Applying Graph Theory and Evolutionary Computation

For A Course Scheduling System

Ruzgar Zere

Senior Thesis for Artium Baccalaureus Honoris

Department of Mathematics and Computer Science, Rollins College

1000 Holt Ave, Winter Park, FL 32789

Abstract

We describe the design and implementation of a course timetabling system. Our system models the timetabling problem as a vertex coloring of a weighted graph, where each edge has a two-component weight corresponding to the two objectives—minimizing conflict and creating compact schedules. Previous work by Wehrer and Yellen (Wehrer and Yellen 2014) resulted in a Java-based system that scheduled the Rollins College Science Division in 2011. (Rickman and Yellen 2014) built upon this framework to explore new scheduling algorithms. As a part of my Student-Faculty Collaborative Research program, (2016) we have reconstructed the system to be more efficient.

We present three coloring algorithms for this graph model: an exact mixed integer programming formulation, one-pass greedy search and beam search. We evaluate the performance of the three algorithms by comparing their performance on a set of randomly generated test problems based on the actual Rollins course schedule. We also discuss the implementation of machine-learning techniques into the existing system with the purpose of fine-tuning the system's parameters.

Finally, we compare our system against the published Rollins College Science Division's Fall 2017, by constructing alternate schedules using our three algorithms based on the survey data from the science divisions. 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 (Myers and Yellen 2018).

1. Introduction

We describe the design and implementation of a course scheduling system for Rollins College. Constructing course timetables for higher education institutions is a challenging problem that requires balancing instructor and student preferences while avoiding conflicts. The underlying model for our system is a weighted graph model with two-component edge weights. The two edge-weights represent our two objectives: avoiding conflict and creating compact schedules. This paper includes the following sections:

- **i.** An overview of the course-timetabling problem and a description of our graph coloring approach that uses two-component edge weights (Section 2)
- **ii.** The different solutions that we use in our research, and an overview of the machine learning approach for adjusting the heuristic relative-weights (Section 3)
- **iii.** The two experimental comparisons—comparing between our algorithms by generating a set of randomized test problems based on Fall 2015 semester schedule at Rollins College, and a comparison between our system's performance and the implemented Fall 2017 Science Division schedule at Rollins College (Section 4)
- **iv.** Visualization of the Fall 2017 Science Division as a graph in which the courses are represented as vertices and course conflicts as edges (Section 5)

2. The Course-Timetabling Problem

2.1. Graph Coloring Approach

The timetabling problem and the graph coloring approach to solving it have a long history. In 1967, Welsh pointed out the "connection between the basic scheduling or timetabling problem with the well-known problem of coloring the vertices of a graph in such a way that no two

adjacent vertices are the same color and (ii) the number of colors used is a minimum". (Welsh) Since then, Many approaches, including heuristic graph coloring (De Werra 1985; Kiaer and Yellen 1992; Burke et al. 1994b, 2004b; Carrington et al. 2007), and metaheuristic algorithms based on local search and evolutionary computation (Abramson and Abela 1991; Corne et al. 1994; Burke et al. 1994a; Thompson and Dowsland 1998; Burke and Newall 1999; Di Gaspero and Schaerf 2000; Burke et al. 2004a) have been used to create course timetables.

2.2. Conflict and Proximity Objectives

A good course schedule must balance two objectives. First, avoiding conflict as much as possible, meaning that we must construct the system so that it avoids assigning courses taught by the same instructor to the same timeslot, assigning two courses to the same room at the same time, or assigning courses with high-student overlap at the same time (e.g. Calculus and Principles of Physics). The second objective is to create compact schedules for students, and faculty. For example, the system avoids assigning one course taught by a faculty member in the morning and a second one taught by the same faculty later in the evening. Our approach is based on a weighted graph model, in which the vertices represent all the unique courses that are offered, and every edge represents a penalty based on overlap and proximity factors.

The first weight corresponds to the conflict penalty that is incurred if the two endpoints of the edge are assigned to overlapping timeslots. We have divided 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 a large number of students.
- Medium — representing conflicts that are not prohibitive but desirable to avoid, students

might be taking these courses in the same semester, but are likely not required to do so.

- Light — representing conflicts that affect only a small number of students.

