A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo

Michael W. Carter,
Mechanical and Industrial Engineering,
University of Toronto,
5 Kings College Road,
Toronto, Ontario, Canada
M5S 3G8
carter@mie.utoronto.ca
http://www.mie.utoronto.ca/staff/profiles/carter.html

Abstract. This paper describes a comprehensive course timetabling and student scheduling system that was developed for the University of Waterloo between 1979 and 1985. The system is based on a "demand-driven" philosophy where students first chose their courses, and the system tries to find the best timetable to maximize the number of satisfied requests. The problem is first decomposed into small manageable sub-problems. Each sub-problem is solved in sequence using a greedy heuristic to assign times to sections, and a Lagrangian relaxation algorithm to assign classrooms. Timetable representatives from each department have interactive access to make final modifications. Finally, each student is individually assigned to the combination of course sections that maximizes timetable satisfaction and balances section sizes. The system has been used successfully for 15 years.

1 Introduction

For the purposes of this paper, course timetabling and student scheduling will be defined as the sequence of problems where each meeting of each course is assigned to a classroom and a time, and students are assigned to sections of each course such that they have no conflicts. At Waterloo, we did not automate the assignment of faculty to course sections; that process is normally done manually in advance.

Although the system was implemented 15 years ago, much of the discussion is still very relevant. In a recent survey of practical course timetabling, Carter and Laporte [6] observed that there were very few papers published that described actual implementations of course timetabling. Aubin and Ferland [1] describe a method that used quadratic assignment formulation for both course timetabling and student sectioning. The program was implemented at a large Montreal High School and two university departments (although they did not provide details on the implementation). Paechter et al. [9] and Rankin [10] describe a genetic course timetabling algorithm that is being used for the Computer Science Department at Napier University. Sampson et al. [11] describe a heuristic that was implemented in the Graduate School of Business at the University of Virginia. The existing university examples were all restricted to a single department or school. There are no published examples of an automated, integrated course timetabling system across the institution. We believe that researchers in practical timetabling can learn a lot from this case.

The University of Waterloo is a comprehensive university with undergraduate and graduate programs in Applied Health Sciences, Arts, Engineering, Environmental Studies, Mathematics and Science. The University offers North America's largest co-operative education program that operates year-round, as well as the more traditional regular program of two consecutive four-month school terms. [12]

In 1985, there were 17 000 undergraduate students and has now grown to close to 20 000. In the Fall term of 1985 (the term we used for our parallel testing and benchmarking) there were 1400 courses offered with about 3000 course sections. (i.e. some courses have multiple sections.)

The system required an estimated 40 person years (1979~1987) to develop. It was designed to run on IBM mainframe equipment, and the database and the programs were all developed in-house. The new system required about 100 new programs to be written and 400 old ones changed. It was implemented between Aug 1985~May 1987. One of the drawbacks of the system is that it has been designed to solve the specific instance of the Waterloo problem; it is not portable, although many of the optimization routines were designed to solve a more general mathematical problem.

In spite of the fact that the system is over 15 years old, we believe that there is still no system that solves the large-scale course timetabling problem with this level of mathematical sophistication.

2 Overview of the System

In North American schools, it is common for students to choose courses from a wide variety of electives. This is particularly true for the humanities and social sciences, but there is often considerable flexibility in upper year science and engineering programs. Waterloo uses a "demand driven" timetabling system: students pre-register for courses from a list of "course offerings" posted in March. The University then constructs a timetable that attempts to minimize potential student conflicts (see Carter and Laporte [6] for a discussion of demand driven timetabling versus master timetabling where the course times are fixed first, and then students pre-register.)

The key milestones in the timetabling/registration cycle (for Fall term) are as follows:

- March: student pre-registration: students submit a list of course requests.
- Early June: initial timetabling: each course is assigned to a day/time slot; initial room assignment: each meeting is assigned to a classroom.
- June~July: on-line timetabling: timetable reps make manual changes.
- August: initial student scheduling: students are assigned to course sections.
- September: on-line scheduling: students revise schedules in a "drop/add" period.

The primary inputs to the process are:

- Courses: list of expected course offerings (with instructors pre-assigned). The departments may also assign preferred meeting times. Historically at Waterloo, 75% of the course offerings are the same as the previous year.

- Student requests: student requests are listed in order of importance with "required" courses first. (In the event that a conflict cannot be resolved, the courses at the end of the list will be dropped.) First year students do not pre-register (until August); so we construct surrogate first year students based on course selection patterns from the previous year. In 1985, just over 50% of the students pre-registered by early June (in time for timetabling).
- Course requests are combined into a "conflict matrix" or "joint demand matrix": the number of students who have requested each pair of courses. Required course conflicts count twice as much as electives.
- Instructors: availability and teaching conflicts.
- Space: we must ensure that there are sufficient rooms of the appropriate size at all times. (room profile)
- Slot system: The University uses a system of standard slot times that are designed to "fit" together nicely. People are not required to use standard slots, but the system introduces a small penalty for courses that do not conform. The list illustrates a few examples of standard slots:

  - 8:30 MWF (3 x 1 hour)
  - 8:30-11:30 T (1 x 3 hours)
  - 8:30-10 TR (2 x 1.5 hours)
  - 8:30 TR (2 x 1 hour)

3 Initial Section Assignments: Homogeneous Sectioning

When courses are offered in multiple sections as they are at Waterloo, it creates a timetabling paradox. Students request a *course*, but timetabling assigns days and times to course *sections*. We cannot assign times to *sections* until we know which students are in each section. But we cannot assign students to *sections* until we know when the sections are timetabled!

