Educational Timetabling

Jeffrey H. Kingston

Abstract. This chapter is an introduction to the problems of timetabling educational institutions such as high schools and universities. These are large problems with multiple sources of NP-completeness, for which robust solvers do not yet exist, although steady progress is being made. This chapter presents the three main problems found in the literature: high school timetabling, university examination timetabling, and university course timetabling. It also examines some major subproblems of these problems: student sectioning, single student timetabling, and room assignment. This chapter also shows how real-world instances of these problems, with their many constraints, can be modelled in full detail, using a case study in high school timetabling as an example.

1 Introduction

Educational timetabling is not a single problem. For each kind of timetable needed by each kind of institution there is a separate problem with a separate literature. These literatures are too large to survey comprehensively within the limits of a book chapter, so only a selection, including recent survey papers, is referenced here. Schaerf (1999) is a good general survey.

Solving a real instance of an educational timetabling problem (an instance taken without simplification from an actual institution) by hand can take weeks of tedious and error-prone work by an expert. Hand-generated timetables are still common, although automated or semi-automated methods are making inroads. For example, the traditional way to timetable students in North American universities, which is to publish lists of course sections and times and expect the students to create their own timetables and sign up for the sections they want, is giving way to a semi-automated method, in which the number of sections of each course and the times they run may still be decided by hand, but the students' timetables are generated automatically.

Jeffrey H. Kingston
School of Information Technologies, University of Sydney, Australia
e-mail: jeff@it.usyd.edu.au

A.Ş. Etaner-Uyar et al. (eds.), *Automated Scheduling and Planning*,
Studies in Computational Intelligence 505,
DOI: 10.1007/978-3-642-39304-4_4,

The lead in educational timetabling has always been given by researchers who are trying to solve real problems from real institutions. This practical orientation has informed the selection of topics for this chapter.

2 Educational Timetabling Problems

The educational timetabling literature mainly studies problems found in *high schools* (schools for older children) and universities. High schools need to timetable their normal activities once per year, or sometimes more often. This is the *high school timetabling problem* (Sect. 4). Universities need to timetable their normal activities once per semester. This is called the *university course timetabling problem* (Sect. 6), to distinguish it from the other main university problem, the *(university) examination timetabling problem* (Sect. 5): the timetabling of examinations after the end of semester. There are other kinds of institutions and problems, but educational timetabling, as it appears in the literature, is essentially about these three problems.

When the number of students enrolled in a high school or university course is large, the course may need to be broken into *sections*: copies of the course, each with its own time, room, and teacher. Each student enrolled in the course must then be assigned to one of its sections. If these assignments are made early in the timetabling process, the result is the *student sectioning* problem (Sect. 7), which aims to assign students to sections so as to facilitate the assignment of times to sections later, by minimizing the number of pairs of sections that share at least one student.

A *phase* is one part of a solver's work, carried out more or less independently of its other phases. Student sectioning is one example of a phase. Other commonly encountered examples are *single student timetabling* (Sect. 8), the creation of a timetable for one student after courses are broken into sections and the sections are assigned times, and *room assignment* (Sect. 9), the assignment of suitable rooms to events after the events' times are fixed. It is often necessary to divide a solve process into phases when faced with large, real instances of timetabling problems, even though it usually rules out all hope of finding a globally optimal solution. This makes individual phases worthy subjects of study in their own right.

All these problems are concerned with assigning times and *resources* (students, teachers, rooms, and so on) to events so as to avoid *clashes* (cases where a resource attends two events at the same time) and violations of various other constraints, such as unavailable times for resources, or restrictions on when events may occur. Some times and resources may be preassigned; others are left open to the solver to assign. Informally, a *timetabling problem* is any problem that fits this description.

Several problems outside the scope of this chapter are timetabling problems by this definition. Nurse rostering is one example. The events are the shifts. The events' times are preassigned, and resources (nurses in this case) must be assigned to them. Another example is sport competition timetabling. The events are the matches, to which times must be assigned. Their resources, largely preassigned, are the teams and venues. These problems are always treated separately, however, and rightly so, because their constraints, other than the avoid-clashes constraint, are very different.

Many timetabling problems can be proved to be NP-complete using a reduction from graph colouring due to Welsh and Powell (1967). For each node of the graph to be coloured, create one event of duration 1. For each edge, create one resource preassigned to the two events corresponding to the edge's endpoints. Then assigning a minimal number of colours to the nodes so that no two adjacent nodes have the same colour is equivalent to assigning a minimal number of times to the events so that no resources have clashes.

The inverse of this construction is the *clash graph*, a widely used conceptual aid. It has one node for each event. An edge joins each pair of events, weighted by the number of resources the two events have in common (Figure 1). Colouring

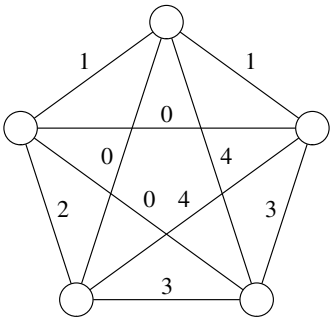

Fig. 1 A clash graph. Each node represents one event; each edge is weighted by the number of resources the events at its endpoints have in common.

this graph to minimize the total weight of edges that join nodes of the same colour is equivalent to assigning a time to each event which minimizes the number of clashes.

