A Multi-Objective Timetabling System That Facilitates Scheduling Across Academic Programs

Dan S. Myers · Jay Yellen

Abstract We describe the design and implementation of a course scheduling system with the primary objective of assigning non-overlapping timeslots to pairs of courses taught by the same instructor or likely to be needed (or desired) by a common set of students, including course pairs in different academic programs. A secondary objective is to create compact schedules for both faculty and students. Our system models the timetabling problem as a vertex coloring of a weighted graph. We present three coloring algorithms for this graph model: an exact mixed-integer programming formulation and two approximate algorithms. We evaluate the performance of the three algorithms on a set of randomly generated test problems derived from actual course schedules at Rollins College. Results show that the heuristic algorithms are competitive with the mixed-integer program and have significantly shorter run times. In comparison with an actual Rollins course schedule, our system reduced the number of conflicts by roughly one half, and created more compact schedules for faculty and students.

Keywords timetabling · course scheduling · graph coloring

1 Introduction

We describe the design and implementation of a course timetabling system for Rollins College. Constructing feasible and practical course schedules that meet the needs of faculty and students is a significant challenge for colleges and universities. Clearly, courses taught by the same instructor must be assigned non-overlapping timeslots. However, there are many other pairs of courses that are required (or desired) by the same set of students in the same semester, and a number of these course pairs will involve different departments when majors in one department elect to minor in another. For example, at Rollins, there are several economics majors who also minor (or double major) in mathematics.

Dan S. Myers, Jay Yellen
Rollins College
Dept. of Mathematics and Computer Science
E-mail: dmyers@rollins.edu
E-mail: jyellen@rollins.edu

The core model underlying our system is a weighted graph with two-component edge weights, extending the models used in [19,11,29]. Each vertex in the graph represents a section of a course, with a set of acceptable timeslots and acceptable rooms. Edges correspond to course pairs that are likely to have students and/or faculty in common or require the same special resource, such as a lab or classroom. The timetabling construction is then modeled as a vertex coloring of the resulting graph.

The two-component edge weights correspond to the two objectives of our model. The first, the *conflict penalty*, is incurred when the endpoints are assigned overlapping timeslots. High conflict penalties (i.e., heavy edge-weights) correspond to pairs of courses that are either taught by the same faculty member or demanded by large numbers of students in the same semester. The second objective, the *proximity penalty*, is incurred when the endpoints are assigned timeslots having a large gap on the same day. Thus, our model incorporates two objectives related to key aspects of the scheduling problem: minimizing conflict and creating compact schedules.

In order to determine which course pairs correspond to adjacent vertices, and the size of the edge-weights, we conducted a survey of the faculty. For each course, faculty members were able to specify a list of acceptable timeslots, a list of acceptable rooms, and a list of courses that should not be scheduled in overlapping timeslots, along with the perceived severity of each potential conflict. These perceived severities were converted to edge weights that then became the input data for our graph model.

Research into automated course timetabling has a long history. See [21,23,9,26, 12] for relevant surveys. Many approaches, including heuristic graph coloring [30, 16,19,5,6,11] and metaheuristic algorithms based on local search and evolutionary computation [1,14,4,27,7,17,3], have been used to create course schedules.

This paper consists of the following sections:

1. A description of the implementation of the dual-objective weighted graph model from faculty survey information (Section 2)
2. Presentation of three approaches to finding suitable colorings for these graphs: an exact mixed-integer programming (MIP) formulation (Section 3.1), a one-pass (Section 3.2) and a tree-based beam search (Section 3.3). The two heuristic algorithms are augmented by a post-construction repair phase that improves the initial schedule (Section 3.4).
3. An evaluation of these approaches using a randomized set of test problems created using actual course schedules at Rollins College as seeds for the random generation (Section 4.1-4.2). Our results show that the MIP produces the highest-quality schedules, as expected, but suffers from highly variable run times. The differences in the numbers of conflicts produced by the three algorithms are small. In particular, the additional conflicts and higher schedule gaps produced by the approximate algorithms affect only a small number of faculty and students.
4. An examination of the performance of our system on a real course timetabling problem: the construction of the Fall 2017 Science Division schedule at Rollins College (Section 4.3). By comparison to the actual schedule produced by conventional ad-hoc methods, we show that our system produces approximately 50% fewer conflicts and creates more compact schedules for instructors.

2 A Dual-Objective Weighted Graph Model

2.1 Timetabling as a Graph Coloring Problem

Timetabling algorithms based on graph coloring have a long history, for example [30, 16, 19, 5, 12, 6, 23]. These models treat each event to be scheduled as a vertex in a graph, with edges representing possible conflicts between event pairs. A proper coloring of the graph assigns a color to every vertex so that adjacent vertices receive different colors.

Our model combines the conflict and proximity penalties into a weighted sum, described in more detail below, representing the total penalty incurred across the entire schedule. An alternate approach would be to formulate the problem as a true multi-objective optimization problem, as in [15, 13]. For the purposes of our system, the single objective form integrates better with the heuristic algorithms discussed in Section 3.

2.2 Conflict and Proximity Objectives

Figure 1 shows an example four-course graph where each edge carries two weights. The first weight corresponds to the conflict penalty that is incurred if the two end points of the edge are assigned to overlapping timeslots.

Continuous-valued conflict penalties are possible, but we have found it helpful in our discussions with other faculty and administrators to classify course conflict penalties into three categories:

- *Heavy*, representing prohibitive conflicts, such as two courses taught by the same instructor or courses required in the same semester by all students in a major.
- *Medium*, representing conflicts that are not prohibitive but desirable to avoid, such as introduction to physics and introduction to computer science.
- *Light*, conflicts that affect only a small number of students, but are still known to regularly occur.

The example of Figure 1 uses weights of 400, 25, and 1 for the three categories, respectively. The choice of conflict-weighting scheme is adjustable as a parameter in our system.