Our approach to this problem is called homogenous sectioning. For every course with multiple sections, we group (cluster) the students with *similar* course requests. We construct balanced groups equal to the number of sections. We then assign these groups to sections to minimize the *expected* number of conflicts (based on the preferred times for each section). This is a temporary pre-assignment used only for automated timetabling. After timetabling is complete, we assign students to sections individually to minimize conflicts.

Grouping students based on similar course selections was chosen because it can dramatically *reduce* the number of edges in the conflict matrix. Figure 1 illustrates a simple example. Suppose nine students sign up for Psych 101. Assume three of the students are second year Math, three are first year Psych majors and three are third year Mechanical Engineers. Suppose each student has all of his/her five courses in common with the other students in their group. If we put the Math students in section 1, the Psych majors in section 2 and the Engineers in section 3, then each section will only conflict with four other courses for a total of 30 conflict edges as shown in Figure 1. However, if we mix the students and put one Math student, one Psych major and one Engineer in each section, then all of the Psych 101 sections will conflict with all of the other courses, and this will create a total of 54 conflict edges. When the number of conflict edges is reduced, the quality of the final solution is greatly improved, and it becomes much easier to find conflict-free combinations.

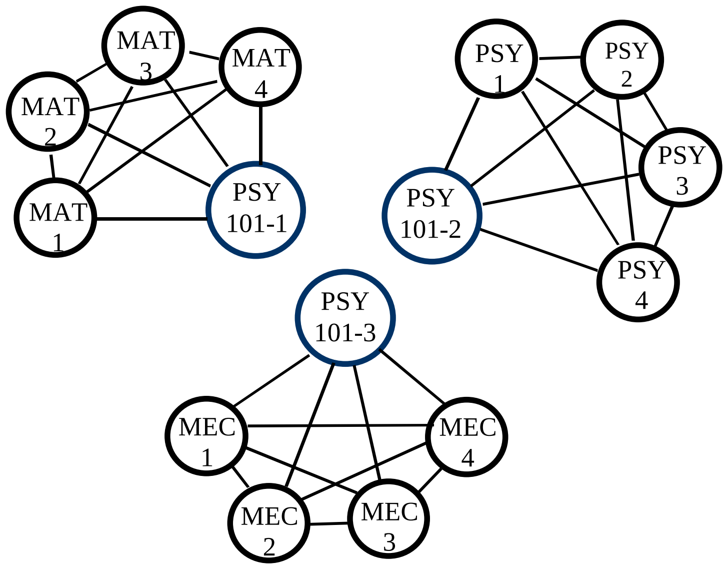

Fig. 1. When the students are assigned to homogeneous sections, the total number of conflict edges is 30. If the students in each program were distributed over the three PSY 101 sections, there would be 54 conflict edges

The algorithm begins by constructing a graph where each vertex represents a student. We draw an edge between each pair of vertices and assign an edge weight that measures the distance between them. The grouping is based on graph clustering where the *distance* between two students varies between 0 and 1 depending on the number of course/section requests that they have in common. Distance 0 means all courses/sections are the same, and 1 means that the two students have nothing in common. The algorithm iteratively combines vertices that are closest to each other without overloading the corresponding section. Whenever two vertices are combined, the new distance to all other vertices is defined as the distance between the two “students” in each pair of vertices of maximum difference. We continue until the number of vertices remaining equals the required number of sections. (As a section “fills up”, it is removed from the problem.)

Consider the example in which eight students have requested a course that will be offered in two sections. These students have also requested other courses as shown in Table 1.

Table 1. Homogeneous grouping example

Student	Other courses			
1	A	B	C	D
2	A	B	C	D
3	A	D	E	F
4	A	B	F	G
5	A	F	G	H
6	C	D	F	G
7	C	F	G	
8	D	E	F	

In Table 2, the *distance* between each pair of students has been computed. In particular, since students 1 and 2 have identical course selections, they can be combined. The next shortest distance between two groups is 1/7 between pairs (3, 8) and (6, 7). Table 3 illustrates the reduced problem with the new distances between sub-groups.

Table 2. Distance between pairs of students

	1	2	3	4	5	6	7	8
1	-	0	$^{4}/_{8}$	$^{4}/_{8}$	$^{6}/_{8}$	$^{4}/_{8}$	$^{5}/_{7}$	$^{5}/_{7}$
2	0	-	$^{4}/_{8}$	$^{4}/_{8}$	$^{6}/_{8}$	$^{4}/_{8}$	$^{5}/_{7}$	$^{5}/_{7}$
3	$^{4}/_{8}$	$^{4}/_{8}$	-	$^{4}/_{8}$	$^{4}/_{8}$	$^{4}/_{8}$	$^{5}/_{7}$	$^{1}/_{7}$
4	$^{4}/_{8}$	$^{4}/_{8}$	$^{4}/_{8}$	-	$^{2}/_{8}$	$^{4}/_{8}$	$^{3}/_{7}$	$^{5}/_{7}$
5	$^{6}/_{8}$	$^{6}/_{8}$	$^{4}/_{8}$	$^{2}/_{8}$	-	$^{4}/_{8}$	$^{3}/_{7}$	$^{5}/_{7}$
6	$^{4}/_{8}$	$^{4}/_{8}$	$^{4}/_{8}$	$^{4}/_{8}$	$^{4}/_{8}$	-	$^{1}/_{7}$	$^{3}/_{7}$
7	$^{5}/_{7}$	$^{5}/_{7}$	$^{5}/_{7}$	$^{3}/_{7}$	$^{3}/_{7}$	$^{1}/_{7}$	-	$^{4}/_{6}$
8	$^{5}/_{7}$	$^{5}/_{7}$	$^{1}/_{7}$	$^{5}/_{7}$	$^{5}/_{7}$	$^{3}/_{7}$	$^{4}/_{6}$	-