It would be a mistake to consider educational timetabling as a branch of graph colouring, however. Timetabling problems have many kinds of constraints. Each may be a source of NP-completeness in its own right, and much of the difficulty lies in handling all of them together. For example, ensuring that students have breaks between examinations is a travelling salesman problem (Sect. 5), balancing teacher workloads is bin packing, and so on (Cooper and Kingston, 1996).

3 Educational Timetabling Models

For most of its history, educational timetabling research has been very fragmented. Each research group has used its own definitions of the problems, and its own data in its own format. There has been little exchange of data, except for one set of instances of the examination timetabling problem.

What is primarily needed to break down these barriers is for researchers to reach consensus on a model, or format, in which instances and solutions can be expressed, including unambiguous rules for calculating the cost of solutions. The terms 'model' and 'format' are roughly interchangeable; 'model' emphasizes the ideas, 'format' emphasizes the concrete syntax that realizes those ideas.

Pure algorithmic problems, such as graph colouring and the travelling salesman problem, are easy to model. Real timetabling problems have many details which vary from institution to institution, and modelling them is a daunting problem—so much so, that it is the largest obstacle to progress in many cases.

Collaborative work on modelling began with a discussion session at the first Practice and Theory of Automated Timetabling (PATAT) conference (Cumming and Paechter, 1995). The work has been carried on continuously since then, primarily within the PATAT conferences. There have been many arguments and some wrong turnings, but the fog is lifting, and data exchange is now becoming common.

One issue has been whether the problems should be modelled in full detail, or simplified to highlight their essence and reduce the implementation burden for solvers. Researchers strive to improve on previous work, and one way to improve is to work with more realistic data, so this issue is resolving itself as time passes.

A second issue has been the choice of level of abstraction. An example of a very abstract format is the input language of an integer programming package. Input in this format allows many kinds of constraints to be expressed, but it is too general to permit the use of solution methods specific to timetabling. Successful formats instead offer a long but finite list of concrete (timetabling-specific) constraint types. Modelling each timetabling problem separately, rather than using a single model for all of them, also makes for concreteness, and has turned out to be best, if only because the work needed to reach a consensus is less, and researchers are more likely to take an interest when their own problem is discussed specifically.

Some generalizations are natural and desirable, however. For example, a model which treats teachers, students, and rooms separately must define a 'no clashes' constraint for each kind of entity. Generalizing to *resources*, which are entities that attend events and may represent individual students, groups of students, teachers, rooms, or anything else (but not times), simplifies the model. Allowing arbitrary sets of times to be defined and named, rather than, for example, just the days and weeks (and similarly for resources), is another useful generalization, as is recognizing each source of cost in a solution as a violation of some kind of constraint. The value of a good generalization is often underestimated: it simplifies solvers as well as models.

The oldest well-known educational timetabling model, and the most successful as measured by the amount of data sharing effected, is the one used by the Toronto data set, which contains 13 real instances of the examination timetabling problem collected by Carter et al. (1996). In recent years it has been criticised for being too simple: it does not model rooms, and its constraints are implicit and so have fixed weights. It is discussed in more detail in Sect. 5.

Another landmark is a format created for nurse rostering in 2005. It models many more constraints than the Toronto data set does, and it models them explicitly, which allows individual instances to choose to include them or not, and to vary their weights. It also uses XML, which is verbose but has the great advantage of being clear and definite. At the time of writing, 20 instances were available in the current version of the format, collected from researchers in 13 countries (Curtois, 2012).

As a case study in modelling educational timetabling problems, the remainder of this section presents a model of the high school timetabling problem called XHSTT (Post, 2012a), developed recently by a group of high school timetabling researchers. The model, which was influenced by the nurse rostering model just described, was refined over several years and tested against real instances. It offers 15 types of constraints, has been used to model about 30 real instances from 10 countries so far, and has achieved widespread acceptance within the high school timetabling research community. The following description, written by this author, is from Post (2012b). Syntactic details are omitted; they may be found online (Kingston, 2009).

An XHSTT file is an XML file containing one *archive*, which consists of a set of instances of the high school timetabling problem, plus any number of *solution groups*. A solution group is a set of solutions to some or all of the archive's instances, typically produced by one solver. There may be several solutions to one instance in one solution group, for example solutions produced using different random seeds.

Each instance has four parts. The first part defines the instance's *times*, that is, the individual intervals of time, of unknown duration, during which events run. Taken in chronological order these times form a sequence called the instance's *cycle*, which is usually one week. Arbitrary sets of times, called *time groups*, may be defined, such as the Monday times or the afternoon times. A *day* is a time group holding the times of one day, and a *week* is a time group holding the times of one week. To assist display software, some time groups may be labelled as days or weeks.

The second part defines the instance's *resources*: the entities that attend events. The resources are partitioned into *resource types*. The usual resource types are a *Teachers* type whose resources represent teachers, a *Rooms* type of rooms, a *Classes* type of *classes* (sets of students who attend the same events), and a *Students* type of individual students. However, an instance may define any number of resource types. Arbitrary sets of resources of the same type, called *resource groups*, may be defined, such as the set of Science laboratories, the set of senior classes, and so on.