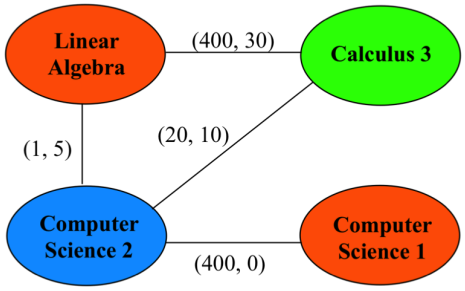

Fig. 1 An four-course graph with dual-weighted edges

The second weight, the *overlap factor*, is used in the calculation of the proximity penalty. Since we desire to create compact schedules for the greatest number of students and faculty, it is reasonable that gaps between courses with several members in common should make a greater contribution to the proximity penalty than courses that have few members in common. The overlap factor quantifies this.

For each course pair (v_1, v_2), the proximity penalty incurred is given by

$$p(v_1, v_2) = overlap(v_1, v_2) \cdot gap(v_1, v_2) \tag{1}$$

where $gap(v_1, v_2)$ returns the time difference between the assigned timeslots for courses v_1 and v_2. Thus, courses that have few students or faculty in common contribute little or nothing to the proximity penalty.

In Figure 1, Linear Algebra and Calculus 3 have a heavy conflict penalty of 400 and a high overlap factor of 30, because they are typically taken in the same semester by all math majors. Computer Science 1 and Computer Science 2 have a heavy conflict penalty, but no overlap factor; this indicates that they are not taken by the same group of students, but might share a required resource, such as a computer lab.

Determining the overlap factors is straightforward if the enrollment for every course is known before the schedule is constructed, as is the case in most U.K. institutions. At U.S. institutions, the overlap factors must be obtained by analyzing historical cross-enrollment data to determine the number of students that are likely to register for a given pair of classes. This is the approach we followed to create a seed for our randomized test problems, discussed in Section 4.1.

In addition to the two-component weights assigned to each edge, each vertex in our model maintains a list of *color-penalty pairs*, with one entry for each color. For a vertex v, the color-penalty pair list has the form

$$[(\eta_1^v, \nu_1^v), (\eta_2^v, \nu_2^v), \ldots, (\eta_c^v, \nu_c^v)]$$

where η_i^v and ν_i^v keep track of the conflict and proximity penalties, respectively, that will be incurred if color i is assigned to vertex v.

During the coloring process (described in more detail in Section 3), the vertex color-penalty vectors track the potential effects of each possible color assignment. Each time a vertex is colored, the color-penalty pairs of all its neighbors are updated, reflecting the conflict and proximity penalties that now exist with respect to the newly colored vertex. This approach allows for rapid calculation of the cost of each color assignment at every step in the coloring.

In some cases, a color i may be inherently unsuitable for vertex v, but not prohibited outright. For example, a faculty member may agree to teach in early morning timeslots, but only if no better options are available . This feature may be implemented by setting the conflict penalty element η_i^v to a nonzero initial weight reflecting the severity of the unsuitability. The proximity penalty elements ν_i^v are solely determined by the color assignments of the neighbors of v, so they are always set to 0 at the beginning of the coloring process.

3 Constructing Schedules by Graph Coloring

3.1 Mixed-Integer Programming

Tulio Toffolo proposed a mixed-integer programming (MIP) formulation for the dual-objective weighted graph model described in Section 2 [28]. The model represents the assignment of a particular course section to a particular room in a particular timeslot as a binary decision variable, with penalties that are incurred if courses are assigned to overlapping timeslots.

$$\min \quad \sum_{i \in E} \sum_{c \in C_i} \sum_{r \in R_i} \alpha_{icr} x_{icr} + \sum_{i \in E} \sum_{j \in E_i} \sum_{c \in C_i} \sum_{d \in C_j} \beta_{ijcd}\, y_{ijcd} + \sum_{(c,d) \in S} \sum_{r \in R} \gamma z_{cdr}$$

s.t.

$$\sum_{c \in C_i} \sum_{r \in R_i} x_{icr} = 1, \quad \forall i \in E$$

$$\sum_{r \in R_i} x_{icr} + \sum_{r \in R_j} x_{jdr} - y_{ijcd} \leq 1, \quad \forall i \in E,\ j \in E_i,\ c \in C_i,\ d \in C_j$$

$$\sum_{i \in E_{cr}} x_{icr} + \sum_{i \in E_{dr}} x_{idr} - z_{cdr} \leq 1, \quad \forall (c,d) \in S,\ \forall r \in R$$

$$x_{icr} \in \{0,1\}, \quad \forall i \in E,\ \forall c \in C_i,\ \forall r \in R_i$$

$$y_{ijcd} \in \{0,1\}, \quad \forall i \in E,\ j \in E_i,\ c \in C_i,\ d \in C_j$$

$$z_{cdr} \geq 0 \text{ and integer}, \quad \forall (c,d) \in S,\ \forall r \in R$$

where

E = the set of courses
E_i = the set of courses that have non-zero conflict severity with course i
E_{cr} = the set of courses for which color c and room r are suitable
C_i = the set of acceptable colors for course i
R_i = the set of acceptable rooms for course i
α_{icr} = inherent penalty of assigning color (timeslot) c and room r to course i
β_{ijcd} = penalty for assigning course i to color c and course j to color d
γ = penalty for double-booking a room
$x_{icr} = 1$ if course i is assigned color c and room r, 0 otherwise
$y_{ijcd} = 1$ if i is assigned color c and j is assigned color d, 0 otherwise
z_{cdr} = number of times room r is assigned overlapping colors c and d

The first component of the objective function captures the inherent unsuitability of timeslot c and room r for course i. The second captures the conflict and proximity penalties of assigning two courses i and j to timeslots c and d. The third component penalizes double-booking room r overlapping timeslots c and d.