Table 3. Distances after partial grouping

	1, 2	3, 8	4	5	6, 7
1, 2	-	$^{5}/_{7}$	$^{4}/_{8}$	$^{6}/_{8}$	$^{5}/_{7}$
3, 8	$^{5}/_{7}$	-	$^{5}/_{7}$	$^{5}/_{7}$	$^{5}/_{7}$
4	$^{4}/_{8}$	$^{5}/_{7}$	-	$^{2}/_{8}$	$^{4}/_{8}$
5	$^{6}/_{8}$	$^{5}/_{7}$	$^{2}/_{8}$	-	$^{4}/_{8}$
6, 7	$^{5}/_{7}$	$^{5}/_{7}$	$^{4}/_{8}$	$^{4}/_{8}$	-

Observe that the distances in rows (3, 8) and (6, 7) are the maximums of the distance in rows 3 and 8 or rows 6 and 7 in Table 2 respectively. Continuing the procedure assigns students (1, 2, 3 and 8) to one group, and students (4, 5, 6 and 7) to the other.

In order to test the impact of homogeneous sectioning in practice, we compared three methods of sectioning students: the previous method (sections were assigned to their preferred times with an algorithm for student sectioning similar to the procedure described in Section 8), assigning students to sections randomly, and assigning students to homogeneous sections (see Table 4).

Table 4. Conflict edge reduction with homogeneous sectioning

	Random	Previous	Homogeneous
Conflict Edges	131 212	125 143	103 072
% Increase	21.4%	17.6%	-

Homogeneous sectioning reduced the total number of conflict edges by 17.6% over the previous method, and by 21.4% over a random assignment.

Unfortunately, homogeneous sections are not particularly attractive for academic reasons. In Psych 101 for example, it would be nice to have a mix of students in each section. At Waterloo, we provided the ability for course coordinators to specify the precise allocation of students to sections ahead of time if they are serious about constructing academically balanced sections. In fact, in the common first year Engineering program, they have a special algorithm that distributes students in sections equitably based on gender, program choice, geographic and biographic information.

Of course, the potential reduction in conflict edges is only a side-benefit of homogeneous grouping. The primary consideration is that we need to assign each student to a specific section so that we can define conflicts between each pair of course sections, and use that for timetabling.

Suppose a specific course has four sections. Homogeneous sectioning creates four roughly equal groups of students, but it does not tell us which group should be assigned to which section. The second phase of homogeneous grouping assigns the student groups to sections. Clearly, if there are no constraints on the sections (all equal), then we can assign sections arbitrarily. Normally, each section will have an instructor pre-assigned, and the instructor will have time slot preferences. We constructed an assignment problem where the weight on each edge is equal to the probability that each student in the set would be conflict free if this set were assigned to the given section. For example, for each section, suppose each professor specifies a preferred period and two alternates. We arbitrarily assumed that the section should get it's preferred time 80% of the time, and each alternate 10% of the time. (Similar assumptions were made for all possible preference schemes.) We can then compute the probability that each student would be conflict free with each section.

For example, suppose a particular student has requested a single section course MATH 110, and a multi-section course PSYCH 101. What is the expected number of conflicts if the student is assigned to section 1 of PSYCH 101? The preferred and alternate times are shown in Table 5.

Table 5. Example of expected conflicts

	Preferred time	Alternate times	
PSYCH 101-1	10:30 MWF	9:30 MWF	11:30 MWF
MATH 110	11:30~2:30 M	8:30~11:30M	11:30~2:30 T

If the student is assigned to section 1 of PSYCH 101, then there is a probability of 0.8 that he will be in the preferred time, which will conflict with the first alternate of Math 110 (prob. 0.1). If PSYCH 101 is assigned to the first alternate (prob. 0.1) it will conflict with the first alternate of MATH (prob. 0.1), and the second alternate of PSYCH (prob. 0.1) conflicts with the preferred time of MATH (prob. 0.8). Therefore, the expected number of conflicts for this student, if he is assigned to section 1, is

$$(0.8 \times 0.1) + (0.1 \times 0.1) + (0.1 \times 0.8) = 0.17$$

This is repeated for all students in each group for all of their courses. What is the total number of expected conflicts if the first homogeneous group is assigned to section 1 of PSYCH 101? The same calculation is repeated for each of the homogeneous groups and each section. We then construct an assignment problem: homogeneous groups are assigned to sections to minimize the total number of expected conflicts.

4 Timetable Decomposition

Since we were solving an integrated timetable across the whole University, the size of the problem was a particularly serious issue. With 3 000 independent course sections, the conflict matrix alone has 9 million entries and with an edge density of about 1% for a total of over 90 000 non-zero entries. This was considered too large for optimization. We developed a method to decompose the original problem into a set of smaller problems that could then be solved independently, and later combined optimally (see the working paper by Carter [4] for details). A variation on this procedure has been successfully tested and implemented by Burke and Newell [2].