The third part defines the instance's *events*: meetings between resources. An event contains a *duration* (a positive integer), a *time*, and any number of *resources* (sometimes called *event resources*). The meaning is that the resources are occupied attending the event for *duration* consecutive times starting at *time*. The duration is a fixed constant. The time may be preassigned or left open to the solver to assign. Each resource may also be preassigned or left open to the solver to assign, although the type of resource to assign is fixed. Arbitrary sets of events, called *event groups*, may be defined. A *course* is an event group representing the events in which a particular class studies a particular subject. Some event groups may be labelled as courses.

For example, suppose class *7A* meets teacher *Smith* in a Science laboratory for two consecutive times. This is represented by one event with duration 2, an open time, and three resources: one preassigned *Classes* resource *7A*, one preassigned *Teachers* resource *Smith*, and one open *Rooms* resource. Later, a constraint will specify that this room should be selected from the *ScienceLaboratories* resource group, and define the penalty imposed on solutions that do not satisfy that constraint.

If class *7A* meets for Science several times each week, several events would be created and placed in an event group labelled as a course. However, it is common in

high school timetabling for the total duration of the events of a course to be fixed, but for the way in which that duration is broken into events to be flexible. For example, class *7A* might need to meet for Science for a total duration of 6 times per week, in events of duration 1 or 2, with at least one event of duration 2 during which the students carry out experiments. One acceptable outcome would be five *sub-events*, as these fragments are called, of durations 2, 1, 1, 1, and 1. Another would be three sub-events, of durations 2, 2, and 2. This is modelled in XHSTT by giving a single event of duration 6. Later, constraints specify the ways in which this event may be split into sub-events, and define the penalty imposed on solutions that do not satisfy those constraints.

The last part of an instance contains any number of *constraints*, representing conditions that an ideal solution would satisfy. At present there are 15 types of constraints, stating that events should be assigned times, prohibiting clashes, and so on. The full list appears in Table 1.

Table 1 The 15 types of XHSTT constraints, with informal explanations of their meaning

Name	Meaning
Assign Resource constraint	Event resource should be assigned a resource
Assign Time constraint	Event should be assigned a time
Split Events constraint	Event should split into a constrained number of sub-events
Distribute Split Events constraint	Event should split into sub-events of constrained durations
Prefer Resources constraint	Event resource assignment should come from resource group
Prefer Times constraint	Event time assignment should come from time group
Avoid Split Assignments constraint	Set of event resources should be assigned the same resource
Spread Events constraint	Set of events should be spread evenly through the cycle
Link Events constraint	Set of events should be assigned the same time
Avoid Clashes constraint	Resource's timetable should not have clashes
Avoid Unavailable Times constraint	Resource should not be busy at unavailable times
Limit Idle Times constraint	Resource's timetable should not have idle times
Cluster Busy Times constraint	Resource should be busy on a limited number of days
Limit Busy Times constraint	Resource should be busy a limited number of times each day
Limit Workload constraint	Resource's total workload should be limited

Each type of constraint has its own specific attributes. For example, a Prefer Times constraint lists the events whose time it constrains, and the preferred times for those events, while a Link Events constraint lists sets of events which should be assigned the same time. Each constraint also has attributes common to all constraints, including a Boolean value saying whether the constraint is hard or soft, and an integer weight.

Traditionally, a *hard constraint* is one that must be satisfied if the timetable is to be used at all, although in practice a few violations of hard constraints are often acceptable, since the school can overcome them by undocumented means (moving a class to after school hours, assigning the deputy principal to a class with no teacher, and so on). A *soft constaint*, on the other hand, is a constraint that is violated quite routinely, although the total cost of those violations should be minimized.

As stated above, solutions are stored separately from instances, in solution groups within the archive file. A solution is a set of sub-events, each containing a duration, a time assignment, and some resource assignments. A solution's *infeasibility value* is the sum over the hard constraints of the number of violations of the constraint times its weight. Its *objective value* is similar, but using the soft constraints. One solution is better than another if it has a smaller infeasibility value, or an equal infeasibility value and a smaller objective value. A web site (Kingston, 2009) has been created which calculates the infeasibility and objective values of the solutions of an archive, and displays comparative tables, lists of violations, and so on.

It used to be said that real instances of timetabling problems required too many types of constraints for complete modelling to be possible. The nurse rostering and high school timetabling models disprove this; they show that careful elucidation of constraints, aided by suitable generalizations, can lead to complete models of real instances which are small enough to be usable. As instances appear that require other types of constraints, those constraints can be added gradually.

4 High School Timetabling

In the *high school timetabling problem*, a set of events of arbitrary integer *duration* is given, each of which contains a *time* and some *resources*: students, classes (sets of students who attend the same events, at least for the most part), teachers, and rooms. The time and resources may be preassigned to specific values, or left open to the solver to assign. The meaning is that the resources assigned to the event are occupied for *duration* consecutive times starting at *time*. The problem is to assign the unpreassigned values so as to avoid clashes and satisfy a variety of other constraints, such as those listed in Table 1 (Sect. 3).

