Generalizing Bipartite Edge Colouring to Solve Real Instances of the Timetabling Problem

David J. Abraham and Jeffrey H. Kingston

School of Information Technologies
The University of Sydney 2006
Australia
jeff@it.usyd.edu.au

Abstract. In this paper we introduce a new algorithm for secondary school timetabling, inspired by the classical bipartite graph edge colouring algorithm for basic class-teacher timetabling. We give practical methods for generating large sets of meetings that can be timetabled to run simultaneously, and for building actual timetables based on these sets. We report promising empirical results for one real-world instance of the problem.

1 Introduction

This paper is concerned with the problem of constructing timetables for secondary schools, in which groups of students meet with teachers in rooms at times chosen so that no student group, teacher, or room attends two or more meetings simultaneously.

One fundamental requirement separates secondary school timetabling from university timetabling: every student is required to be in class during every teaching period. This makes it infeasible for every student to have an individual timetable; instead, the students are placed in groups, and it is these groups that are timetabled, not individual students.

In secondary schools known to the authors, the principal technique used for offering students some choice is the *elective*. Suppose that there are 180 students in one year (age cohort), enough to form six separate classes of 30 students each. Late in the previous year the students would be offered a list of subject areas (e.g. French, German, Biology, History, Economics, Business) and required to select exactly one. Depending on their responses, the school management decides how many classes of each type to offer. These classes then run simultaneously the following year.

Electives give rise to meetings containing many resources. Our example elective would contain one student group, six teachers, and six rooms, all constrained to be used at the same set of times. A student's week is filled completely, with a mixture of compulsory subjects and electives. For Mathematics the students are typically grouped by ability, and this requires all the Mathematics classes for a given year to run simultaneously, creating something similar to an elective except that all the classes within it are Mathematics. For the other compulsory subjects the groups of students within a year may often be timetabled independently.

It is not possible to preassign teachers to large elective meetings, since the resulting timetabling problem would be hopelessly over-constrained. Only a few teacher slots (typically in the most senior classes) are preassigned; the rest are assigned as part of the

timetabling process, after times have been assigned. Naturally, this teacher assignment phase must assign English teachers to English classes, Economics teachers to Economics classes, etc., so each teacher slot must record the category of teacher it needs. These categories or *teacher types* are not disjoint: some teachers teach several subjects, others are qualified to teach junior subjects but not senior, and so on. Rooms must be assigned too, and they also have categories: Science laboratories, Music studios, ordinary classrooms, and so on.

Away from electives the meetings may be much smaller, the natural minimum being a meeting containing three resources: one preassigned student group, one teacher, and one room. Manual high school timetabling is usually accomplished by timetabling the large meetings first, then fitting the small ones around them.

Although soft constraints do exist in this problem, concerned with the even spread of classes through the week, not overloading any teacher on any one day, etc., the problem is dominated by the basic hard constraints already described: finding times and qualified teachers which avoid clashes and therefore also keep every student group occupied for every time of the week.

Earlier work on this problem [2, 4] has been successful in assigning times to meetings in such a way that, at each time, resources are sufficient to fill all the resource slots of meetings scheduled for that time. This would be a complete solution except for one problem.

The problem is the *teacher constancy requirement*, which states that, when a meeting contains multiple times, any teacher assigned to it must attend for all of those times. We do not want, say, an English teacher slot to be filled by Smith for the first two times, Jones for the next three, and Robinson for the last time. Violations of this requirement, known as *split assignments*, have often been unacceptably frequent when solving the problem using the cited earlier methods. The problem does not arise with student group slots, since they are preassigned, and is usually considered unimportant for room slots, except when times are adjacent.

If every meeting contained the same number of times, say k, then it would be easy to achieve teacher constancy. Replace the k time slots in each meeting with just one time slot, solve the resulting problem, then duplicate each meeting k-fold. This is equivalent to the method, often used in North American universities, of defining certain patterns of times in advance (e.g. Mondays 9-10 plus Wednesdays 9-10 plus Fridays 9-10) and requiring all meetings to choose one pattern rather than a set of times. Unfortunately, in Australian secondary schools (and elsewhere) the number of times in each meeting depends on the importance of the subject matter. English and Mathematics each require 6 time slots; other subjects may have 6, 5, 4, 3, 2, or 1 time slot each. The 'time patterns' approach cannot be applied.

