On Using Graph Coloring to Create University Timetables with Essential and Preferential Conditions

TIMOTHY A. REDL
University of Houston-Downtown
Department of Computer and Mathematical Sciences
One Main Street, Suite S705, Houston, Texas
UNITED STATES OF AMERICA
REDLT@UHD.EDU

Abstract: We develop and describe a mathematical and computational model for creating university timetables using techniques and heuristics of graph coloring that incorporates the satisfaction of both essential and preferential timetabling conditions. The model involves creating a conflict graph from assembled university course data, properly coloring the conflict graph, and transforming this coloring into a conflict-free timetable of courses. From this timetable, one can then assign the courses to classrooms based on room capacity and availability. Once a course timetable is constructed, a conflict-free schedule of final examinations for the courses can quite easily be obtained.

Key–Words: university timetabling, graph coloring

1 Introduction

Timetabling is the scheduling of a set of related events in a minimal block of time such that no resource is required simultaneously by more than one event. In university timetabling, the resources involved, which we assume may be required by no more than one course at any particular time, are instructors, classrooms, and students. University timetabling is a practical application of graph coloring. The minimum coloring problem and the timetabling problem have been classified as NP-hard problems in the general case. This means that it is unlikely that it will be possible to find fast (i.e., polynomial-time) algorithms to solve these problems. In order to find optimal solutions to such NP-hard problems, it is usually necessary to consider all possible solutions to choose the best one. However, solutions to large-scale practical problems, including timetabling, are often desired and needed much more quickly than any exhaustive search algorithm could hope to provide. As a result, numerous heuristics have been developed for such problems which produce "near-optimal" satisfactory solutions in much less time.

University course timetabling problems involve pairwise restrictions on the courses being scheduled. That is, there exist restrictions on which courses can be scheduled simultaneously. Restrictions involved in creating a timetable of courses may be divided into two categories, which we call essential and preferential timetabling conditions.

1.1 Essential and Preferential Conditions

Essential timetabling conditions are conditions or constraints that must be satisfied in order to produce a legal or feasible timetable. A few examples of essential timetabling conditions include the following:

- Two or more courses taught by the same instructor cannot be scheduled for the same time slot [7, among others].
- Two or more courses that require the same classroom cannot be scheduled for the same time slot [7, among others].
- Equivalently, two or more courses scheduled for the same time slot cannot be assigned the same classroom [5, among others].
- Instructors must be available at the times their courses are scheduled [4].
- Each course must be scheduled for exactly one time slot and one room, to remain constant throughout the scheduling period [3].
- Some courses are required to meet a certain fixed number of times per week (e.g., 3 times per week, 2 times per week, or 1 time per week) [5].
- Each course must be scheduled in an available classroom that can accommodate its size [1, among others].

- Specific outlined room requirements/type for a particular course, if any (e.g., a laboratory workspace, or a computer workstation with projection screen) should be taken into account [5, among others].
- Any courses marked as "prescheduled" should be scheduled to the specified time [3, among others].
- Courses which enroll the same set of students (for example, a chemistry lecture course and its accompanying laboratory course) cannot be scheduled for the same time slot [5].

Preferential timetabling conditions are additional conditions or constraints that need not necessarily be satisfied to produce a legal or legitimate timetable, but if satisfied may very well produce a more "acceptable" timetable for students and/or faculty members. These conditions are requests that should be fulfilled, if possible. Preferential timetabling conditions are often reasonable, but it may or may not be possible to fulfill each of them in addition to all of the essential conditions. A few examples of preferential timetabling conditions might include the following:

- An instructor may have a time preference as to when a course is scheduled to meet (e.g., in the morning, afternoon, or evening).
- An instructor may have a specific room request for a course, beyond the scope of the outlined room requirements specified above (this is an example illustrating the sometimes "fine line" between an essential and preferential timetabling condition).
- It might be preferable to assign each course to a classroom that is located in or close to the building in which that course's department or school is based, and/or close to the office of the course's instructor [3, among others].
- Most courses should not be scheduled in the evening, but rather during the morning and afternoon (i.e., during "normal business hours"), unless an evening time slot is specifically requested for a particular course [1, among others].
- Try to schedule sufficiently large courses at times that "minimize pain for students" [8].
- Classrooms should be just large enough to hold the courses in them, in order to eliminate the presence of unused empty space [3].
- Minimize the number of classrooms used or needed when scheduling the courses [1].
- Attempt for an even distribution of courses among time slots; this will aid in room assignment and also allow for maximum utilization of resources [2, among others].

In addition to the above as well as other essential and preferential timetabling conditions, we also incorporate the following observations in our model:

- Many university courses meeting 3 times per week are usually scheduled to meet on Mondays, Wednesdays and Fridays for periods of 1 hour each (or 50 minutes, allowing 10 minutes of travel time for students between classes), for a total of 3 hours per week.
- Many university courses meeting 2 times per week are usually scheduled to meet on Tuesdays and Thursdays for periods of $1\frac{1}{2}$ hours each (or 80 minutes, allowing 10 minutes of travel time for students between classes), for a total of 3 hours per week.
- Many university courses meeting 1 time per week are usually scheduled to meet for a period of 3 hours (or 170 minutes, allowing 10 minutes of travel time for students between classes) on either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays (i.e., there are no Saturday or Sunday courses).

2 Assembling Course Data

Before the timetabling of university courses for a particular semester can take place, we must first assemble a collection of university course data for that semester, to serve as input to our problem instance. Each course to be scheduled constitutes a data entry, containing the required or optional information below. Such information is expected to be supplied by either the instructor of the course, the academic department or school to which the course belongs, or some other appropriate source.

Required Data Fields:

- **COURSEID**: The "name" of the course, identified by academic department and course number (for example, MATH101). If a course has multiple sections, each section will be entered as a different course.

- **INSTRUCTOR**: The name of the instructor of the course, identified by “LastName,FirstName,MiddleInitial” (for example, “Redl,TimothyA.”).

Optional Data Fields:

- **NUMDAYS**: A preference for the number of days each week that the course is to meet at its scheduled time (i.e., “3”, “2”, or “1”). A “–” denotes “no preference”.
- **DAYS**: A preference for the days of the week that the course will meet. Monday = “M”, Tuesday = “T”, Wednesday = “W”, Thursday = “R”, and Friday = “F”. Courses meeting 3 times per week meet on Mondays, Wednesdays, and Fridays; if NUMDAYS = “3”, then DAYS = “MWF”. Courses meeting 2 times per week will meet on Tuesdays and Thursdays; if NUMDAYS = “2”, then DAYS = “TR”. Courses meeting 1 time per week can meet on one of either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays; if NUMDAYS = “1”, then DAYS = “M”, “T”, “W”, “R” or “F”. There are no courses scheduled on Saturdays and Sundays. A “–” denotes “no preference”.
- **TIMEOFDAY**: A preference for the general time of day (i.e., morning, afternoon, or evening) that the course will meet. A “1” denotes “morning”, a “2” denotes “afternoon”, and a “3” denotes “evening”. Also, a “4” denotes “not in the evening”, and a “–” denotes “no preference”.
- **ROOMTYPE**: A preference for a particular type of room in which the course will be taught, as dictated by the type of course (e.g., “lecture”, “seminar”, or “laboratory”).
- **ROOM**: A preference for a specific room in which the course will be taught, identified by building name and room number. Requests for a particular *room* may often not be honored, since only one course can occupy a given room at any one time. A request for a particular *room type* is probably more realistic, given there are often several rooms of the same type at a university.
- **CLASSMAXSIZE**: This allows one to specify the maximum enrollment of the course, in number of students (e.g., “30”). This information will likely be helpful in later assigning courses to available classrooms. No course may be assigned to a room which has a smaller capacity than the maximum course enrollment.