When student sectioning (Sect. 7) is needed, it is carried out as a separate initial phase. Accordingly, it is usually considered not to be part of high school timetabling proper. This is one way in which high school timetabling differs from university course timetabling. Another is that the high school problem timetables groups of students (classes) which are usually occupied together for all, or almost all, of the times of the cycle, whereas the university problem timetables individual students whose timetables contain a significant amount of free time. Some clashes are probably inevitable among the thousands of individual university students' timetables, whereas clashes are not acceptable for classes.

A major division exists within high school timetabling instances with respect to teachers. On one side lie schools whose teachers are mostly part-time and tend to be preassigned to specific courses, and the emphasis is on providing timetables for the teachers which require their attendance for a minimal number of days per week and give them few idle times (free times in between busy times) on those days. On the other side lie schools whose teachers are mostly full-time, making it impractical to preassign teachers to specific courses except in the senior years, and the emphasis is on finding a timetable with teacher assignments that assign a qualified teacher to every lesson of every course.

Schmidt et al. (1980) comprehensively surveys the early history of high school timetabling research, including a description of a very basic version of the problem called *class-teacher timetabling*, which can be solved in polynomial time by edge colouring. Appleby et al. (1960), Gotlieb (1962), and De Werra (1971) are examples of papers that were influential in their day. Carter et al. (1997) is a good snapshot of a more recent era. The PATAT 2012 conference (Kjenstad, 2012) contains several high school timetabling papers, most of them stimulated by the Third International Timetabling Competition (Post, 2012b).

The only recent survey is Pillay (2010). It classifies about 40 papers which solve high school timetabling problems. Their methods include (in decreasing order of popularity) evolutionary algorithms, tabu search, integer programming, simulated annealing, and constraint programming. Many papers hybridize several methods. All of these papers pre-date the creation of standard benchmarks (for which see Sect. 3), so any attempt to rank them would be futile. At the time of writing the only paper with any objective claim to eminence is Fonseca et al. (2012), which describes the work that won the Third International Timetabling Competition (Post, 2012b). A more recent on-line version of this survey (Pillay, 2012) lists many more papers.

5 Examination Timetabling

The (university) examination timetabling problem has one event for each course, representing the course's final examination. The durations of the events may differ, although that is often a minor consideration. Each event's resources are the students who attend the course, and possibly a room. The aim is to assign a time to each event, avoiding clashes in the students' examination timetables. Room assignment (Sect. 9) may be required, and it is characteristic to include *proximity constraints*, expressing in some way the undesirability of attending two examinations close together in time. Proximity constraints might prohibit two examinations for one student on one day, for example, or two consecutive examinations ignoring day boundaries.

Insight into proximity constraints can be gained from the unrealistic special case where the number of examinations equals the number of times, and examinations may not occur simultaneously, ruling out all possibility of clashes. (This could arise in practice if the examinations are clustered before they are timetabled.) If there are two examination sessions per day, then minimizing the number of cases where a student attends two examinations in one day is equivalent to finding a maximum matching of minimum weight in the clash graph, for which there is a polynomial time algorithm. Alternatively, ignoring day boundaries and simply minimizing the number of cases where a student attends examinations at two consecutive times is a travelling salesman problem in the clash graph and so is NP-hard (Figure 2).

Despite involving many of the same resources, examination timetabling is much easier to model than university course timetabling. Coming as it does later in the semester, after enrolments have settled, it has not the dynamic character of course timetabling. Student sectioning is not required, even if different examinations are given to different sections, because it has already been done; and room requirements

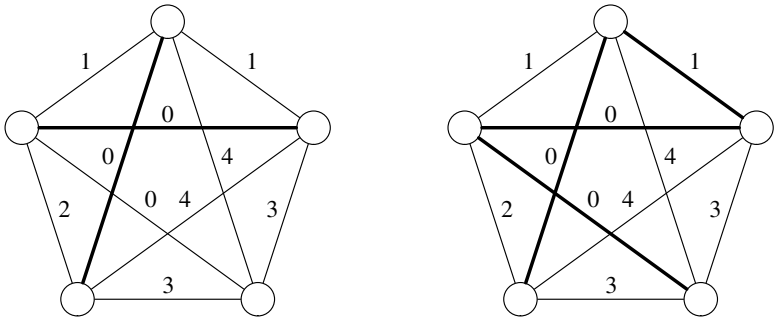

Fig. 2 A clash graph, showing (left) a minimum matching, which defines a pairing of the events which minimizes the number of cases where a resource attends both events of a pair; and (right) a travelling salesman path, which defines an ordering of the events which minimizes the number of cases where a resource attends two consecutive events. Example taken from Kingston (2010).

are usually more uniform, except perhaps for a few practical examinations which take place in laboratories.

Examination timetabling has a relatively long history of use of benchmark data, specifically the 13 real instances of the Toronto data set (Carter et al., 1996), still available and in use. These have quite elaborate proximity constraints, assigning a high penalty for two consecutive examinations, a lower penalty for examinations separated by one time, and so on; but these and the student no-clashes constraints are the only constraints. A more recent data set used by the Second International Timetabling Competition follows a more realistic model (McCollum et al., 2012). A guide to examination timetabling data sets is available online (Qu, 2012).