The first constraint ensures that each course is assigned only one room and timeslot. The second forces $y_{ijcd} = 1$ when courses i and j are assigned colors c and d, respectively; this incurs penalty β_{ijcd}. The third forces $z_{cdr} = 1$ if room r is double-booked in timeslots c and d, incurring penalty γ.

Algorithm 1 One-Pass Scheduler

Input:
G the uncolored weighted graph representing the timetabling problem
H_v the vertex-selector heuristic parameters
H_{cr} the color-selector and room-selector heuristic parameters

while there are uncolored vertices **do**
$v \leftarrow selectVertex(G, H_v)$
$(color, room) \leftarrow selectColorAndRoom(v, G, H_{cr})$
assign $(color, room)$ to v

Despite the advantages of an exact MIP, there are still reasons to investigate approximate algorithms. In particular,

- The run time of the MIP is highly variable: four of the ten problems in our test set require at least one hour of solution time and one requires almost three hours. Section 4.2 discusses these results in more detail.
- The efficiency of the scheduler depends on the power of the underlying MIP solver, but not all organizations performing scheduling have access to high-quality commercial solvers.
- Our work with the College's Registrar has shown that schedule always requires a series of post-construction edits to deal with shifts in faculty preferences and unexpected changes. In previous work, this editing process was enhanced by re-running the scheduler to evaluate the impact that each potential change would have on the overall schedule [29]. Even run times on the order of minutes (the fastest achieved by the MIP on our randomized test problems) are too slow to support this process.

3.2 One-Pass Greedy Search

Our two search algorithms treat the graph coloring process as a state-space search, where nodes in the state-space tree represent partial colorings of the graph, the root node represent the initial uncolored graph, and leaf nodes represent completed colorings. Figure 2 depicts an example of such a search tree.

The ultimate goal of the state space search is to identify the leaf node having minimal penalty: this node represents the best available coloring, and hence, the best possible schedule. However, a complete examination of all possible leaves is impossible outside of trivial cases. Therefore, practical search strategies must rely on heuristics.

The simplest heuristic for creating a coloring is the *one-pass* approach, which begins at the root and works to a single leaf, choosing one vertex to color at each step [29]. At each step, the method selects a "troublesome" vertex that is likely to cause difficulty if its coloring is deferred until later in the scheduling process. For example, a vertex of high degree is likely to be hard to color if most of its neighbors have already been colored. The algorithm chooses a timeslot and room assignment that seem best with respect to both the current incomplete schedule and the remaining uncolored vertices. These steps repeat until all vertices have been colored. Algorithm 1 summarizes this procedure.

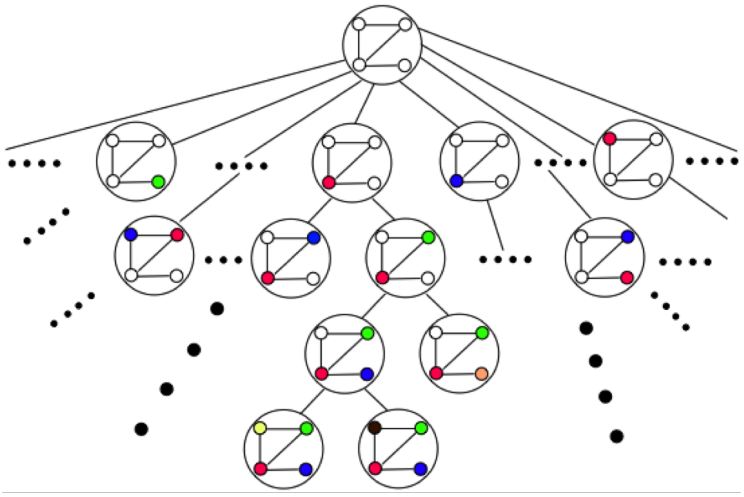

Fig. 2 An example search tree over possible graph colorings

Designing an effective greedy algorithm requires balancing choices that are optimal with respect to the current in-process coloring against the need to leave scheduling options open for vertices that have yet to be colored. Previous work has explored using combinations of heuristics to identify both troublesome vertices and make good color assignments [10]. These heuristics are based on intrinsic properties of the weighted graph, such as the density of edges or the number of acceptable colors at a given vertex.

3.3 Beam Search

The one-pass strategy is limited in that it generates only one path to a complete coloring without backtracking. We extended the one-pass system by incorporating a beam search approach that allows for both branching and backtracking [24].

The beam search algorithm maintains a fixed-size priority queue of partial colorings [2]. At each step, the algorithm selects the most promising partial coloring from the queue and expands it by generating a set of successor colorings, each of which includes one more colored vertex. If each expansion step selects N_v vertices and assigns each vertex one of N_c colors, the total number of successors (i.e., the branching factor) is $N_v N_c$.

These successors are scored by a priority function and inserted into the queue. If necessary, the least-promising colorings are pruned to keep the queue length within a fixed limit. The beam search can thus be considered a limited-memory variant of the traditional A* search, where the "best" node (as chosen by some priority function) is expanded at each iteration, but only a fixed-size subset of the most promising nodes are maintained in memory [22].

The principal challenge of the beam search implementation is selecting a good priority function for ordering the queue of partial colorings. Intuitively, a promising coloring is one that is likely to lead to a leaf (that is, a completed coloring) with

Algorithm 2 Beam Search Scheduler

Input:

G	the uncolored weighted graph representing the timetabling problem
queue	an initially-empty priority queue of partial colorings
P	the priority function
L	maximum priority queue length
N_v	number of vertices to choose and expand on each iteration
N_c	number of color-room combinations to apply to each expanded vertex
H_v	the vertex-selector heuristic parameters
H_{cr}	the color-selector and room-selector heuristic parameters

```
queue.setPriorityFunction(P)
queue.push(G)

while true do
    g ← queue.pop()
    if g is a complete coloring then
        return g

    for 1 to N_v do
        v ← selectVertex(v, g, H_v)
        for 1 to N_c do
            (color, room) ← selectColorAndRoom(v, g, H_cr)
            g' ← assign (color, room) to v in partial coloring g
            queue.push(g')

    if queue.length > L then
        remove all but L most promising partial colorings from queue
```

low total penalty. As in traditional A*, the priority function may be decomposed into two parts:

- The current penalty accrued by the incomplete partial coloring
- An estimate of the difficulty of completing the coloring

[24] introduced a set of heuristics based on properties of the graph model, similar to the heuristics used in the one-pass vertex-selection, to estimate the difficulty of completing a given partial coloring.