3 Creating a Course Conflict Graph

Once we have gathered the necessary university course data for a given semester, we construct a course timetabling conflict graph G reflecting the given data. Vertices represent courses, and an edge connects each pair of “incompatible” or “conflicting” courses. Suppose we have a set of n courses $\{c_1, c_2, \ldots, c_n\}$ to be scheduled. Each course c_i will be represented by exactly one vertex v_i in G. Therefore G contains n vertices, and $V(G) = \{v_1, v_2, \ldots, v_n\}$. Each vertex in G is first classified as belonging to exactly one of ten “groups”, depending on its corresponding course’s NUMDAYS, DAYS, and TIMEOFDAY entry:

Group 1: Courses with NUMDAYS = “3”, DAYS = “MWF”, and TIMEOFDAY = “1”
Group 2: Courses with NUMDAYS = “3”, DAYS = “MWF”, and TIMEOFDAY = “2”
Group 3: Courses with NUMDAYS = “3”, DAYS = “MWF”, and TIMEOFDAY = “3”
Group 4: Courses with NUMDAYS = “3”, DAYS = “MWF”, and TIMEOFDAY = “4” or “–”
Group 5: Courses with NUMDAYS = “2”, DAYS = “TR”, and TIMEOFDAY = “1”
Group 6: Courses with NUMDAYS = “2”, DAYS = “TR”, and TIMEOFDAY = “2”
Group 7: Courses with NUMDAYS = “2”, DAYS = “TR”, and TIMEOFDAY = “3”
Group 8: Courses with NUMDAYS = “2”, DAYS = “TR”, and TIMEOFDAY = “4” or “–”
Group 9: Courses with NUMDAYS = “–”
Group 10: All remaining courses, namely those with NUMDAYS = “1”

Next, edges are added to the graph. Edges indicate pairs of courses that we either cannot or do not want to schedule at the same time. Edges reflecting essential timetabling conditions *must* be added to the graph. Additional edges may then be added, to reflect preferential timetabling conditions.

If the INSTRUCTOR of course c_i and course c_j are the same, then we *must* add an edge between vertex v_i and vertex v_j, since courses c_i and c_j cannot be scheduled for the same time slot. An instructor teaching k courses will induce a clique of order k in the graph, since each of the k courses he or she teaches must be scheduled for a different time slot.

If the ROOM requested for course c_i and course c_j are the same, then in order to allow both courses to be assigned that room, we add an edge between vertex v_i and vertex v_j, so that courses c_i and c_j will not be scheduled for the same time slot. If a particular room is requested by r different courses, and we wish to allow all r of those courses to be assigned that room, then we introduce a clique of order r to our graph, so that each of the r courses is scheduled for a different time slot. We may not be able to honor all or even most requests for a particular *room*. However, there are often several rooms of the same *room type* at a university. If a requested room is unavailable, there may be another available room of similar type and size that would suit just as well.

We can also add an edge between:
(1) each vertex in Group 1 and Group 5, and
(2) each vertex in Group 1 and Group 6, and
(3) each vertex in Group 1 and Group 7, and
(4) each vertex in Group 1 and Group 8, and
(5) each vertex in Group 2 and Group 5, and
(6) each vertex in Group 2 and Group 6, and
(7) each vertex in Group 2 and Group 7, and
(8) each vertex in Group 2 and Group 8, and
(9) each vertex in Group 3 and Group 5, and
(10) each vertex in Group 3 and Group 6, and
(11) each vertex in Group 3 and Group 7, and
(12) each vertex in Group 3 and Group 8, and
(13) each vertex in Group 4 and Group 5, and
(14) each vertex in Group 4 and Group 6, and
(15) each vertex in Group 4 and Group 7, and
(16) each vertex in Group 4 and Group 8.