It is desirable to assign times to meetings in such a way that pairs of meetings either overlap completely in time or not at all. We say that timetables with this property have good *time coherence* [4]. Good time coherence will minimize the number of pairs of clashing meetings, and should minimize the forces which push teachers into split assignments.

In this paper we present a new algorithm for constructing secondary school timetables. Inspired by the classical edge colouring algorithm for class-teacher timetabling, but designed to handle the general problem, this new algorithm tries to schedule as many

meetings as possible into the first time in the week, then as many of the remaining meetings as possible into the second, and so on. This approach seems to be comparable with earlier work in its ability to find suitable times for all time slots, but, unlike earlier work, offers much better prospects for making highly time coherent timetables, as Section 2 will explain.

In addition to proposing a new method of constructing timetables, this paper contains an initial empirical study which shows that the new method is promising in practice. As the reader will find, there are several points where different means could be used to achieve the same ends, and we are only at the beginning of the task of exploring these alternatives.

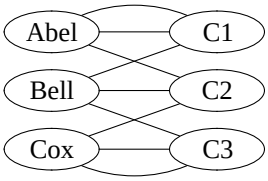
Section 2 introduces the new algorithm, and Section 3 explains how we can test whether a set of meetings can run simultaneously, before assigning resources to the meetings. Sections 4 and 5 explore the two main phases of the algorithm in detail. Section 6 presents our results so far, and Section 7 contains our conclusions and plans for further work. A more detailed exposition of our work appears in [1].

2 Generalizing the classical edge colouring algorithm

Our new algorithm is inspired by the classical edge colouring algorithm for bipartite graphs, attributed to König [9], and apparently first applied to class-teacher timetabling by Csima [6] (see also [5, 7, 10]). We begin with a brief recapitulation of that algorithm, then proceed to its generalization.

The edge colouring algorithm applies when each meeting contains one preassigned teacher, one preassigned student group, and any number of time slots. Times are to be assigned to these slots so that no teacher or student group has a clash; this is the only constraint.

Build a bipartite graph by creating one left-hand node for each teacher, one right-hand node for each student group, and one edge for each time slot. Each time slot lies in a meeting containing one teacher and one student group, and the corresponding edge connects the vertices corresponding to these two resources. For example, the graph

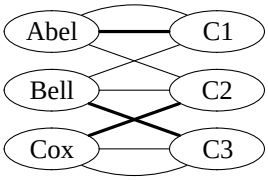


corresponds to three teachers (Abel, Bell, and Cox) teaching three student groups (C1, C2, and C3). Abel takes C1 twice, and Cox takes C3 twice.

An *edge colouring* is an assignment of colours, or equivalently integers 1, 2, 3, … to the edges so that no two edges adjacent to any vertex have the same colour. If we interpret these colours as times, then colouring the edges corresponds to finding a timetable, and the rule prohibiting two edges with the same colour from touching any one vertex is equivalent to the timetabling requirement that no resource be required to attend two meetings at the same time.

There is an obvious lower bound on the number of colours needed to solve this prob-

lem: the maximum vertex degree. Edge colouring theory proves that this bound can always be achieved, by a polynomial time algorithm based on repeatedly finding maximum matchings. For example, here is our example graph with a maximum matching in bold:



These edges are assigned the first colour, which corresponds with assigning the first available time to the corresponding time slots, then deleted. Because the edges form a matching, no two of them are adjacent, so no teacher or student group can have a clash at this time. A new matching is found and the second colour assigned to its edges, and so on until no edges are left. For minimality it turns out to be necessary to restrict the matching algorithm at each step to the vertices of maximum degree and the edges and vertices adjacent to them.

Edge colouring is not used in practical timetabling because it models too restrictive a version of the problem. In practice, meetings may have many more than two resources, and the resources are not necessarily preassigned. Conversely, some of the time slots may be preassigned. These generalizations make the problem NP-complete [3, 7]. Nevertheless, if we interpret the edge colouring algorithm in timetabling terms we obtain an interesting idea for an algorithm for the general problem:

> *Timetable as many meetings as possible into the first time of the week, concentrating on those meetings that are hardest to timetable. Delete the assigned time slots, delete any meetings that now have no time slots, and repeat on the second time of the week, then the third, and so on.*

This is the algorithm we study in this paper.