Algorithm 2 summarizes the beam search procedure.

3.4 Repair and Improvement Phase

Initial solutions generated by approximate algorithms can often be improved by a applying a local search heuristic to the completed schedule [8]. Because the approximate algorithms must consider the effects of timeslot and room assignments on other courses, they produce looser, more conservative schedules with larger gaps than those generated by the MIP.

Our current implementation uses a randomized hill-climbing search for repair and improvement. At each step, the system randomly selects a course section, with the selection weighted so that sections with higher penalties are more likely to be selected. It then considers alternate timeslot and room assignments for the selected course, and, if possible, selects one that reduces the overall penalty of

the complete schedule. This process repeats until the schedule converges and no further improvement is possible.

This technique, like all hill-climbing methods, is fast, easy to implement, and is guaranteed to produce a result at least as good as the starting schedule. In our experiments, the search procedure typically improves the initial penalty scores of the one-pass scheduler by 20-30% and the beam search scheduler by 10-20%.

Hill climbing lacks flexibility, however, because it cannot trade a short-term increase in penalty for a larger long-term decrease. Other methods, such as simulated annealing, have more flexibility and might result in greater overall improvement [20]. Investigating more sophisticated improvement algorithms is a subject of our future research.

4 Experimental Comparisons

4.1 Generating Random Test Problems

To test the effectiveness of our three proposed solution algorithms, we obtained the complete College schedules for each of the last five semesters, including records on student cross-enrollments in each pair of courses. Each complete schedule contains approximately 650 individual course sections with more than 300 instructors meeting in more than 150 distinct meeting times (e.g., MWF 8:00-8:50, MW 8:00-9:15, etc.).

Using this data as a seed, we generated a set of randomized test problems for our experimental comparisons. Each course is given a random set of five acceptable rooms drawn from the same building as its assigned room, and a set of up to five acceptable timeslots meeting the same days for the same number of contact hours as its original assigned timeslot. To simulate variability in enrollments across semesters, we also randomized the conflict penalties and overlap factors on the edges between pairs of courses, with a small probability of introducing edges among previously non-adjacent pairs or deleting existing edges. To maintain feasibility, the original assigned rooms and timeslots were maintained as acceptable options for every course.

4.2 Performance of the Three Algorithms

Tables 1, 2, and 3 summarize the performance of the three algorithms (including the hill climbing repair and improvement phase) on 10 randomized problems. Each table reports the final penalty function value (a weighted sum of both the conflict and proximity scores) obtained by the three algorithms on each problem, as well as the number of heavy, medium, and light conflicts and the number of courses lacking assigned rooms at the end of the scheduling process.

Both heuristic algorithms produce zero heavy conflicts. This is encouraging, since a heavy conflict would represent a prohibitive violation that makes the schedule non-viable. As expected, the MIP achieves lower total penalties than both the one-pass and beam search schedulers; its final penalty scores are consistently 20-30% lower than the corresponding one-pass scheduler penalties. Only one course (out of 6580 scheduled across the 10 trials) is left with an unassigned room by

Table 1 Results of the MIP scheduler on 10 randomized problems

Problem	Total penalty	Heavy conflicts	Medium conflicts	Light conflicts	Unassigned rooms	Run time (seconds)
1	12503	0	1	52	0	6570
2	13386	0	2	40	0	513
3	14050	0	4	46	0	10748
4	13456	0	1	50	0	790
5	15088	0	6	38	0	475
6	15100	0	5	42	0	4406
7	15236	0	5	35	0	4226
8	15101	0	5	43	0	473
9	14027	0	4	32	0	215
10	13971	0	2	40	0	880

Table 2 Results of the one-pass scheduler on 10 randomized problems

Problem	Total penalty	Heavy conflicts	Medium conflicts	Light conflicts	Unassigned rooms	Run time (seconds)
1	17130	0	2	58	0	2.9
2	17295	0	4	55	0	2.8
3	16941	0	5	48	0	3.0
4	18406	0	2	50	1	2.9
5	19274	0	6	56	1	2.9
6	19145	0	7	54	0	2.9
7	18621	0	5	54	0	2.9
8	21122	0	5	50	2	2.8
9	18844	0	6	38	0	2.9
10	18102	0	2	49	0	3.0

Table 3 Results of the beam search scheduler on 10 randomized problems

Problem	Total penalty	Heavy conflicts	Medium conflicts	Light conflicts	Unassigned rooms	Run time (seconds)
1	15299	0	1	59	0	145
2	16838	0	3	53	0	124
3	17153	0	4	50	0	165
4	16135	0	2	46	0	147
5	19680	0	6	49	1	140
6	18973	0	6	55	0	147
7	18304	0	5	36	0	137
8	20084	0	6	44	0	148
9	16944	0	4	32	0	141
10	17246	0	2	59	0	128

the beam search scheduler, and only four by the one-pass algorithm. Although unassigned rooms are not ideal, they can usually be dealt with by moving a course to a free room in a nearby building.

Differences in the three algorithms' conflict penalty scores, therefore, are due to differing numbers of medium and light conflicts. As discussed in Section 2, medium conflicts represent courses that are sometimes taken by overlapping groups of students during the same semester, but do not rise to the level of a prohibitive conflict—at Rollins College, an overlap of three to six students would typically be considered a medium conflict. Light conflicts are less serious and represent a conflict that affects only one or two students.