The second weight represents the proximity penalty. Since our goal is to create compact schedules for the greatest number of students and faculty, it is reasonable that gaps between courses with overlapping members should make a greater contribution to the overall proximity penalty that courses that have few members in common. This is why the proximity penalty takes into account the overlap factor in addition to the gap factor. For each course pair, the proximity penalty is calculated by

$$p(v_1, v_2) = overlap(v_1, v_2) \cdot gap(v_1, v_2)$$

The overlap factor is based on the conflict penalty and is assigned an arbitrary weight that we have decided on based on the heavy, medium or light penalties. The gap factor is calculated based on the time difference between the assigned timeslots for courses v_1 and v_2.

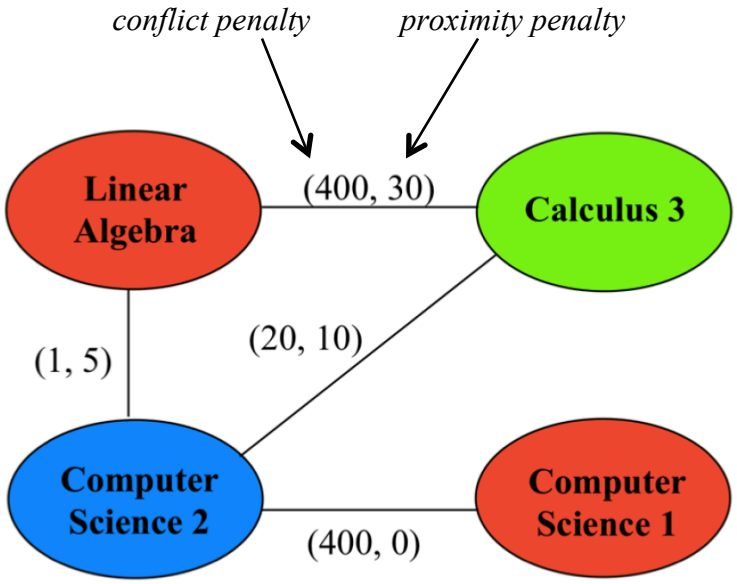

Figure 1. **The Weighted-Graph Model Example**

There is a heavy conflict severity between Computer Science 1 and Computer Science 2, which might indicate that they share a required resource, such as the same special room. The three time slots are assigned to three different colors, and the gap factor is calculated as the absolute difference between the beginning times of the courses. For example, the total conflict penalty between Linear Algebra and Calculus 3 is calculated as 400 because it is possible that the same instructor teaches both classes, and the overlap factor as 30, because these two courses are taken in the same semester by all math majors.

3. Solutions

3.1. Mixed Integer Programming (MIP)

An exact mixed-integer programming (MIP) formulation for our dual-objective graph model has been developed in 2015. (Toffolo, 2015) The MIP formulation yields optimal results, but suffers from long runtimes in certain instances and has a high variability for different problems (4 minutes to 3 hours). In addition, many high-quality MIP solvers require purchasing a commercial license, which not all organizations are willing or able to pursue. Lastly, the MIP interface does not allow us to make alterations to already constructed schedules without rescheduling the entire timetable and not easily adaptable to the changes that could occur during the creation of an actual course-timetable simply because re-running the exact solver takes a long time. All of these considerations pointed us in the direction of pursuing much faster, heuristic approaches to this problem.

3.2. One-Pass Greedy Search

One pass is a search algorithm that explores the space of possible colorings for a given

graph model, and the search space is represented as a tree. The root of the search tree is the initial uncolored graph, internal nodes of the tree represent partial colorings, and leaves are complete colorings. We construct the overall course schedule by repeating two steps. The first step is to find the most troublesome course, which is the vertex in the graph that is the most likely to cause problems, or has many edges. The second step is to pick a timeslot that is the most suitable and has the least impact on the neighboring vertices or neighboring courses, trying avoiding an increase in previously good penalties. We simply repeat this procedure from root to leaf until every course is scheduled to a timeslot or in other words, every vertex in the graph is colored. The one-pass solution makes only one scheduling decision at each step, without considering alternate choices or backtracking. Therefore, solution algorithms that explore a larger portion of the search space may obtain better results, with a tradeoff between the desire to thoroughly explore the search space and the desire for fast solution generation.