To see why this algorithm is likely to deliver the time-coherence promised in Section 1, consider the set of meetings chosen during the first step to occupy the first time of the week. If all of these meetings contain more than one time slot, this exact same set of meetings may be re-used for the second time. In general we cannot expect that all of the meetings chosen will have exactly the same number of time slots, but by encouraging the algorithm to choose sets of meetings with a similar number of time slots, and taking care over what to do with leftover fragments of longer meetings, it should be possible to produce a very time-coherent timetable.

Clearly, the success of this algorithm will partly depend on whether large sets of meetings able to run simultaneously can be found. We have pursued an approach in which all such sets are computed in an initial phase, and this is the subject of Sections 3 and 4. After that, a second phase selects a combination of sets from the first phase that together cover all the meetings. This selection phase is the subject of Section 5.

3 Testing sets of meetings for compatibility

Our first task is to find an efficient test which can tell us whether or not a set of meetings S is *compatible*, that is, whether or not its meetings can run simultaneously.

Meetings contain time slots and resource slots, which may be unconstrained, somewhat constrained, or completely constrained (i.e. preassigned).

Apart from a few preassignments, time slots in school timetabling problems are effectively unconstrained. There is often a soft constraint that the times of a meeting be spread through the week in some desirable pattern, but this does not affect any meeting's ability to run at any particular time.

We restructure the meetings of S to ensure that each meeting either contains exactly one preassigned time slot, or else it contains one or more unconstrained time slots. We do this by repeatedly finding any preassigned time slot s which is not the only time slot in its meeting m, creating a new meeting containing s as its only time slot and copies of all the resource slots of m, and deleting s from m. We then merge meetings whose time slots contain the same preassigned time; there is no need to distinguish meetings that are constrained to run simultaneously. Ignoring resource constraints for the moment, a set of such restructured meetings is compatible if no two parts of what was originally one meeting are involved, and the meetings' time slots do not include preassignments to two different times. (We do not currently have a complete implementation of this part of our test, but the number of preassigned times in our data is so small that it does not matter for present purposes.)

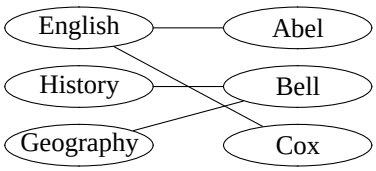
Resource constraints are the main problem. We need to determine whether the supply of resources is sufficient to fill all the resource slots of the set of meetings S. This problem has been solved before [2]; we briefly recapitulate that solution here.

We assume that all resources are available, unless the set of meetings contains a preassigned time (there can be at most one), in which case we leave out resources known to be unavailable at that time.

In practice we always find that each resource slot is constrained independently of the others to be filled by an element of some fixed subset of the available resources: it may require an English teacher, or a Science laboratory, and so on. Preassignment is included in this model: the fixed subset contains just one element.

Build a bipartite graph whose left nodes are all the resource slots in the set of meetings S, and whose right nodes are all the available resources. Connect each slot node to every resource able to fill that slot (these may form an arbitrary subset of the available resources). The meetings are compatible if a matching exists which touches every resource slot, for this matching represents an assignment of resources to all the slots.

For example, suppose we are testing three meetings for compatibility: English, History, and Geography. Considering teachers alone, the bipartite graph might look like this:



There are enough teachers, there is a qualified teacher for every slot, but there is no matching and these meetings are not compatible.

Although our test for compatibility is reasonably efficient as described, we will be calling it thousands of times, so we have implemented some optimizations. We check that no two slots are preassigned with the same resource, and that for every defined category (e.g. English teacher, Science laboratory) there are at least as many qualified resources as resource slots. Only if the meetings pass these quick tests do we take the time to build the bipartite graph and carry out the full test.

It is also important for efficiency that we can take a known compatible set of meetings for which a matching has been created and stored, as well as the totals needed to implement our quick tests, and efficiently test whether one extra meeting can be added without losing compatibility. The quick tests just add to the totals they keep; the standard bipartite algorithm can build on the existing matching, doing only the relatively small amount of work needed to add in nodes corresponding to the resource slots from the extra meeting. We update the matching as we add the slots, making failed matchings terminate faster and leaving less to undo. Deletion of the extra meeting's nodes after the test, if required, is also efficient.