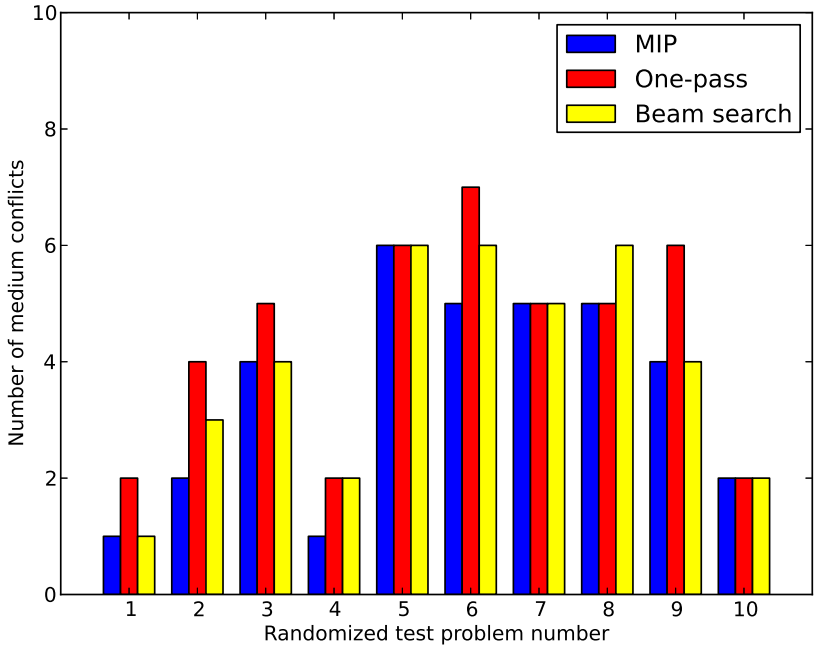

Fig. 3 Numbers of medium conflicts incurred by the three scheduling algorithms

Figure 3 plots the number of medium conflicts incurred by each of the three scheduling algorithms on each of the three test problems. In every case, the one-pass and beam search algorithms are within two conflicts of the optimal MIP result. Given that each medium conflict represents a problem for perhaps six students, this suggest the additional conflicts generated by the approximate algorithms affect no more than 10-20 students.

Figure 4 plots the number of light conflicts incurred by each of the three algorithms. Here, the difference between the MIP and the approximate algorithms is larger. On problem 7, for example, the MIP's schedule contains only 35 light conflicts, but the one-pass result contains 54. Light conflicts tend to occur for idiosyncratic reasons, such as a single student majoring in an unusual combination of fields; some light conflicts also reflect overlaps between academic courses and athletic practices. Still, given that each light conflict affects only one or two students, the additional conflicts generated by the approximate algorithms affect a group of only 20-40 students.

Figure 5 demonstrates the performance of the three algorithms on the secondary objective of creating compact schedules. The figure plots distribution of the total length of the teaching day for all faculty members on test problem 1, which is tied for the largest difference in total penalty between the MIP and approximate algorithms. Similar results are obtained on the other test problems.

The figure shows that all three algorithms produce similarly compact schedules. Approximately 30% of faculty have teaching length days of 1 hour: these faculty have no more than one section scheduled on each day. A small number have very long days of more than ten hours: these are due to a small number of faculty

