling problem. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of the Third International Workshop on Ant Algorithms (ANTS 2002)*, volume 2463, pages 1–13. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany.

Stallaert, J. (1997). Automated timetabling improves course scheduling at ucla. *Interfaces*, **27**, 67–81.

Stamatopoulos, P., Viglas, E., and Karaboyas, S. (1998). Nearly optimum timetable construction through clp and intelligent search. *International Journal on Artificial Intelligence Tools*, **7**(4), 415–442.

Steuer, R. E. (2003). *Multiple criteria optimization: theory, computation and application.* Wiley.

Tamiz, Jones, and Romero (1998). Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operational Research*, **111**, 569–581.

Tillett, P. I. (1975). An operations research approach to the assignment of teachers to courses. *Socio-Economic Planning Sciences*, **9**(3-4), 101–104.

Tripathy, A. (1984). School timetabling – A case in large binary integer linear programming. *Management Sci.*, **30**, 1473–1489.

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

Tsuchiya, K. and Takefuji, Y. (1997). A neural network parallel algorithm for meeting schedule problems. *Applied Intelligence*, **7**(3), 205–213.

Tuyttens, D., Teghem, J., Fortemps, P., and Nieuwenhuyze, K. V. (2000). Performance of the MOSA method for the bicriteria assignment problem. *Journal of Heuristics*, **6**(3), 295–310.

van Hoeve, W.-J. (2000). *Towards the Integration of Constraint Logic Programming and Mathematical Programming*. Master's thesis, University of Amsterdam.

van Hoeve, W.-J. (2001). The all_different constraint: A survey. Submitted manuscript. Available from http://www.cwi.nl/wjvh/papers/alldiff.pdf, 2001.

van Hoeve, W.-J. (2005). *Operations Research Techniques in Constraint Programming*. Ph.D. thesis, University of Amsterdam.

Winters, W. K. (1971). A scheduling algorithm for a computer assisted registration system. *Commun. ACM*, **14**(3), 166–171.

Wood, J. and Whitaker, D. (1998). Student centred school timetabling. *The Journal of the Operational Research Society*, **49**(11), 1146–1152.

Appendix A

Appendix

A.1 Datasets

Table A.1 gives an overview of the four major datasets we used throughout this thesis. All datasets are collected from a building of a university in Sydney, Australia. 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. 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 courses to fit without splitting. For `Tut`, the main characteristic of this data-set is the

Data-set Name:	`Lec`	`Wksp`	`Tut`	`Sem`	`Tut-trim`
Spacetype:	**Lec**	**Workshop**	**Tutorial**	**Seminar**	**Tutorial**
num. of modules:	608	1077	2088	3711	620
num. of rooms:	20	16	184	88	47
timeslots number:	50	48	46	46	50
Seat-Hours, courses:	69,983	86,140	290,839	440,131	87,678
Seat-Hours, rooms:	188,244	39,408	163,500	176,318	41,350

Table A.1: The five data-sets 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.

small capacity of rooms and their uniformity, e.g. most rooms have sizes in the range 8-20, enforcing a group size is therefore trivial in this case. The full data-set, `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 data-set, `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.

A.2 Dataset description

In this section, we provide an executive description of the main datasets used in this study, with main aim of providing better insight to the size and property of instances used to run experiments. In the `Lec` dataset, for example, We have 20 rooms, and each has 50 timeslots. This gives a total of 1000 timeslots, whereas the lecture courses only have 608 events. Also, the total seat-hours demand from the lecture courses is 69983 whereas the total supply from the rooms is 202650. Hence, in the initial data set, the lectures are substantially under-subscribed, in the sense that the total demand for seat-hours and timeslots from the courses is much smaller than the supply of seat-hours and timeslots from the rooms. In order to explore a wider range of these supply-to-demand ratios, we have opted against creating more courses, as it would make the problems unnecessarily large. The options of reducing rooms or reducing timeslots are similar in that they reduce the available seat-hours. Eliminating rooms requires a decision of which ones to remove, and it is hard to know what counts as a fair reduction, especially as we suspect that it is the distribution of room and course sizes

Module ID	Size	Timeslots	Department	Faculty
FC6328	90	3	Management	Business
FC6349	220	3	Law	Arts
FC6429	110	3	Law	Arts
FC6435	210	2	Management	Business
FC6445	210	4	Management	Business

TABLE A.2: Module profile and format used in experiments

that is the most important, and so do not want to change it accidentally (and this is also why we do not attempt to use a random generator for instances). So, instead, we uniformly reduce timeslots for all rooms. Hence, we create Lecture Room problem instances, LR(T), with the timeslots per room reduced to T. In the original data T=50, but we also studied T=10,18, and 30,as described in **chap.** 3. The case T=18 is the smallest T in which the `seat-hours` demand could potentially be still be met by the rooms. For the other datasets, `Wksp`, `Tut`, (data in table A.1) where splitting was mostly involved, more freedom was left in defining the different events. Since splitting is dynamic, the event profile will change as the search goes on and different groups get formed to fit the available space. The room space details are mostly formed of non-partitionable rooms with sizes ranging from 20 to 100 seats and with the same timeslot structure of the lecture datasets.

A.3 Event Profile

Table A.2 provides a snapshot of the module profiles used in most experiments. For example, module `FC6328` has 90 students enroled, and is taught in 3 timeslots per week. The module is offered by the `Management` of the `Business` school. Generally module sizes vary between 40-50 students up to 450 students, and have different number of classes.

Room ID	Capacity	Timeslots	Spacetype	Department	School
RO121	300	45	Lecture	Finance	Business
RO111	235	30	Lecture	Law	Arts
RO124	200	45	LEC	Law	Arts
RO201T	25	40	Tutorial	Management	Business
RO222T	15	45	Tut	Law	Arts
RO222W	35	45	Workshop	Finance	Business
RO282W	30	45	Workshop	Law	Arts

TABLE A.3: Room profiles used in experiments

A.4 Room Profile

Table A.3 shows a snapshot[1] of the room profiles used in most experiments. For example, room `RO121` has a capacity of 300 students, and is available on 45 timeslots per week. The spacetype is a lecture theater, and is located in the management department of the Business school.

[1]Data is available from `http://www.cs.nott.ac.uk/čbb/TSA`

A.5 Opl Mod Files