The leading recent survey is Qu et al. (2009), online at (Qu, 2012), a major work which describes other surveys, models, methods, and data sets, and cites 160 papers and 15 PhD theses. Carter (1986) and Carter et al. (1996) are still worth reading.

6 University Course Timetabling

University course timetabling aims to break university courses into sections, and assign times, students, rooms, and possibly instructors to the sections, subject to constraints like those of high school timetabling, only applied to individual students (each of whom has a distinct timetable) rather than to groups of students.

A *static* timetabling problem is one that is set up once, solved, and used. There may be some exploration of alternative scenarios, but once an acceptable timetable has been found, the work is finished. In contrast, a *dynamic* problem is one whose requirements and solution evolve over time. Most timetabling problems are static, and they are also small enough to be set up by a single person, the local expert.

University course timetabling is different. It is so large that no single person ever understands it all; local experts are scattered across the faculties and departments of

the university, working largely independently of each other. It takes months to set up the problem, and even after semester begins there are continuous changes to the timetables of individual students, and some changes to the events as well (opening and closing sections in response to late changes in student demand).

Traditionally, the tiger was tamed by re-using as much of the previous year's timetable as possible (more than deserved to be, in many cases), and by partitioning the problem. The central administration controlled the main lecture theatre block, the departments controlled the rest. Each faculty made sure that students working entirely within the faculty could get workable timetables, but liaison across faculties was limited to the bare essentials. And if something did not work, a student was simply advised not to do it.

Vestiges of this approach can still be found, but its deficiencies are so glaring (poor room utilization and unhappy students, who are often paying customers these days) that it cannot survive for much longer. At the same time, the presence of a web browser on every desk has resolved the dilemma of a non-partitionable problem whose data are distributed: the departments are required to send their data via the web to the centre, which owns all the resources and does all the timetabling.

So the problem as it stands today is to timetable the entire university, not one department or faculty, including delivering an individual timetable to every student. Most changes after the start of semester can be handled by single student timetabling (Sect. 8), so the focus is on finding a good timetable before semester begins.

Two general approaches to university course timetabling may be distinguished. Emphasis may be placed on ensuring that certain sets of courses can be taken in combination, because students need them to satisfy the degree rules. Such sets of courses are called *curricula*, and this approach to the problem is called *curriculum-based university course timetabling*. Although ultimately each student must receive an individual timetable, in its pure form the curriculum-based approach does not utilize enrolment data for individual students.

In universities where students have large-group lectures and small-group tutorials and laboratories, there is an important sub-problem: assigning times and rooms to the lectures, given some basic information about what combinations of courses the students are likely to choose. Curriculum-based timetabling addresses this kind of problem, and serves as a model of what it may be worthwhile to do before the dynamic phase of university course timetabling begins.

Declaring a set of events to be a curriculum amounts to saying that a number of students will be taking those events in combination. In high school timetabling the same declaration is made by placing a preassigned class resource into the events. This relationship between curriculum-based timetabling and high school timetabling has been exploited in a few papers, such as Nurmi and Kyngäs (2008).

The alternative to curriculum-based timetabling is *enrolment-based university course timetabling*, in which the enrolment data for individual students are used to determine which courses should be able to be taken in combination.

The two approaches are not mutually exclusive. Student enrolment data often becomes available fairly late, in which case curriculum-based timetabling may be

used early to lay out the basic structure of the timetable (such as the times of large-group lectures), while enrolment-based timetabling is used later to fine-tune it.

When constructing a timetable with sections based on student enrolment data, a basic dilemma emerges: whether to assign students to sections before or after assigning times to sections. Neither alternative is fully satisfactory, so, in practice, implementations of both approaches always include some way of reconsidering the first phase after completing the second.

Assigning times first has the advantage that students can then be assigned to sections using single student timetabling (Sect. 8), which is known to work well. If the result is poor, the time assignment can be adjusted.

A semi-automatic version of this method is used at the author's university. Initial values for the number of sections of each course, and their times, are chosen manually, based on curricula, history, and incomplete student enrolment data. Then a dynamic process of refinement begins. As student enrolment information improves, dummy runs of single student timetabling applied to each student (but not published to the students) are carried out at the centre. Results are distributed to departmental coordinators, who respond by opening and closing sections as enrolment numbers become clearer, and moving sections left underfilled by single student timetabling to other and hopefully better times, subject to room and teacher availability.

It is possible that a fully automated timetabling system could be built by this process of repeated time assignment then testing by single student timetabling. Even if single student timetabling is highly optimized and virtually instantaneous for one student, it will still take several seconds to timetable every student, which is too slow to support an extensive search through the space of time assignments. So there would probably be time to re-timetable only those students directly affected by each time adjustment. But this method does not seem to have ever been tried, except by Aubin et al. (1989), who used it to timetable 'a large high school in Montreal'.