Lines (1) through (16) above reflect both NUMDAYS and DAYS preferences. A course c_i that requests to meet 3 days (MWF) per week will be scheduled for a different time slot than a course c_j that requests to meet 2 days (TR) per week.

We can also add an edge between:
(17) each vertex in Group 1 and Group 2, and
(18) each vertex in Group 1 and Group 3, and
(19) each vertex in Group 2 and Group 3, and
(20) each vertex in Group 5 and Group 6, and
(21) each vertex in Group 5 and Group 7, and
(22) each vertex in Group 6 and Group 7.

Lines (17) through (22) above strictly reflect TIMEOFDAY preferences. Two courses that request to meet on the same days but at different general times of the day will be scheduled for different time slots.

We can also add an edge between:
(23) each vertex in Group 3 and each vertex v_i in Group 4, if TIMEOFDAY of course c_i = "4", and
(24) each vertex in Group 7 and each vertex v_i in Group 8, if TIMEOFDAY of course c_i = "4".

Lines (23) and (24) above specifically reflect "not in the evening" TIMEOFDAY preferences. Courses that request "not in the evening" will not be scheduled for evening time slots.

To reflect TIMEOFDAY preferences for courses represented by vertices in Group 9, we add edges to G as follows. For each vertex v_i in Group 9 corresponding to a course c_i, we add an edge between:
(25) v_i and Groups 2, 3, 6 & 7, if TIMEOFDAY of c_i = "1", or
(26) v_i and Groups 1, 3, 5 & 7, if TIMEOFDAY of c_i = "2", or
(27) v_i and Groups 1, 2, 5 & 6, if TIMEOFDAY of c_i = "3", or
(28) v_i and Groups 3 & 7 if TIMEOFDAY of c_i = "4".

Finally, we reflect DAYS and/or TIMEOFDAY preferences for courses represented by vertices in Group 10 by adding edges to G as follows. For each vertex v_i in Group 10 corresponding to a course c_i, we add an edge between:
(29) v_i and Groups 5, 6, 7 & 8, if DAY of c_i = "M", "W", or "F", or
(30) v_i and Groups 1, 2, 3 & 4, if DAY of c_i = "T" or "R", and
(31) v_i and Groups 2, 3, 6 & 7, if TIMEOFDAY of c_i = "1", or
(32) v_i and Groups 1, 3, 5 & 7, if TIMEOFDAY of c_i = "2", or
(33) v_i and Groups 1, 2, 5 & 6, if TIMEOFDAY of c_i = "3", or
(34) v_i and Groups 3 & 7, if TIMEOFDAY of c_i = "4".

4 Coloring the Conflict Graph

After constructing our course conflict graph, we can then properly color its vertices, and ultimately use that proper vertex coloring to construct a conflict-free timetable of courses. Recall that in a proper vertex coloring of a graph G, a pair of vertices v_i and v_j are colored with different colors if there is an edge between them. Vertices that do not share an edge may be colored with either different colors or the same color.

Our general approach will follow that of the sequential graph coloring algorithm. Sequential graph coloring algorithms are commonly referred to as "greedy algorithms". A greedy graph coloring algorithm examines each vertex of the graph one at a time according to some particular order and tries to color the vertex with one of the colors used so far; that is, it tries to add the vertex to one of the existing color classes. If this is not possible, then a new color class is created and the vertex is assigned the color of that new class. Greedy sequential graph coloring algorithms attempt to properly color a graph using the *minimum* number of colors possible.

Four greedy graph coloring algorithms (SIMPLE-SEARCH GREEDY (SSG), LARGEST-FIRST-SEARCH GREEDY (LFSG), SMALLEST-FIRST-SEARCH GREEDY (SFSG), and RANDOM-SEARCH GREEDY (RSG)) were each implemented with the ability to accommodate each of four different initial orderings of the vertices of the graph: 1) a specific predefined ordering (SO); 2) an ordering by decreasing degree (DD) (i.e., largest-degreed vertices first); 3) an ordering by smallest degree (SD) (i.e., smallest-degreed vertices first); and 4) a purely random ordering of the vertices (RO). These four greedy graph coloring algorithms (SSG, LFSG, SFSG, and RSG), each with four potentially different initial vertex orderings (SO, DD, SD, and RO), resulted in implementations of 16 different variations of the greedy or sequential approach to graph coloring [6].