As a first step in reducing the number of edges, we decided that we should be able to *ignore* any edges where the number of students affected is low (for the purposes of decomposition). Therefore, if a second year Math student wants to take a fourth year German course, we will try to accommodate the request in the final section allocation, but we should not design the sub-problems with this constraint included. We could also remove any conflict edges between a pair of sections that would never overlap in time based on pre-assigned times. For example, a course that must be in the morning cannot conflict with an evening course; and a class pre-assigned to Monday will not conflict with one set on Wednesday. Combining these two concepts, we computed the *expected number of student conflicts* between each pair of sections. It is defined as the number of students requesting both sections multiplied by the (estimated) probability that the two sections will overlap. The probability that the sections overlap uses the principle described in the previous section that the preferred time has a high probability of being chosen, and the alternate periods have a much lower chance. If the expected number of student conflicts is less than one, we delete the edge. This pre-processing reduced the total number of edges significantly (by a factor of 35 in one example). Clearly, this rule could be adjusted.

Decomposition is based on the intuitive assumption that there exists a "natural" partition of the problem into subsets of sections with high interaction. In particular, we would not expect much interaction between advanced language courses and advanced Engineering courses. However, students in both of these programs could be taking Psych 101.

Mathematically, the decomposition algorithm is based on the following concept. Suppose that we can divide the timetabling problem into two pieces called problem A and problem B. Define the edges connecting problems A and B as the "cut set". We will call the decomposition *reasonable* if the two problems can be solved independently. In particular, we want to solve problem A (ignoring B); and then fix those assignments, and solve problem B. If the number of edges in the cut set is small enough, this decomposition should have little or no impact on the solution.

Theoretically, this was modeled as a graph colouring problem assuming that there are no constraints on the available periods for each section. Suppose that you solve the two problems separately and that the solution is represented as a colouring. Is it then possible to find a permutation of the colours in one problem so that there are no conflicts with course section (colour) assignments in the other problem?

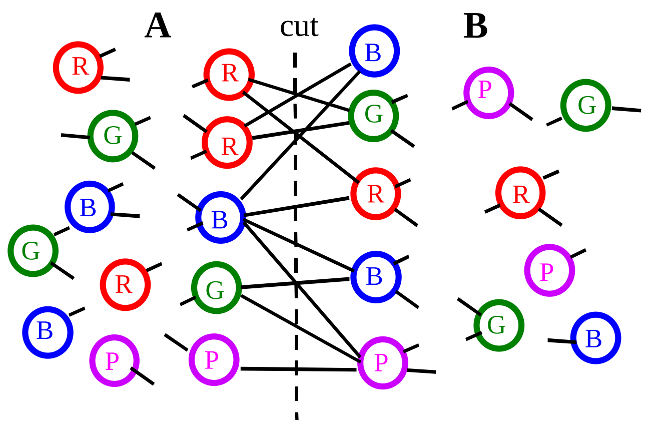

Fig. 2. An example decomposing a colouring problem into two parts

Consider the hypothetical problem illustrated in Figure 2. Suppose there is a large problem that has been divided into two sub-problems, A and B (where only a few of the edges are shown). These two problems are solved independently (using colours Red, Blue, Green and Purple). Observe that there are conflicts between pairs of vertices on either side of the "cut" edges between the two sub-problems. (A conflict occurs if two nodes with the same colour are connected by an edge.) Can we permute the colours in problem B (to an equivalent solution) such that there are no conflicts on the cut set?

Consider only the vertices and edges in the cut set. First, we merge nodes on the same side with the same colour (the two Red nodes in A, and the two Blue nodes in B). The merged nodes inherit the edges from both predecessors. The resulting problem is shown in Figure 3(a). Now, we find the complement set of edges in the cut set (illustrated in Figure 3(b)). An edge appears in the complement if there is no edge in the original graph. If a pair of nodes is connected in the complement, then they are allowed to have the same colour.

We now try to find a *matching* in the complement graph: i.e., find a subset of edges in the complement such that every node in A is connected to a unique node in B. The matching is illustrated in Figure 3(b) as arrows. It is fairly easy to see that, if a matching exists, then the nodes in B can be permuted to take the same colour as the corresponding matched nodes in A: i.e., if *all* nodes in B are permuted such that green becomes blue, red becomes green, blue becomes purple, and purple becomes red, then the solution in B is equivalent, and there will be no conflicts with sub-problem A.

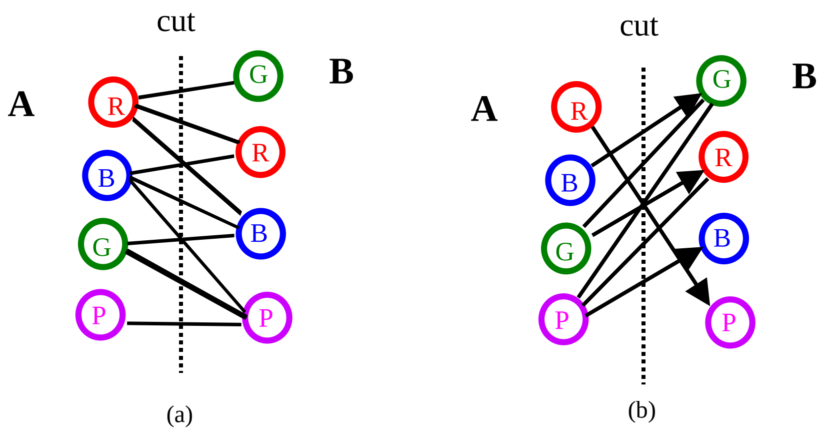

Fig. 3. Finding a matching in the cut set edges

We were able to show that, for randomly generated graphs, this is "almost always" true if the density of cut set edges between the two sub-problems is below a defined value (given in [4] as a series of formulae). For example, if there are 20 colours available, and each of the two problems has 40 vertices, then the problems can be solved independently (with probability 99.99%) if the number of edges in the cut set between the two is less than 588 or about 37% density.