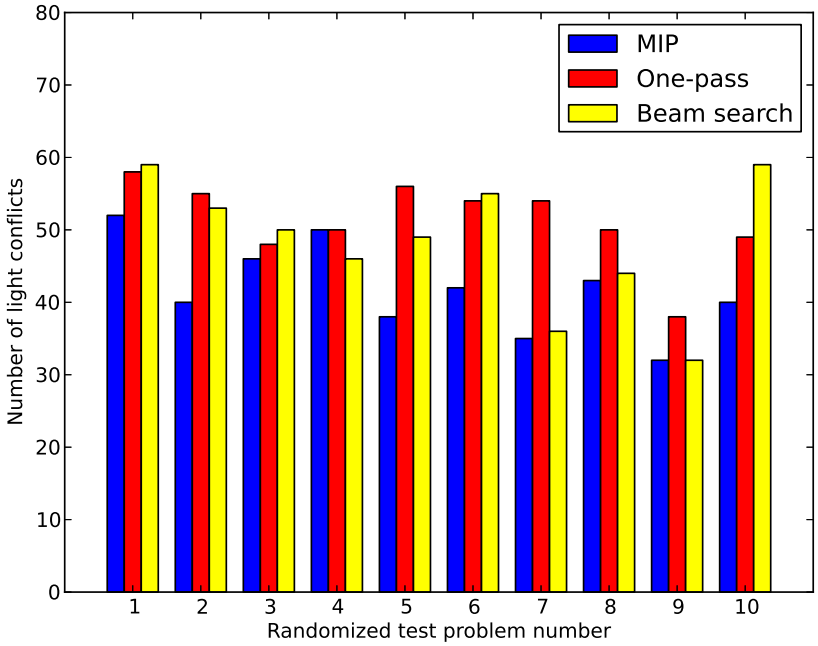

Fig. 4 Numbers of light conflicts incurred by the three scheduling algorithms

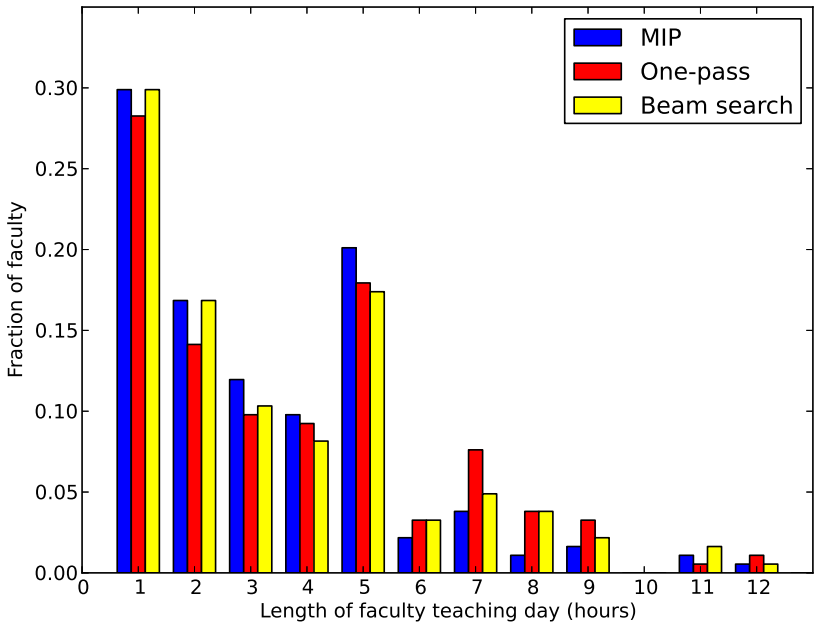

Fig. 5 Distribution of lengths of faculty teaching days

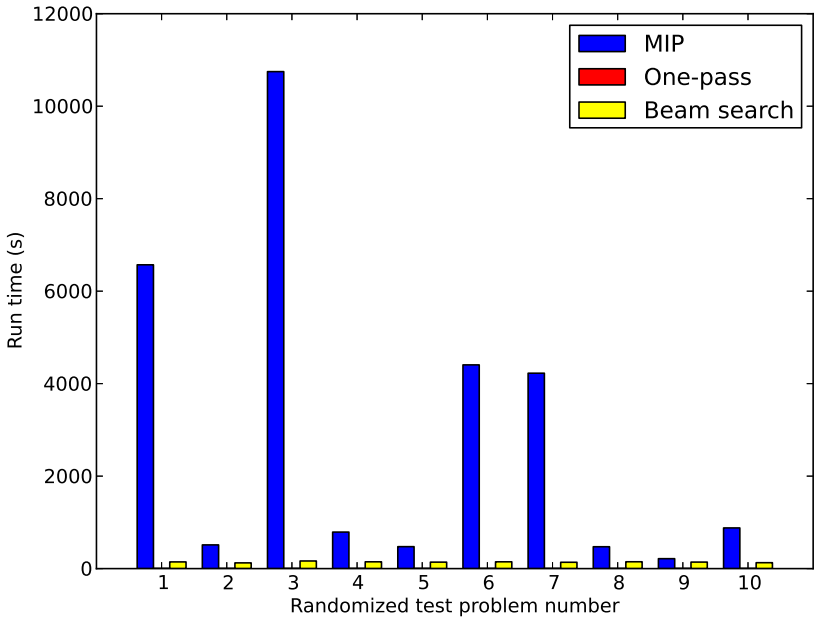

Fig. 6 Run times of the three algorithms on 10 randomized problems. The one-pass algorithm achieves run times of two the three seconds on all problems.

that must teach general education courses at reserved times in the morning and evening courses for non-traditional students late on the same day. In practice, faculty facing this situation typically reconfigure their schedules to teach the early and late courses on different days, but we did not attempt to automatically apply that optimization in our tests.

The MIP has the highest fraction of short teaching days, followed by the beam search and then the one-pass, as might be expected. The MIP exhibits superior performance with medium-length gaps: it is able to schedule a greater proportion of courses in the three to six hour range than the other two schedulers, which produce a few more six to eight hours gaps instead. Overall, though, the results indicate that both heuristic algorithms due a reasonable job of creating compact schedules, and are competitive with the MIP for the large majority of faculty schedules.

Figure 6 plots the run times for each of the three solution algorithms on the set of 10 randomly generated test problems, which shows the high variability in MIP run times, ranging from four minutes to nearly three hours. This variability is somewhat surprising, since the ten problems are generated according to the same random process and are of roughly equal difficulty. Run time is not clearly correlated with final solution penalty. Problem 3, with a run time of nearly three hours, has the sixth highest penalty in Table 1; problem 1 has the lowest penalty produced by the MIP but the second highest run time. Future work may seek a clearer understanding of how the structural characteristics of the graph contribute to high MIP run times.

The beam search and one-pass algorithms exhibit consistent performance across all ten problems. The beam search consistently takes two to three minutes to generate each solution and the one-pass algorithm requires only three seconds. Note that the one-pass algorithm is so fast that it is barely visible in Figure 6. These algorithms have been implemented in Python and, though they are efficient, have not been optimized for performance. They would be even faster if re-implemented in a compiled language such as C++.

4.3 Comparison to an Actual Course Schedule

Finally, we compare the results produced by our method to the actual course timetable for the Rollins College Science Division in the fall semester of 2017.

At Rollins, course scheduling is handled at the level of academic departments. Each department, led by its chair, creates its own schedule, including the assignment of timeslots. Chairs generally have awareness of conflicts that exist in their own program, but only rarely do they take into account potential cross-department conflicts or students' scheduling concerns. The resulting schedules always require considerable manual reworking by the College registrar, particularly of room assignments. Science scheduling is always complicated by limitations on available lab space and the need to avoid conflicts among introductory courses in physics, biology, chemistry, and calculus that are frequently taken by the same groups of students. Most natural science courses meet at least four days a week with a four hour lab period.

We surveyed the members of our six Science Division departments—biology, chemistry, computer science, mathematics, physics, and psychology—and asked faculty to supply the following information for the courses offered in the Fall 2017 semester:

- The list of acceptable timeslots for each course.
- The list of acceptable rooms for each course, with an option that any sufficiently large room in the department's normal building was acceptable.
- The courses that, in the faculty's perception, should not be scheduled in overlapping timeslots with their own courses (i.e., potential conflicts) along with estimated severities (heavy, medium, or light).