Our goal is to properly color the vertices of our conflict graph using one of the 16 above greedy graph coloring variations and thereby partition our vertices into independent color classes. These color classes designate sets of courses which can safely be assigned to the same time slot without conflicts. Vertices corresponding to courses which cannot be assigned to the same time slot due to potential conflicts will be colored with different colors in our model.

We propose coloring the vertices of our graph in "Group Order", by first coloring all the vertices in Group 1, then Group 2, then Group 3, then Group 4, then Group 5, then Group 6, then Group 7, then Group 8, then Group 9, and then finally Group 10. Additionally, there are various ways in which we can order the courses/vertices in each group. Within each vertex group we could:

- sort courses *by COURSEID*, (e.g., departments with a larger number of courses to be scheduled in a particular vertex group would be colored before departments with fewer courses), or

- sort courses *by INSTRUCTOR*, (e.g., instructors with greater seniority or more courses to be scheduled in a particular vertex group would be colored before instructors with less seniority or fewer courses to be scheduled, and any courses currently without a known instructor could be colored last), or

- sort courses *in the order in which their preference requests were received by the scheduler*, so that in each vertex group, we "reward" instructors for submitting their course preference requests early, and course vertices are colored on a "first come, first served" basis, or

- sort courses *by (CLASSMAXSIZE)*, so that in each vertex group we color vertices corresponding to larger-sized courses first, followed by those corresponding to smaller-sized courses, or

- sort courses *in random order*, so as not to intentionally "favor" any particular course over another when coloring.

In addition, we could choose to "precolor" particular vertices in our course conflict graph, based on some priority criteria. Such vertices or courses would be preassigned a specific color and/or preassigned a specific time slot prior to the actual coloring and timetabling procedure, in such a way as to satisfy an essential timetabling condition that any courses designated as "prescheduled" should be scheduled to their respective specified time slots. Of course, we must be careful to *properly* precolor vertices corresponding to such courses, so as not to induce any immediate consequential scheduling conflicts!

One of the preferential timetabling conditions described earlier in this paper calls for an attempt to maintain an even distribution of courses among time slots, which should ultimately aid in room assignment and also allow for maximum utilization of university resources. Such a condition is often strongly preferred by both small-sized and large-sized universities. Therefore, for purposes of university course timetabling, it appears that one particular "color searching procedure", namely that in the SMALLEST-FIRST-SEARCH GREEDY algorithm variation, will be the most preferable to use in coloring our conflict graph. During the coloring procedure, SFSG ranks the color classes by the number of vertices currently in them, and searches this ranked list of color classes in order by *ascending size*. Based on this searching heuristic, SFSG attempts to balance the size of the color classes. If color classes correspond to time slots in our timetabling-by-graph-coloring model, then the SFSG color searching procedure should greatly aid in satisfying the above preferential timetabling condition of even distribution of courses among time slots as much as possible.

5 Computational Results

To test the effects of a variety of different color searching procedures as well as the effects of a variety of different initial vertex orderings on graph coloring performance, we conducted computational experiments involving each of the above 16 greedy algorithm variations and the coloring of different conflict graphs G arising from data obtained from Rice University and the University of St. Thomas in Houston, TX. Detailed results of such experiments, along with an in-depth analysis of said results, appear in [6] and are summarized briefly below.