In an early version of the decomposition algorithm, we attempted to find all minimum cut sets (using the Gomory~Hu method [8]), but this was not useful on practical problems. (The Gomory~Hu tree was a star-graph where each cut set cut off one vertex!) Instead, we designed an algorithm that searches for groups of course~sections that *should* be timetabled together based on a high internal density of edges. Specifically, the algorithm looks for a large *clique* in the graph (a subset of course-sections where every pair are in conflict). These must certainly be timetabled together.

The program uses an algorithm developed by Carter and Gendreau [5] to find a large clique in a graph. We then search for any single node that has more edges connected to the subset than to the rest of the graph, and add it to the subset: i.e., this rule reduces the number of edges connecting the subset to the outside. We repeat this process until no new vertices can be added to the subset. These vertices are "removed" from the graph, and we look for another clique in the remaining sub-graph.

The procedure is illustrated in Figure 4. First, the clique routine finds the four nodes denoted by "C". Every pair of nodes in a clique is connected by an edge. Observe that the node labelled "1" has three edges *into* the clique and only two edges *out* to the rest of the graph. We therefore add node "1" to the subset. Node "2" has three edges into the subset (including "1" now) and two out, so we add "2" to the subset. All remaining nodes have more edges *out* than *in*, so we stop, remove the subset, and repeat the procedure on the remaining graph.

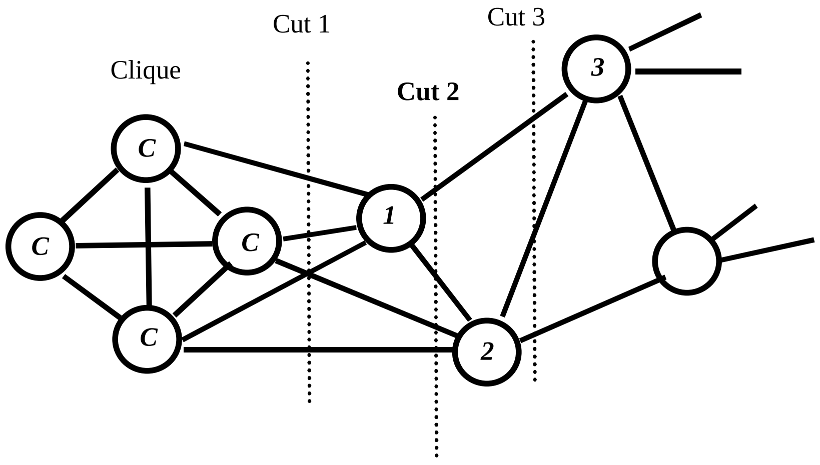

Fig. 4. Decomposition algorithm example

In practice, it was interesting to note that the procedure did not always identify "obvious" sub-problems within a department. For example, one group consisted of 13 third year Health and Kinesiology sections, plus section 04 of Chemistry 123, section 12 of Health 140, and section 01 of Sociology 101. Decomposition had the affect of identifying a large number of small clusters (36) that could be solved independently plus one large central problem! This was not really what we were hoping for; however, the large problem only had 2 500 edges compared to the initial 90 000 plus. We measured the size of the problem in terms of the number of zero~one variables x_{ij} = 1 if section i is assigned to time slot j. Many sections only had a few alternate time slots. On average, the 36 sub-problems had 89 0~1 variables (ranging from 34 to 247). This is still challenging for quadratic 0~1 problems, but not completely hopeless.

5 Automated Course Timetabling

Practical course timetabling is 10% graph theory, and 90% politics! When we first began designing the system, we were warned: “You cannot dictate to professors when they will teach courses!” Consequently, we were told that course timetabling could not work. Or, more precisely, we cannot assume that the timetable can use a clear slate. There are many possible reasons for this. Part-time faculty may be available only on certain days or times. University professors often have other commitments and industrial research projects. Teaching is only a part of their job, and probably less than half of their time is devoted to teaching.

Our solution to this problem was to allow people to constrain the timetable as much (or as little) as they wanted. For each course section: you can specify a “preferred day and time” as indicated in the following examples:

- Must be 10:30 MWF
- Prefer 10:30 MWF, but would accept 9:30 or 11:30 MWF
- Any time in the morning
- Cannot be Friday afternoon
- Any night or weekend slot
- No restrictions.

Our objective was to minimize the total number of student conflicts. (Required course conflicts count double.) The primary constraints were instructor availability, time preferences/restrictions (as described above) and space limitations (room profile). In this phase, we did not attempt to assign specific rooms to each meeting. Instead, for each time period of the week, we constructed a *Room Profile*, as illustrated in Table 6). The profile keeps track of the total number of rooms available by size. We then added constraints to ensure that there would be sufficient space of the appropriate type at all times. For example, if we only have six lecture rooms that hold more than 250 students, we introduce a constraint that no more than six classes over 250 can occur at any one time. Similarly, we constructed separate room profiles for special purpose space. For example, if there are only four physics labs, we constrain the number of concurrent physics lab meetings. Specific rooms are assigned in the room assignment algorithm described in the following section.

Table 6. Room profile (e.g., Wednesday 10:30~11:00)

			Cumulative	
Room size	No. available	No. required	No. available	No. required
1000	1	0	1	0
500	4	3	5	3
250	1	2	6	5
120	7	5	13	10
100	5	8	18	18
90	10	7	28	25
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…

The course timetabling algorithm solves the sub-problems created by the decomposition procedure one at a time. We solve the largest (most difficult) sub-problem first, and then these assignments are fixed and we proceed to the next sub-problem; the assignment tries to allocate the new sections taking any adjacent conflicting sections as pre-assignments. Originally, we were hoping that we could apply a true zero~one integer programming algorithm to solve the sub-problems optimally. Unfortunately, the core problem was still too large (in 1985), so in the end we used a fairly simple "greedy" procedure:

1. Assign the most restrictive course section to the best possible time slot.
2. After all course sections have been assigned, apply a 2-opt procedure to look for any potential improvements.