The resulting data contained 124 total course sections taught by 44 faculty in 65 different acceptable timeslots. The faculty surveys identified 169 total heavy conflicts between course sections, 532 total medium conflicts, and 49 light conflicts—most faculty did not notate light conflicts on their surveys. The courses with the largest numbers of conflicts were introductory chemistry, biology, and physics sections, their labs, and calculus.

Table 4 summarizes the number of conflicts produced by our scheduling system with this data, compared to the results obtained by taking the actual Fall 2017 schedule and evaluating it with respect to the survey information provided by faculty. Run times were acceptable in all cases, including the MIP scheduler, and all schedules contained appropriate room assignments. All three scheduling algorithms produce results that improve upon the actual course schedule.

The optimal MIP schedule contains 7 heavy conflicts: investigation shows that these are unavoidable, and are caused by a precalculus class and multiple sections

Table 4 Comparison of actual and optimized Fall 2017 science schedules

Schedule	Heavy conflicts	Medium conflicts	Light conflicts	Run time (s)
Optimal (MIP)	7	28	5	235
Beam-search	8	34	4	9
One-pass	9	39	5	1
Published	20	57	3	N/A

of a chemistry lab that have only Friday afternoon meeting times. We could likely avoid these conflicts by providing more flexibility in the precalculus meeting times. The actual schedule, by contrast, includes 13 additional heavy conflicts. These include, for example, required pre-medical courses (Genetics and Microbiology) offered at the same level by the department, but scheduled at competing timeslots. It also includes several cross department conflicts that are likely to affect groups of students, including organic chemistry and general biology and general biology and calculus. Medium conflicts show similar concerns, though these are ranked as less significant by faculty.

These results suggest that, despite our best efforts, the current scheduling process does not minimize conflicts among related courses, even in cases where the conflict is obvious and the courses are offered by the same department. We have shared these results with faculty with the goal of improving our scheduling in future semesters.

Figure 7 plots the distribution of total teaching day lengths for faculty in the optimized MIP schedule and the actual timetable—results for the other heuristic algorithms were qualitatively similar to the MIP. The distribution shows that more than 30% of faculty have long teaching days of six hours or more under the actual schedule; under the MIP results, the comparable number is only 15% and most of these are unavoidable.

5 Related Work

[30] described the basic application of graph coloring to job scheduling and provided a bound on the number of required colors. More recently, [12] investigated and compared several heuristic approaches on a set of real and randomly generated exam timetabling problems and [5] described the implementation of several algorithms in a spreadsheet-based decision support system. [10] introduced the concept of color selection using linear combinations of primitive heuristics. Detailed surveys of both algorithms and systems are available in [9,6,21,23].

In the context of this work, [19] introduced a weighted graph model that was the predecessor of our model. [11] adapted this model to examination timetabling and [29] applied to the 2011 Science Division schedule at Rollins. [24] introduced the beam search algorithm and presented a comparison of the effectiveness of different linear combinations of heuristics.

Metaheuristic algorithms based on local search are another important class of algorithms. Local search techniques explore the search space of an optimization problem by beginning with an initial solution and iteratively moving to neighbor

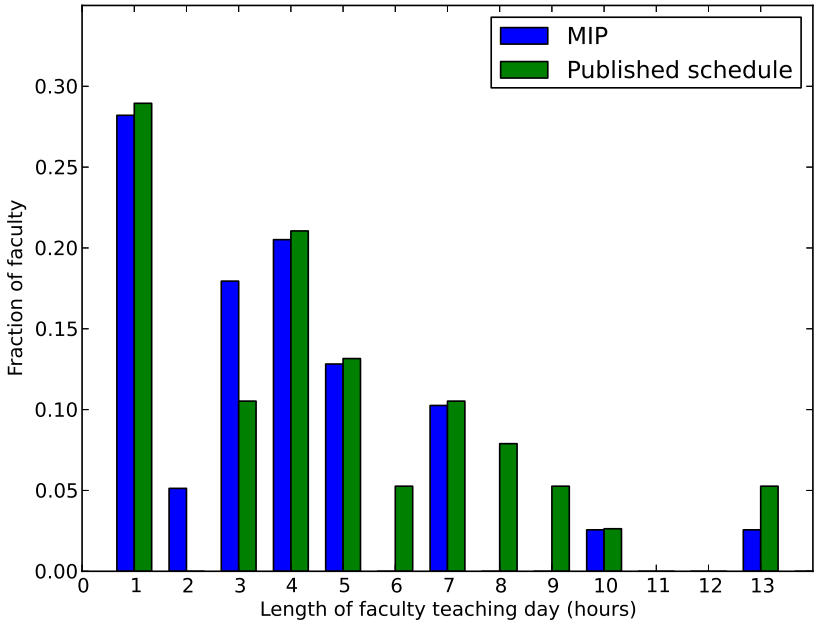

Fig. 7 Distribution of lengths of faculty teaching days in the Fall 2017 schedule

solutions. Our system incorporates a hill-climbing-based local search as part of its post-construction improvement phase.

[27] and [3] presented complete timetabling systems based on simulated annealing, a local search strategy that incorporates random movements [20]. [8] specifically considered the application of annealing to improve constructed schedules. [17] applied tabu search to examination timetabling, which enhances simulated annealing by maintaining a list of "tabu" regions that the search has recently explored and should not revisit [18]. Although our current improvement phase algorithm is fast and effective, these methods are more sophisticated and allow the schedule to make a short-term increase in penalty in pursuit of a larger long-term decrease.