4 Generating compatible sets of meetings

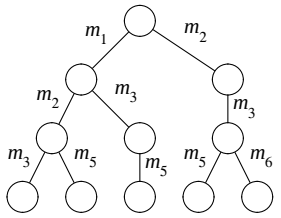
A compatible set of meetings can be timetabled at any time during the week, except in the rare cases where it contains a preassigned time. Because of this, we decided to try to exhaustively generate all compatible sets of meetings in an initial phase, reasoning that we could select from this collection over and over again to build the timetable.

Finding large compatible sets of meetings is an NP-hard problem similar to finding independent sets in a graph. However, unlike independent sets, a set of meetings can be pairwise compatible yet not compatible overall.

Our algorithm for generating all compatible sets of meetings is a dynamic programming algorithm based on the matroid recurrence

Every subset of a compatible set of meetings is compatible.

We generate all compatible sets of meetings of size (number of meetings) 1, then all compatible sets of meetings of size 2, etc. Our dynamic programming table holds the set of all sets of compatible meetings of size i, structured as a tree:

Sets stored	*Tree-structured table*
$\{m_1, m_2, m_3\}$ $\{m_1, m_2, m_5\}$ $\{m_1, m_3, m_5\}$ $\{m_2, m_3, m_5\}$ $\{m_2, m_3, m_6\}$	

When extending a set $\{m_{j_1}, m_{j_2}, \ldots, m_{j_i}\}$ to a larger set $\{m_{j_1}, m_{j_2}, \ldots, m_{j_i}, m_{j_{i+1}}\}$, by our

recurrence we only need to test for compatibility those meetings $m_{j_{i+1}}$ for which all i-element subsets of $\{m_{j_1}, m_{j_2}, \ldots, m_{j_i}, m_{j_{i+1}}\}$ are in the table (i.e. are compatible), and the tree structure helps to find the set of all such $m_{j_{i+1}}$ efficiently. Whenever a new set is added, we delete all its proper subsets, ensuring that only maximal compatible sets of meetings are in the table at the end. See [1] for more details.

Despite careful optimization of space and time, we have found that in practice there are too many maximal sets of compatible meetings for us to be able to compute them all, so we have been forced to leave out meetings with three or fewer resource slots. Like manual timetablers, we plan to pack these small meetings around the large ones in a final phase (Section 6).

5 Selecting compatible sets of meetings

At the end of the phase just described, we have a table containing all maximal compatible sets of meetings, excluding the small meetings which we had to leave out. If there are T times in the week, we now need to select T of these sets, ensuring that each meeting m appears in the selected sets as many times as there are time slots in m. Repeated selection of the same set is allowed, and indeed preferred since it leads to time coherence.

This is a standard set covering problem and many methods are available for solving it. We use a tree search with a greedy heuristic [8] for selecting the set of meetings to try next. We limit the branching factor at deep levels in the tree, and use forward checks to prune futile subtrees.

Our greedy heuristic is to prefer sets of meetings with larger numbers of resource slots. The number of meetings in the set is irrelevant at this stage: what matters is to utilize as many resources as possible.

Two optimizations well known in set covering are implemented. If a meeting m appears in only one set, that set is selected immediately. And if every set containing meeting m_1 also contains m_2, we make sure that any set we select for covering m_2 also covers m_1, since we must cover m_1 eventually and at that time we will definitely be covering m_2 as well, by assumption.

When we select a set, we reduce by one the number of time slots of each of its meetings that remain to be assigned. When this number reaches 0 for some meeting m, we delete m from all remaining compatible sets of meetings, and if this causes any set S to become a proper subset of another, we delete S. Of course, if we backtrack we have to undo these changes. These operations are expensive and data structures to optimize them are required, but they pay off in reducing the amount of data handled at the deeper levels of the tree where most time is spent, and in permitting forward checks, which would otherwise not be based on current information.

Some valuable forward checks are implemented. If there are M meetings remaining to be timetabled and T times remaining to be used, then one of the remaining sets of meetings must contain at least $\lceil M/T \rceil$ meetings – if not, we backtrack immediately. Similarly, if the remaining resource slots of a particular type (e.g. Science laboratory slots) total S times' worth of that type of slot, and there are T times remaining to be used, then one of the remaining sets of meetings must contain at least $\lceil S/T \rceil$ of those slots.