The most restrictive course section is defined as the section that has the least number of feasible periods. (In particular, there were a large number of pre-assigned sections at Waterloo; so these sections had only one feasible period.) In fact, all course sections with fixed time periods were assigned before any of the sub-problems were solved. Since there are no choices, we might as well use the information when scheduling other sections.

For each feasible period, we calculated the number of student conflicts with other sections that had already been scheduled. The period with the lowest total conflict score was selected. As discussed above, we allowed people the option of specifying a *preferred* time slot and *alternate* time periods. We introduced a penalty for using the alternate time slots so that the algorithm would use the preferred times unless there was a *significant* reduction in student conflicts with the alternate time.

Every section is assigned to some period unless there is no feasible period! (Realistically, the only time this occurs is when too many "large" sections are all pre-assigned to the same time period, and we run out of large rooms.)

The 2-opt local improvement heuristic was quite effective in finding good solutions. At the end of the initial assignment phase, we would ensure that there is no single section that can move to a new feasible time slot and reduce the total student conflicts. We then ensure that there is no *pair* of sections that can switch time slots and reduce the total cost.

For the Fall 1985 session, approximately 60% of the meets were pre-assigned to fixed times by the departments. The remaining 40% (1 230 out of 3 052) had potential alternate times. The program used alternate time slots for only 55 of the meets. (All others were assigned to their preferred time. By assigning these 1.8% of the meets to an alternate time, we were able to reduce the total student conflicts from 19 229 to 17 109 (an 11% decrease). (Note that the previous system simply assigned all sections to their preferred times.)

6 Automated Classroom Assignment

In this phase, we assign classes to specific rooms across campus. At Waterloo, the central Registrar's Office controls the regular lecture rooms. No one is allowed to book these rooms until the classroom assignment procedure has been run. The laboratories are typically reserved for specific lab courses, and the room assignment is simple. However, labs could easily be accommodated as separate sub-problems.

The algorithm uses a lagrangian relaxation approach to solve the problem as a generalized assignment problem (see Carter [3] for complete details; Carter and Tovey [7] present a general discussion of the difficulty of classroom assignment).

$$\underset{x \in \{0,1\}}{\text{minimize}} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} \quad \text{for all } i$$

$$\sum_{i \in P_k} x_{ij} \leq 1 \quad \text{for all } j, k$$

where c_{ij} is the cost of assigning class i to room j, x_{ij}=1 if class i is assigned to room j and P_k is the set of all classes in period k. The problem is a tough mathematical problem in itself. However, it is complicated by the problem of trying to determine an appropriate set of costs. We assign a cost of zero if the room is a perfect match. There are a number of criteria that are used to measure the quality of the match between a class and a given room:

- Location: The University of Waterloo is spread out over a few dozen buildings over a square mile. Each instructor has preferred buildings, often based on the proximity to his/her office or to the department. However, it may be based on the predominant type of seating, facilities or other factors.
- Space Utilization: We want to make effective use of the available space. This includes having sufficient space, and not wasting too many seats. Moreover, when room assignment is run in early June, we do not know exactly how many students are likely to be taking the course; we only have partial pre-registration data. At this stage in 1985, less than 50% of the

students had pre-registered. The user must enter a “minimum required seats” for each class. We assume that any room that is too small is not feasible. We penalize rooms that are too large based on the number of empty seats. Forecasting the minimum required number of seats is a major problem on its own. Professors may tend to over-estimate, and historical trends can be misleading as curricula are revised. At Waterloo, the Registrar’s Office makes an educated guess based on discussions with the Faculty.

- Preferred Rooms: Many instructors have specific rooms that they like best, and we allow them to express a preference. We give a small penalty to all non-preferred rooms. Unfortunately, instructors tend to prefer the same rooms, so it was difficult to satisfy all requests.
- Special Facilities: Instructors may request special features such as audio-visual, computer support, television, special seating types, large blackboards, etc. Some of these are considered requirements while others can be substituted with a penalty.
- Fragmentation: When a class meets two or three times a week, professors at Waterloo liked to get the same room in all meetings. They felt that some people might get confused if meetings were assigned to more than one room. We tried to convince people that this might not be advisable. For example, a poor assignment on Monday (due to lack of proper space) would require the same space on Wednesday and Friday meetings! This was a particularly difficult constraint. It meant that each day could not be solved separately. In the end, we simply used the Monday assignment as a strong preference when we solved the Wednesday problem, etc.
- Standard Slots: The problem of timetabling is much easier if everyone is using a standard set of slot times for their courses. For example, classes at Waterloo tend to meet on the half hour commencing at 8:30 Monday, Wednesday and Friday for one hour. Similar slots occur beginning at 9:30, 10:30, etc. Daytime slots such as two 1.5 hours, two hours, three hours, etc are normally held on Tuesday and Thursday with the slot pattern beginning at 8:30. When someone uses a meeting time of Monday at 9:00 for two hours and Thursday at 2:00 for one hour, they create conflict times for all students in their class for a large number of other standard time slots. We allowed non-standard times, but we introduced a slight penalty: when two classes that are identical in all other respects are competing for the same (preferred) room, the class using standard times will win.