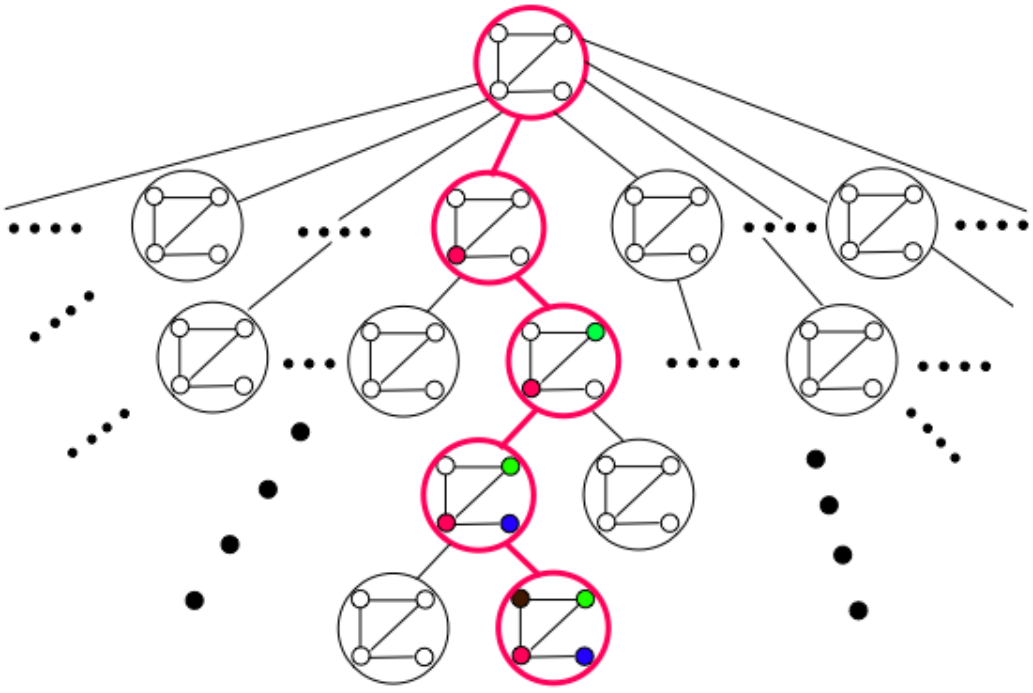

Figure 2. **Root to leaf one-pass**

3.3. Beam Search

Beam search is a restricted tree search, in which the amount of memory available for storing the set of alternative search nodes is limited, and non-promising nodes can be pruned at any step in the search (Zhang 1999). The pruning of non-promising nodes is determined by problem-specific heuristics (Zhang 1999). The set of most promising, or best alternative, search nodes is called the "beam" (Xu and Fern 2007). At each iteration, we generate two or more children of the most promising node encountered so far. A heuristic evaluation of how promising a given node (partial coloring) is, uses a linear combination based on various aspects of the node:

- The penalty of the partial coloring
- Heuristic evaluating the difficulty of coloring the remaining vertices
- Total Bad Value of Colors – the sum of the bad value of colors for all uncolored vertices
- Total Bad Value of Edges between uncolored vertices

The number of children generated is called the branching factor. Because the search tree grows exponentially, we use a relatively small branching factor by selecting a small number of vertices to color and selecting a small number of colors for each vertex chosen to avoid spending too much time searching. We select the next node to expand by using the heuristic evaluation, and a fixed-size priority queue keeps track of promising nodes. Less-promising nodes are discarded when the queue is full. A promising node is one that appears likely to have a leaf-node descendent with a low total penalty. If the branching factor is 1, then the algorithm effectively reduces to a one-pass. Beam search also allows for backtracking to different parts of the search tree. The current most promising node is always at the front of the queue so it's always the next one expanded. As mentioned above, heuristics are used in various components of the beam search: ordering the priority queue; setting the branching factor, and; selecting the child nodes

that are generated during each expansion. A challenge in beam search–as well as one pass–has been generating heuristic weights that increase performance for different problems across the board.