Assigning students first leads naturally to a three-phase method (Carter, 2001). First, assign students to sections before times are assigned, aiming to minimize the number of pairs of sections with students in common. This is the *student sectioning* problem, discussed in detail in Sect. 7. Second, assign times to the sections. This is a graph colouring problem similar to examination timetabling without proximity constraints. Finally, make one pass over the entire student list, re-timetabling each student using single student timetabling. (Experience at the author's university has shown that two or even three passes help to even out section numbers.)

Only two systems which solve the full university course timetabling problem have been published in detail. The first is the system described in Carter (2001), which has been in use since 1985. It is specific to one university, although of course the ideas are portable. It performs enrolment-based timetabling using the three-phase method just described.

The second system, UniTime (2012), is free, open-source, and not specific to one university—a combination of features apparently unavailable elsewhere. It offers both curriculum-based and enrolment-based timetabling, following Carter (2001) for the latter, and is in use at several universities, although development continues. A long list of papers is given on its web site (UniTime, 2012); only a selection can be

cited here. Murray et al. (2002) is the original work. Müller et al. (2004) considers the problem of finding minimally perturbed timetables when circumstances change after the timetable has been published, an important problem whose study has barely begun. Murray et al. (2007) and Murray et al. (2010) present the mature system.

The dynamic nature of the university course timetabling problem is an obstacle to designing a realistic data model for it. No data sets are available for the full problem, although partial data are available. The UniTime web site offers data sets for several sub-problems: departmental problems and so on. Several timetabling competitions have targeted university course timetabling problems. The most recent of these is the Second International Timetabling Competition (McCollum, 2007), within which Track 2 is enrolment-based, and Track 3 is curriculum-based. Both tracks offer only drastically simplified instances: they are static, they model one faculty rather than the whole university, their events have equal duration, and every course has just one section. The data are still available and are the focus of many papers. No comprehensive survey of these papers is known to the author.

7 Student Sectioning

As explained earlier, a course may break into *sections*: copies of it, each with its own time, room, and teacher. Each student enrolled in the course must be assigned to one of its sections. The *student sectioning* problem asks for an assignment of students to sections which is likely to work well when times are assigned to the sections later, typically by minimizing the number of pairs of sections that have at least one student in common.

Some formulations of the problem also ask for a clustering of the sections, such that if the sections in each cluster run at the same time, but different clusters run at different times, then no students have clashes. This is a natural extension, since a good clustering proves that a student sectioning is successful.

Student sectioning arises in university course timetabling (Sect. 6), when the choice is made to assign students to sections before assigning times to sections.

Essentially the same situation arises in high schools. High school instances often have complex events called *electives*: sets of courses that the school decides to run simultaneously. Each student chooses one course from each elective. Electives are usually determined by surveying the students to find out which courses they intend to take, and ensuring that there are enough sections of each course to accommodate the students who wish to take it, and that the sections of popular combinations of courses lie in different electives. This problem of defining the electives is a student sectioning problem, including the clustering extension.

The student sectioning literature is very fragmentary. There is no survey, and one must search for discussions of student sectioning in papers on university course timetabling and high school timetabling. To add to the confusion, the term ‘student sectioning’ is sometimes used for the full university course timetabling problem, for which it is clearly a misnomer.

Carter (2001) is the seminal paper for student sectioning. It has several pages of practical discussion of the problem. For each course, each enrolled student is made into a node of a graph. An edge joins each pair of nodes, with a weight between 0 and 1 determined by how similar the two students' selection of courses is: 0 means identical, 1 means disjoint. The students are then grouped into sections by a standard graph clustering algorithm. This heuristic method produces relatively few pairs of sections with students in common. Murray et al. (2007) follow Carter (2001) in the student sectioning phase of their university course timetabling algorithm.

The student sectioning problem is smaller in high schools than in universities, which may explain why the few high school sectioning papers known to the author use more ambitious methods: de Haan et al. (2007) use branch and bound, while Kristiansen and Stidsen (2012) use adaptive large scale neighbourhood search.

8 Single Student Timetabling

The *single student timetabling* problem, usually encountered as a phase of university course timetabling (Sect. 6), asks for a timetable for a single student after courses are broken into sections and the sections are assigned times. Its first priority is to find a clash-free timetable for the student; its second is to assign the student to less full sections (say, by minimizing the total enrolment of the sections chosen), so that as it is run for many students, the sections are kept approximately equally full.

If all sections have duration 1, this is a weighted bipartite matching problem. One set of nodes represents the courses, the other represents the times of the week. An edge is drawn from a course to a time whenever a section of that course occurs at that time, weighted by the current enrolment of that section.

In practice, sections of different courses may have different durations, and there may be additional constraints, such as that the assignment of sections of two courses be correlated in some way (ordered in time, for example). Even so, real instances can be solved to optimality very quickly using a tree search. Each node of the tree represents one course, and each downward edge out of that node represents the assignment of the student to a section of that course. Edges that produce clashes with higher edges are not followed.

In the senior years, where course enrolments are lower and the number of sections is correspondingly fewer, the tree search just described may be adequate as it stands. But junior courses may have many sections. A course with 500 students that breaks into sections of 20 students each will have about 25 sections, and if a student takes several such courses the search will need to be optimized.