In order to keep the problem manageable, we divided the week into separate days, and divided each day into three parts. We introduced a rule that no class should straddle the 12:30PM time, and no class should go through 6:30PM. This was consistent with the standard time slots in use, and consistent with most logical schedules. The rule allowed us to solve each day as three separate problems: morning, afternoon and evening. Any classes that insisted on straddling the boundaries could either be pre-assigned to a room, or could look for a room after the automated assignment has been run.

The room assignment algorithm is based on the following simplification. Suppose all classes were one hour long (on the half hour). Then room assignment could be solved using a sequence of assignment problems. Using the above criteria, we could compute a *cost* for assigning each class to each room. Unfortunately, at Waterloo, classes have varying lengths. The assignment solution would have students in three~hour classes changing rooms at the end of each hour (potentially), which is not really acceptable. (In practice, most classes stayed in the same room.) In our algorithm, we solved the individual assignment problems, and then introduced "room jumping" constraints to stop classes from changing rooms every hour. These constraints were incorporated as penalty terms in the objective function, and the resulting problem was again solved as a sequence of assignment problems. In a sense, the hard part was coming up with a set of costs that would imitate the preferences and trade-off evaluations of the current schedulers.

The results were excellent. In a parallel comparison with the manual assignments, the algorithm satisfied 50% more preferred building requests (from 89% to 95.5% of the requests for first or second choice space). The automated system had fewer empty seats (from an average of ten empty seats a meeting down to seven). The implication is that the computer system leaves more large rooms empty. This produces more flexibility in the later stages when revisions and additions require late modifications.

The manual system was unable to satisfy 12% of special facility requirements, while the automated system met them all. For room preferences, the computer system satisfied 50% while the manual approach met 25%. The manual system had fewer examples of fragmentation (47 compared with 57), but this was deemed acceptable. Moreover, the manual assignment was estimated to take three people three weeks each term to perform (three terms a year). Therefore, the estimated savings from room assignment alone are 27 person-weeks per year.

In the Fall of 1988, the University of Waterloo opened the Davis Building. A total of 40 (smaller) classrooms across campus were replaced by 30 generally larger classrooms in the new building. The number of rooms decreased, but the total number of seats increased. The Scheduling Office was very concerned about how the new space would impact space allocation. The automated system effectively solved a potentially serious planning problem. The algorithm can be used as a fast "What if?" tool to evaluate proposed changes in space allocation.

7 Interactive Course Timetabling

At Waterloo, each of the 57 departments has a "timetable representative" who has access to on-line timetable information. After the initial timetable and room assignment has been run, timetable reps can log into the system and make manual improvements. The primary display for a selected course section shows a "timetable grid" with days of the week across the top, and times of the day down the right margin. Each square displays the number of student conflicts (required and elective) that would be created if this course section were moved to that day and time. It also displays room (R) and instructor (I) conflicts. The user can select any square in the matrix and drill down to find out what programs the conflicting students are in, or the actual students affected. The display in Figure 5 is intended only to illustrate the type of information in the grid. It is not an accurate representation of the actual screen.

MATH 110 Lecture 01   9:30 MWF   87 students   Room MC 3012 Prof. Carter

Time	Monday	Tuesday	Wednesday	Thursday	Friday
8:30	5	8 R	3	174	4 I
9:00	5	8 R	3	174	4 I
9:30	9	30	8	40	10
10:00	8	30	8	42	10
10:30	6	75	7 R	174 IR	6
11:00	6	75	7 R	174 IR	6
11:30	20 IR	174	22	20	15
etc…					

Fig. 5. Conceptual timetable grid

The user can choose a specific course to investigate, or let the system select "problem" courses. For example, in one option the system searches for courses with a better available time. The system also identifies pairs of courses that could be interchanged to reduce total conflicts.

The timetable reps can also change room assignments. For example, some professors will not get their preferred rooms. They will have an opportunity to change the time for the class to a time when the room is available. Again, the timetable reps use the conflict matrix. They can search for an appropriate available room (at a specified time), display when a given room is available, and book a new room. Essentially, any room that is still open is available to them.

Typically, at this stage, if there is a better time for a class, the program did not find it because the original constraints prohibited some times. The timetable reps can now contact their counterparts in other departments to discuss mutual problems; and/or go directly to the corresponding professors in "problem" courses, and recommend a new time slot. They are allowed to move any of their *own* classes to a new time; but only if the *total cost* improves. If the timetable reps want to move a course to a worse period, they need the approval of the Registrar's Office once approved by the Dean of the Faculty. The changes are done in real time, so that any revisions will immediately be available to all other users.

8 Automated Student Scheduling (Final Section Assignments)

In the homogeneous sectioning algorithm, students were assigned to sections of courses based on similarity to other students in the group. In this routine, we now consider each student separately, and assign him or her to course sections in order to minimize conflicts. The routine also tries to balance the number of students assigned to each section. Each student is solved separately in the order that his or her course requests were posted electronically. The algorithm searches for a feasible section for each course in the order that the students entered them on the registration form. We do not schedule a course for a student unless all of the corresponding sections (lectures, labs, tutorials, etc.) have a feasible assignment.

We wanted to ensure that all students get a space in their required (core) courses. Otherwise, students who register early could fill up the available space in some courses and prevent later students from getting into their core courses. In many cases, we did not know which courses were core for the students, so we simply assumed that the students' first *n* choices were required. A table that governs the value of *n* by Program, Year & Term was developed to separately maintain the calculation of this value. So, we perform the algorithm in two passes. In the first pass, each student is assigned to sections of his or her required courses. We then repeat from the beginning on elective courses.