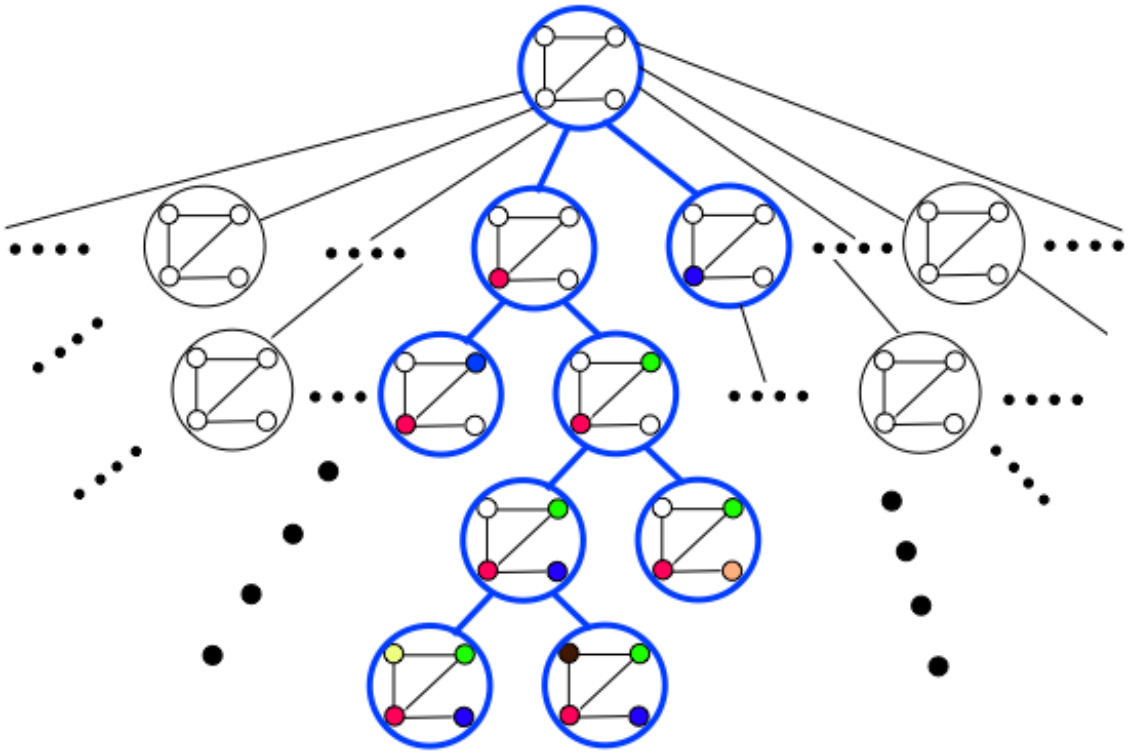

Figure 3. **Beam search with branching factor = 2**

3.4. Machine Learning

In order to adjust the relative weights of the heuristics to yield closer-to-optimal results we have used machine-learning techniques, and "trained" the weights over a set of training problems. The weights for the different heuristics were tuned using an evolutionary local-search algorithm (Norvig 1992). The tuned beam search achieved the same result as the optimal MIP scheduler on the Fall 2011 Science Division problem. (Yellen 2016)

We utilized the Distributed Evolutionary Algorithms in Python (DEAP) library and adapted it to our own Python framework, which improved our average results by lowering the total conflict penalties. “DEAP is an evolutionary computation framework for prototyping and testing of ideas. Its design departs from most other existing frameworks in that it seeks to make algorithms explicit and data structures transparent, as opposed to the more common black-box frameworks.” (Fortin et. al. 2012)

The DEAP framework adjusts the scheduling algorithms’ heuristic parameters using an evolutionary genetic algorithm, which is a search heuristic that mimics the process of natural selection. The algorithm trains the heuristic weights for both the one-pass and beam search by generating an initial random population of potential weight vectors, then scoring each member of that population based on its performance on our set of randomized test problems. Poorly performing weights are pruned from the population and new candidate solution are generated by combining—*crossing over,* in genetic algorithm terminology—the high performers. The result is a new generation of candidate solution vectors, which has, on average, higher overall fitness than the previous generation. This process repeats until the population reaches a local optimum.

4. Experimental Comparison

4.1. Generating Randomized Problems

To test our system, we generated a set of randomized test problems based upon a real semester schedule at Rollins College. We acquired the Fall 2015 registration information from the Rollins registrar, which included 711 individual course sections, 157 unique meeting times (e.g., MWF from 8:00-8:50, MW 8-9:15, etc.), and more than 300 instructors, as well as student cross-enrollment data. Building upon the actual Fall 2015 schedule, we generated a set of test

problems by randomizing the acceptable timeslots, acceptable rooms, and student cross-enrollment weights between the course pairs. The randomized problems are constructed in such a way that every course has at least one valid timeslot and room assignment.