One obvious optimization is to assign the courses with the fewest sections first. Then courses with only one section are (in effect) preassigned, and there are more choices at the lower levels of the tree.

A second optimization, sometimes called *intelligent backtracking*, is as follows. Suppose the search is at some node of the tree, and that every section of that node's course has been tried and has failed owing to a clash with sections assigned higher in the tree. A simple tree search would return to the parent of that node and continue

with its next alternative. But if the parent was not involved in any of the clashes, that is futile: the same clashes will recur. Instead of backtracking to the parent, intelligent backtracking backtracks to the closest ancestor involved in a clash.

A third optimization focuses on minimizing the total enrolment of the sections. First, the cost of assigning a section is changed from the current enrolment of that section to the amount by which that current enrolment exceeds the current enrolment of the least full section of its course. For example, enrolling a student in a least full section costs 0. Then branch and bound is used to terminate a search path when its cost equals or exceeds the cost of the best solution found so far; and if a complete solution is found whose cost is 0, the entire search is terminated early. Combined with sorting the sections so that the least full ones are tried first, this optimization is effective at reducing the size of the search tree when there are many solutions.

It seems likely that many universities would have such solvers, given their need to timetable thousands of students, and to re-timetable them when their enrolments change. Laporte et al. (1986) describes one, only without intelligent backtracking. Another, employing all these optimizations, has been used routinely at the author's university for many years (unpublished). It timetables a single student on demand virtually instantaneously, producing virtually equal section enrolment numbers. So single student timetabling is a solved problem.

9 Room Assignment

The *room assignment problem* asks for an assignment of rooms to events after the events' times are fixed. Each event has its own room requirements, such as for a specialist room (a Science laboratory, a lecture theatre, and so on), or for a room capable of holding at least a certain number of students. This problem occurs in all kinds of educational timetabling.

Carter et al. (1992) is a fascinating compendium of results on room assignment. It observes that room assignment is exactly list colouring of interval graphs, a well-known NP-complete problem, and shows that it remains NP-complete even when the cycle contains only two times, a remarkable result.

Each room may be tested against each event's room requirements, and in this way any combination of room requirements for an event may be reduced to a set of suitable rooms for that event before solving begins. Among suitable rooms some may be more suitable than others, in which case the outcome of the testing is an integer rating of each room's suitability for each event.

When all events have duration 1, and all rooms lie within easy walking distance of each other, the instance of the room assignment problem for the events assigned a given time t is independent of the instances at other times. It can be solved to optimality by finding a maximum matching in the bipartite graph whose nodes are the events assigned time t and the rooms available at time t, with an edge joining an event node to a room node whenever the room is suitable for the event (Figure 3). If rooms have integer ratings, the edges are weighted by the ratings.

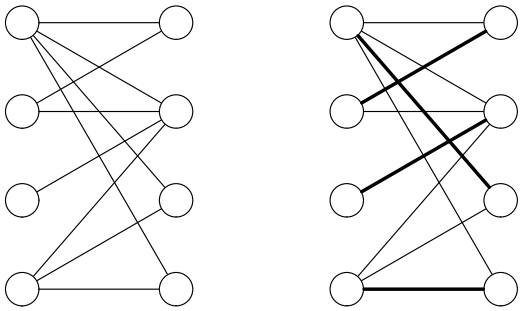

Fig. 3 A bipartite graph (left), and the same graph showing a maximum matching (right). Each left-hand node represents an event assigned a particular time t and demanding one room, each right-hand node represents a room, and edges indicate which rooms are suited to which events.

Kingston (2012) reports perfect results from a room assignment algorithm which assigns rooms using a constructive heuristic followed by adjustment using ejection chains, while maintaining the existence of an unweighted maximum matching of optimal size at each time as an invariant throughout the solve. The algorithm runs in polynomial time and takes less than one second on real instances. In practice, then, room assignment, like single student timetabling, is a solved problem.

Room assignment is usually a phase of a larger problem. Kingston (2012) gives techniques for efficiently maintaining the invariant while times are being assigned to events. In this way, a nearly exact guarantee can be given that room assignment will succeed, without actually assigning any rooms.

Can other resources be assigned in the same way as rooms? Students and classes are usually preassigned, leaving nothing to do. Teachers, too, are often preassigned. When they are not, there is usually a requirement, called *teacher stability* or the *avoid split assignments constraint*, that the teacher assigned to the events of one course be the same. Kingston (2012) investigates this variant with some success, although it is significantly more difficult in practice than room assignment.

10 Conclusion

Educational timetabling has been an active area of research for over 50 years. While there is no sign that provably optimal solutions to large, real instances will be found any time soon, the more practical goal of quickly and reliably finding timetables that are preferred to manually produced ones is in sight. More realistic models and data sets are needed in some areas, and more powerful solving techniques are needed in others, but steady progress continues to be made on both fronts, the transfer of research methods into commercial software is growing, and there is every reason to believe that success is not far off.

Acknowledgements. The author thanks Keith Murray, Ender Özcan, and Gerhard Post for their comments on earlier drafts of this chapter.