The routine uses an extension of the Hungarian method for constructing an "alternating path" in a graph. In this case, we construct an "alternating tree". Consider the graph in Figure 6. Each vertex represents a section of a course (for a single student). The vertices are clustered into groups for one course. Suppose that we have already assigned section 01 to course A, section 02 to course B, and we are now trying to find a section for a new course C. If there is a feasible section, we take the one with the maximum number of free seats. Otherwise, we construct a tree where each section of course C is connected to all sections of other courses that would have to be rescheduled (because of conflicts) if this section were assigned to the student. For each of these "conflicting" sections, we construct an edge to every alternate section that could be feasibly assigned. An alternating tree is a feasible set of reassignments to provide an opening for one of the sections of the new course.

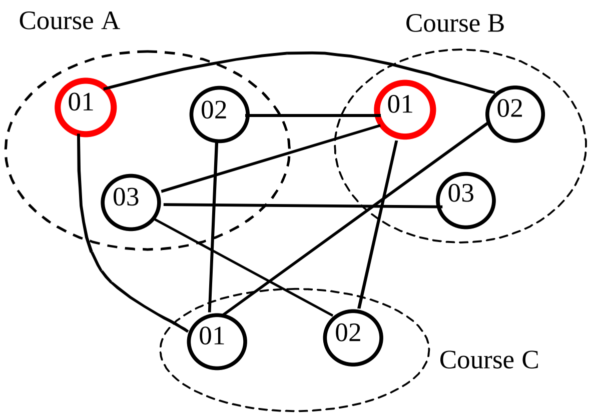

Fig. 6. Multi-section student scheduling

In addition, we want to construct *balanced* section sizes. Therefore, on the *first pass* of the algorithm, we only include sections with the largest number of empty spaces. If there is no feasible solution, we add the next smallest sections and repeat.

9 Student "Drop/Add" Process

In September, the students arrive at the University and go through a "drop/add" procedure. They may change some of their course selections because they did not get all of the courses they asked for, or they may simply show up at the first lecture and decide to switch courses. Department timetable reps are able to change the student schedules on-line. The routine contains a "re-optimize" facility that allows the student to juggle some or all of their existing course sections in order to find a way to insert a new course. The on-line routine allows pre-determined sections, preferred sections and permissible conflicts if approved by the instructors.

10 Conclusions

When the project was initiated, the authors believed that the key to a successful system would depend on the quality of the assignments produced by the automated algorithms. In retrospect, while these components have played a key role, we also realize that simply giving timetable reps the facility to make real time on-line changes was the single most important contribution. Timetabling is highly political, and you cannot hope to incorporate the complete spectrum of preferences and alternatives in a quantitative system.

The system was in use for 15 years. Unfortunately, like so many other schools, Waterloo found themselves with serious year 2000 problems. Moreover, the system was not portable – even within Waterloo. They have decided to purchase a commercial package from Peoplesoft to provide their administrative student system support, and they are in the process of buying other packages for optimization. Converting the old ones with a new database is to labour intensive.

11 Acknowledgements

There were a large number of people who contributed to the success of this project. At the risk of omitting someone, the author would like to mention Vivienne Ballantyne, Adam Belsky, Martha Dolan, Mike Esmond, Janice Foster, Dave Fritz, Gary Marx, Dave Mason, Cam McKay, Wayne Montgomery, Cheryl Myers, Lynn Tucker, Roy Wagler and Rudy Zigler

I am particularly grateful to Karen Ledrew, Associate Registrar, for her help in the design of the system and her constant support.

References

1. Aubin, J. and Ferland, J.A., "A Large Scale Timetabling Problem", Computers & Operations Research 16, 67-77, 1989.
2. E.K. Burke and J.P. Newall, "A Multi-Stage Evolutionary Algorithm for the Timetable Problem", the IEEE Transactions on Evolutionary Computation, Vol 3.1, 63-74, 1999
3. Carter, M.W., "A Lagrangian Relaxation Approach to the Classroom Assignment Problem", INFOR 27, No. 2, pp. 230-246, 1989.
4. Carter, M.W., “A Decomposition Algorithm for Practical Timetabling Problems”, Working Paper, Mechanical and Industrial Engineering, University of Toronto, 1983 (available from www.mie.utoronto.ca/staff/profiles/carter.html)
5. Carter, M.W., and Gendreau, M., “A Practical Algorithm to Find a Large Clique in a Graph”, Working Paper, Mechanical and Industrial Engineering, University of Toronto, 1999.
6. Carter, M.W. and Laporte, G., "Recent Developments in Practical Course Timetabling", in Practice and Theory of Automated Timetabling, Springer-Verlag Lecture Notes in Computer Science 1408, E.K. Burke and M.W. Carter, eds., Springer, 1998.
7. Carter, M.W. and Tovey, C.A., "When Is the Classroom Assignment Problem Hard?", Operations Research 40, S28-S39, 1992.
8. Hu, T.C., Integer Programming and Network Flows, Addison-Wesley, Reading, MA, 1969.
9. Paechter, B., Cumming, A. and Luchian, H., "The Use of Local Search Suggestion Lists for Improving the Solution of Timetable Problems with Evolutionary Algorithms", In Proceedings of the AISB Workshop on Evolutionary Computing, (Sheffield, April 3~7, 1995.
10. Rankin, R.C., "Automatic Timetabling in Practice", in Practice and Theory of Automated Timetabling, Springer-Verlag Lecture Notes in Computer Science 1153, E.K. Burke and P. Ross, eds., Springer, 1996.
11. Sampson, S.E., Freeland, J.R. and Weiss, E.N., "Class Scheduling to Maximize Participant Satisfaction", Interfaces 25, No. 3, 30-41, 1995.
12. University of Waterloo web site http://www.uwaterloo.ca.