- Rice University, Fall 2000: G has 749 vertices, 147,440 edges; chromatic number $\chi(G) = 11$ (verified by OPLStudio), with computing time for all colorings between 0.03 and 0.09 seconds

- Rice University, Spring 2001: G has 760 vertices, 173,730 edges; chromatic number $\chi(G) = 13$ (verified by OPLStudio), with computing time for all colorings between 0.03 and 0.07 seconds

- University of St. Thomas, Spring 2001: G has 708 vertices, 86,858 edges; chromatic number $\chi(G) = 16$ (verified by OPLStudio), with computing time for all colorings between 0.01 and 0.03 seconds

Our computational tests further confirmed our hypothesis that coloring vertices with the color searching heuristic found in SMALLEST-FIRST-SEARCH GREEDY in order of decreasing degree (i.e., variation SFSG:DD) produces the most balanced and evenly distributed color classes. Similar (and in some cases, more desirable) results were obtained by coloring the vertices according to a specific predefined ordering using coloring variation SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (SFSG:SO) and the initial “Group Order” vertex ordering described above in Section 4.

6 Concluding Remarks

After our course conflict graph G has been properly colored, each independent color class represents a non-overlapping time slot. We thus have an extremely straightforward way of transforming a proper coloring of G to a conflict-free timetable of courses. Each course will be assigned to the time slot designated by the color of its corresponding vertex in G. A σ-colored conflict graph is equivalent to a timetable with σ time slots. If $\chi(G) = k$, then we can transform G to a conflict-free course timetable with k time slots. We may wish to designate particular color classes containing more courses with large CLASSMAXSIZE, or a historically high enrollment, to “less painful” time slots when possible, in efforts to “minimize pain for students” [8].

Once we have constructed our conflict-free timetable of courses, the task remains of assigning each of the courses an appropriate classroom in which to meet. There exist both essential and preferential timetabling conditions with regard to room assignment, some of which were stated in Section 1.1. A number of simple algorithms, loosely based on the concept of “bin packing”, can be used to assign already-timetabled courses to classrooms, after the coloring of a conflict graph has taken place. These algorithms, called FIRST FIT DECREASING room assignment (FFDra) and BEST FIT DECREASING room assignment (BFDra) are described in more detail in [6]. If, for example, we wish to attempt to assign courses to classrooms just large enough to hold them, in order to minimize the presence of unused empty space, we would favor BFDra over FFDra, since BFDra will specifically seek to accomplish this goal.

Finally, after we have successfully constructed a satisfactory and acceptable conflict-free timetable of courses, and each course has been assigned an appropriate time slot at which to meet, and classroom in which to meet *during* the semester, we can then transform this course timetable to a final exam timetable or schedule for the *end* of the semester. A procedure for doing so, along with some variations, is further discussed in [6]. Such variations include attempting to minimize consecutive final exams for students via integer programming, and attempting to pack multiple final exams into a single classroom.

References:

[1] E. K. Burke, D. G. Elliman, and R. Weare, A university timetabling system based on graph coloring and constraint manipulation, *Journal of Research on Computing in Education*, 26, 1993.

[2] P. M. Cavanaugh and C. Tran, Private communication, University of Houston, Houston, TX.

[3] W. Erben and J. Keppler, A genetic algorithm solving a weekly course-timetabling problem, In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, 1995.

[4] C. C. Gotlieb, The construction of class-teacher time-tables, *Proc. IFIP Congress*, 62:73-77, 1963.

[5] S. K. Miner, S. Elmohammed, and H. W. Yau, Optimizing timetabling solutions using graph coloring, 1995 NPAC REU Program, NPAC, Syracuse University, 1995.

[6] T. A. Redl, *A Study of University Timetabling that Blends Graph Coloring with the Satisfaction of Various Essential and Preferential Conditions*, Ph.D. Thesis, Department of Computational and Applied Mathematics, Rice University, May 2004, 159 pages.

[7] M. Trick,
Network resources for coloring a graph. http://mat.gsia.cmu.edu/COLOR/color.html

[8] D. Woods and A. Trenaman, Simultaneous satisfaction of hard and soft timetable constraints for a university department using evolutionary timetabling, Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland, 1999.