References

Appleby, J.S., Blake, D.V., Newman, E.A.: Techniques for producing school timetables on a computer and their application to other scheduling problems. The Computer Journal 3, 237–245 (1960)

Aubin, J., Ferland, J.A.: A large scale timetabling problem. Computers and Operations Research 16, 67–77 (1989)

Carter, M.W.: A survey of practical applications of examination timetabling algorithms. Operations Research 34, 193–202 (1986)

Carter, M.W., Tovey, C.A.: When is the classroom assignment problem hard? Operations Research 40, S28–S39 (1992)

Carter, M.W., Laporte, G.: Recent developments in practical examination timetabling. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)

Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: algorithmic strategies and applications. Journal of Operational Research Society 47, 373–383 (1996)

Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)

Carter, M.W.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 64–81. Springer, Heidelberg (2001)

Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 283–295. Springer, Heidelberg (1996)

Cumming, A., Paechter, B.: Standard formats for timetabling data. In: Unpublished Discussion Session at the First International Conference on the Practice and Theory of Automated Timetabling, PATAT 1995, Edinburgh (August 1995)

Curtois, T.: Employee scheduling benchmark data sets, `http://www.cs.nott.ac.uk/~tec/NRP/` (Cited September 15, 2012)

De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, and results. In: Proceedings, 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal (August 2008)

de Haan, P., Landman, R., Post, G., Ruizenaar, H.: A case study for timetabling in a Dutch secondary school. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 267–279. Springer, Heidelberg (2007)

De Werra, D.: Construction of school timetables by flow methods. INFOR—Canadian Journal of Operational Research and Information Processing 9, 12–22 (1971)

Fonseca, G.H.G., Santos, H.G., Toffolo, T.A.M., Brito, S.S., Souza, M.J.F.: A SA-ILS approach for the high school timetabling problem. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway (August 2012)

Gotlieb, C.C.: The construction of class-teacher timetables. In: Popplewell, C.M. (ed.) Information Processing 1962 (Proceedings of the 1962 IFIP Congress), pp. 73–77 (1962)

Kingston, J.H.: The HSEval high school timetable evaluator (2009), http://www.it.usyd.edu/au/~jeff/hseval.cgi (Cited September 15, 2012)

Kingston, J.H.: Timetable construction: the algorithms and complexity perspective. In: Proceedings of the Eighth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010), Belfast, UK (August 2010)

Kingston, J.H.: Resource assignment in high school timetabling. Annals of Operations Research 194, 241 (2012)

Kjenstad, D., Riise, A., Nordlander, T.E., McCollum, B., Burke, E.: In: Proceedings, of the 9th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2012, Son, Norway (August 2012)

Kristiansen, S., Stidsen, T.R.: Adaptive large neighborhood search for student sectioning at Danish high schools. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway (August 2012)

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

McCollum, B.: The Second International Timetabling Competition (ITC 2007), Track 3 (2007), http://www.cs.qub.ac.uk/itc2007 (Cited September 17, 2012)

McCollum, B., McMullan, P., Parkes, A.J., Burke, E.K., Qu, R.: A new model for automated examination timetabling. Annals of Operations Research 194, 291–315 (2012)

Müller, T., Rudová, H., Barták, R.: Minimal perturbation problem in course timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 126–146. Springer, Heidelberg (2005)

Müller, T., Rudová, H.: Real-life curriculum-based timetabling. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2012, Son, Norway (August 2012)

Murray, K., Rudová, H.: University course timetabling with soft constraints. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 310–328. Springer, Heidelberg (2003)

Murray, K., Müller, T., Rudová, H.: Modeling and solution of a complex university course timetabling problem. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 189–209. Springer, Heidelberg (2007)

Murray, K., Müller, T.: Comprehensive approach to student sectioning. Annals of Operations Research 181, 249–269 (2007)

Nurmi, K., Kyngäs, J.: A conversion scheme for turning a curriculum-based timetabling problem into a school timetabling problem. In: Proceedings, of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal (August 2008)

Pillay, N.: An overview of school timetabling. In: Proceedings, of the 8th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2010, Belfast, UK, pp. 321–335 (August 2010)

Pillay, N.: Classification of school timetabling research, http://titan.cs.unp.ac.za/~nelishiap/st/classification.htm (Cited September 15, 2012)

Post, G.: Benchmarking project for (high) school timetabling, http://www.utwente.nl/ctit/hstt/ (Cited September 15, 2012)

Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. In: Proceedings, of the 9th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2012, Son, Norway (August 2012)

Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12, 55–89 (2009)

Qu, R.: Benchmark data sets in exam timetabling, http://www.cs.nott.ac.uk/~rxq/data.htm (Cited September 15, 2012)

Schaerf, S.: A survey of automated timetabling. Articifial Intelligence Review 13, 87–127 (1999)

Schmidt, G., Ströhlein, T.: Timetable construction–an annotated bibliography. The Computer Journal 23, 307–316 (1980)

UniTime: a comprehensive university timetabling system, http://www.unitime.org/ (Cited September 18, 2012)

Welsh, D.J.A., Powell, M.B.: An upper bound for the chromatic number of a graph and its application to a timetabling problem. The Computer Journal 10, 85–86 (1967)