Evolutionary and genetic algorithms are another important class of metaheuristics. Inspired by natural selection, a genetic algorithm evolves a "population" of potential solutions by repeatedly combining and mutating higher-performing candidate solutions and pruning lower-performing candidates [22]. Applications to course and exam timetabling are discussed in [1, 4, 14, 25]. [7] employed a *memetic* algorithm, a hybrid approach where members of the population are allowed to improve themselves with local search before the evolutionary process is applied.

The number of required iterations can make evolutionary algorithms slow. One of the key goals of our project is the creation of an *interactive* system, so we have chosen to focus on algorithms that have fast, predictable performance. There remains, however, an interesting possibility of incorporating a graph-based search algorithm, like the one-pass or beam search, into a metaheuristic framework, with the goal of quickly exploring a large part of the search space .

6 Conclusions and Future Work

We have described the design and implementation of a course timetabling system. Our system includes both a primary objective of avoiding conflicts and a secondary objective of creating compact schedules for students and faculty. Experimental comparisons based on actual course schedules from Rollins College have shown that the schedules produced by approximate algorithms can be competitive with exact mixed-integer programming solutions and that our system would improve upon the actual schedules currently being created at Rollins College.

There are several opportunities for future work. First, improving our heuristic algorithms by exploring other approximate solution strategies, such as the tabu search. Local search heuristics could also be applied to our post-construction repair phase. Within the context of the one-pass and beam search algorithms, we are interested in further exploration of heuristics for vertex and color selection. Second, there is the challenge of creating a practical timetabling system that is accessible to non-technical users. Our system has received positive response from both faculty and administrators. We are currently exploring the possibility of constructing the 2019-2020 course schedule for the entire college.

References

1. Abramson, D., Abela, J.: A parallel genetic algorithm for solving the school timetabling problem. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, N.S.W. (1991)
2. Bisiani, R.: Beam search. In: S. Shapiro (ed.) Encyclopedia of Artificial Intelligence, pp. 56–58. John Wiley and Sons (1987)
3. Burke, E., Bykov, Y., Newall, J., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. IIE Transactions **36**(6), 509–528 (2004)
4. Burke, E., Elliman, D., Weare, R.: A genetic algorithm based university timetabling system. In: Proceedings of the 2nd East-West International Conference on Computer Technologies in Education, vol. 1, pp. 35–40 (1994)
5. Burke, E., Elliman, D., Weare, R.: A university timetabling system based on graph colouring and constraint manipulation. Journal of Research on Computing in Education **27**(1), 1–18 (1994)
6. Burke, E.K., Kendall, G., Mısır, M., Özcan, E., Burke, E., Kendall, G., Özcan, E., Mısır, M.: Applications to Timetabling. Handbook of Graph Theory (2004)
7. Burke, E.K., Newall, J.P.: A multistage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation **3**(1), 63–74 (1999)
8. Burke, E.K., Newall, J.P.: Enhancing timetable solutions with local search methods. In: International Conference on the Practice and Theory of Automated Timetabling, pp. 195–206. Springer (2002)
9. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **140**(2), 266–280 (2002)
10. Burke, E.K., Pham, N., Qu, R., Yellen, J.: Linear combinations of heuristics for examination timetabling. Annals of Operations Research **194**(1), 89–109 (2012)
11. Carrington, J.R., Pham, N., Qu, R., Yellen, J.: An enhanced weighted graph model for examination/course timetabling. In: Proceedings of the 26th workshop of the UK Planning and Scheduling Special Interest Group, pp. 9–16 (2007)
12. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society **47**(3), 373–383 (1996)
13. Cheong, C.Y., Tan, K.C., Veeravalli, B.: Solving the exam timetabling problem via a multi-objective evolutionary algorithm-a more general approach. In: 2007 IEEE Symposium on Computational Intelligence in Scheduling, pp. 165–172. IEEE (2007)

14. Corne, D., Ross, P., Fang, H.L.: Evolutionary timetabling: Practice, prospects and work in progress. In: Proceedings of the UK Planning and Scheduling Special Interest Group Workshop. Strathclyde (1994)
15. Datta, D., Deb, K., Fonseca, C.M.: Solving class timetabling problem of iit kanpur using multi-objective evolutionary algorithm. KanGAL, Report **2006006**, 1–10 (2006)
16. De Werra, D.: Graphs, hypergraphs and timetabling. Methods Operational Research (Verlag A. Hain, Königstein) **49**(ROSE-ARTICLE-1985-003), 201–215 (1985)
17. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 104–117. Springer (2000)
18. Glover, F.: Tabu search-part i. ORSA Journal on computing **1**(3), 190–206 (1989)
19. Kiaer, L., Yellen, J.: Weighted graphs and university course timetabling. Computers & Operations Research **19**(1), 59–67 (1992)
20. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., et al.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
21. McCollum, B.: University timetabling: Bridging the gap between research and practice. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 15–35. Springer (2007)
22. Norvig, P.: Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1992)
23. Qu, R., Burke, E., McCollum, B., Merlot, L.T., Lee, S.Y.: A survey of search methodologies and automated system development for examination timetabling. J. of Scheduling **12**(1), 55–89 (2009)
24. Rickman, J., Yellen, J.: Course timetabling using graph coloring and A.I. techniques. In: Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling (2014)
25. Ross, P., Hart, E., Corne, D.: Genetic algorithms and timetabling. In: Advances in Evolutionary Computing, pp. 755–771. Springer (2003)
26. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review **13**(2), 87–127 (1999)
27. Thompson, J.M., Dowsland, K.A.: A robust simulated annealing based examination timetabling system. Computers & Operations Research **25**(7), 637–648 (1998)
28. Toffolo, T.: Private Communication. University of Ouro. Preto, Brazil (2015)
29. Wehrer, A., Yellen, J.: The design and implementation of an interactive course-timetabling system. Annals of Operations Research **218**(1), 327–345 (2014)
30. Welsh, D.J., Powell, M.B.: An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal **10**(1), 85–86 (1967)