6 Results

We have tested our new algorithm on BGHS98, an instance of the secondary school timetabling problem taken without simplification from a school in Sydney, Australia. BGHS98 contains 40 times, 150 resources (30 student groups, 56 teachers, and 64 rooms), and 208 meetings. The number of time slots per meeting varies between 1 and 6, averaging just under 3. On average, we need to schedule 15.4 meetings into each time slot, and the meetings in each time slot must contain 102.6 resources on average. This is considerably less than the 150 resources total, but some parts of the resource load are very tight, notably the student groups (which must attend at every time) and the scarce Science laboratories (whose utilization must be virtually 100%).

Our tests were run on standard hardware. When we omitted meetings with three or fewer resource slots, only 57 of the 208 meetings remained, but these contained 75% of the resource slots. Finding all compatible sets of meetings among the 57 large meetings took only a few seconds, although adding just a few of the omitted meetings caused the program to fail to terminate (not surprisingly, since these small meetings scarcely constrain each other and so combine in exponentially many ways). Finding a cover for the 57 meetings also took just a few seconds. All the parts of our algorithm described in Section 5 contributed to this efficiency, in the sense that leaving any one out produced no useful result in any reasonable time. For example, without backtracking our greedy heuristic with forward checking and optimizations produced a timetable requiring 42 times, two more than the number of times available.

Assignment of the remaining small meetings in a final phase was not as easy as we had hoped it would be. Many methods are of course possible at this stage. Our first attempt, a simple heuristic assignment, was unsuccessful, and a subsequent hill-climber made almost no difference. So we tried meta-matching [2] and this assigned times to all but five meetings. The leftovers were Science meetings unable to find Science laboratories at the limited times that their student group resources were available.

7 Conclusion

We have presented a new algorithm for constructing secondary school timetables, based on the classical edge colouring algorithm for class-teacher timetabling.

Our success in timetabling all but five small meetings in a few seconds on standard hardware is very promising. However, this is work in progress and there is still a lot of work to do.

We need to do more work on the assignment of small meetings. A more elaborate final phase, with a backtracking element for example, might be sufficient. If not, we will need to integrate the smaller meetings into the main assignment phase, perhaps by greedily augmenting the selected compatible sets with compatible small meetings. This would allow the smaller meetings to influence the forward checking and backtracking, and provide a natural way to obtain time coherence among the small meetings.

Then will come the task of evaluating our solutions for time coherence, and tuning our algorithm to enhance it. We may need to encourage the selection of compatible sets of meetings with similar numbers of time slots. Finally, we will need to try out some resource assignment algorithms and verify that improved time coherence really does lead to fewer split assignments, as we expect.

References

[1] David J. Abraham. The high school timetable construction problem. Honours thesis (2002), School of Information Technologies, The University of Sydney.

[2] Tim B. Cooper and Jeffrey H. Kingston. The solution of real instances of the timetabling problem. *The Computer Journal* **36**, 645–653 (1993).

[3] Tim B. Cooper and Jeffrey H. Kingston. The complexity of timetable construction problems. In *First International Conference on the Practice and Theory of Automated Timetabling*. Napier University, Edinburgh, UK, 1995.

[4] Tim B. Cooper and Jeffrey H. Kingston. A program for constructing high school timetables. In *First International Conference on the Practice and Theory of Automated Timetabling*. Napier University, Edinburgh, UK, 1995.

[5] J. Csima and C. C. Gotlieb. Tests on a computer method for constructing school timetables. *Communications of the ACM* **7**, 160–163 (1964).

[6] J. Csima. *Investigations on a Time-Table Problem*. Ph.D. thesis, School of Graduate Studies, University of Toronto, 1965.

[7] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing* **5**, 691–703 (1976).

[8] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences* **9**, 256–278 (1974).

[9] D. König. Graphok es alkalmazasuk a determinansok es a halmazok elmeletere. *Mathematikai es Termeszettudomanyi* **34**, 104–119 (1916).

[10] G. Schmidt and T. Ströhlein. Timetable construction—an annotated bibliography. *The Computer Journal* **23**, 307–316 (1980).