To create the randomized test problems, we:

- Assigned each course a randomized selection of acceptable timeslots based on its actual assigned timeslot – the same number of meeting days and minutes per week
- Assigned a randomized selection of suitable rooms in the same building/department. Lab, studio, and performance courses were only assigned suitable locations.
- Randomized the strength of the intersection between each pair of courses by increasing or decreasing the estimated number of cross-enrolled students. In some cases, this resulted in the removal of edges if the intersection between a course pair dropped to zero. With a small probability, we added new edges between course pairs that had no intersection in the original schedule. (Yellen 2016)

4.2. Performance of the Three Algorithms

Table 1. Results of the MIP scheduler on ten randomized problems

Problem	Total penalty	Heavy conflicts	Medium conflicts	Light conflicts	Unassigned rooms	Runtime (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. Performance of the one-pass scheduler on ten randomized test problems

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

Table 3. Performance of the beam search scheduler on ten randomized test problems

Problem	Total penalty	Heavy conflicts	Medium conflicts	Light conflicts	Unassigned rooms
1	15299	0	1	59	0
2	16838	0	3	53	0
3	17153	0	4	50	0
4	16135	0	2	46	0
5	19680	0	6	49	1
6	18973	0	6	55	0
7	18304	0	5	36	0
8	20084	0	6	44	0
9	16944	0	4	32	0
10	17246	0	2	59	0

Table 1 shows the performance of the MIP scheduler using the Gurobi solver on a set of 10 randomized test problems based on the Fall 2015 schedule. The MIP scheduler was able to assign a suitable room for every course while avoiding creating any heavy conflicts. The average penalty for the ten problems for the exact solver is 14138. The performance of the MIP scheduler on the Fall 2015 problems is encouraging, but there are still reasons to investigate heuristic schedulers. Our experience and discussions with the college's registrar have shown that the course schedule must always be manually edited after being initially constructed (Yellen 2016). MIP, being comparably quite slow is not time-efficient considering that the solver would have to

be run multiple times. The heuristic approach is much more time-mindful in the face of manual changes to the schedule.

Table 2 gives the results for the one-pass system on the set of 10 randomized problems. The average penalty for the 10 problems is 18488. The resulting schedules have no heavy conflicts, and only four courses are left without a room. The one-pass system is extremely fast, considering that our current Python implementation constructs the entire 700-course schedule in less than three seconds.

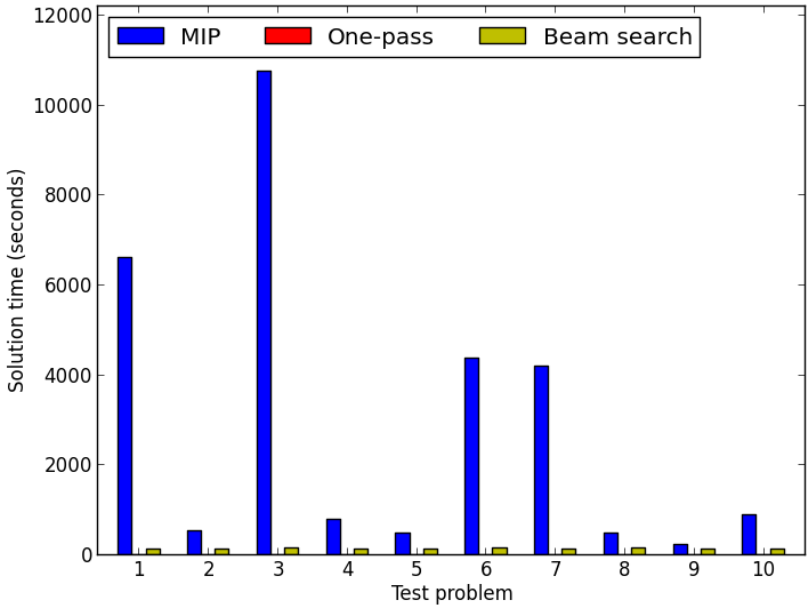

Figure 4. **Runtime on the 10 test problems** (Yellen 2016)

Figure 4 demonstrates the gap in the runtimes of MIP, one-pass and beam search algorithms. The MIP algorithm gives an exact solution, with the tradeoff of long runtimes that varies from 4 minutes to 3 hours. Both of our beam search and one-pass approaches are significantly faster than the exact solution approach. The beam-search scheduler is slower than the one-pass approach, but its optimal performance in constructing the Fall 2011 Science Division schedule motivated its application to the 700-course problem. Table 3 reports results on the 10 randomized problems for a version of the beam search algorithm that adapts the vertex

and timeslot-room selection heuristics from the one-pass system. The heuristics for selecting the partial coloring to expand at each step have been tuned using evolutionary local search. The average for the ten problems is 17665, which is within 25% of the optimal result obtained by the MIP scheduler. The beam search system is considerably faster than the MIP solver, yet slower compared to the one-pass approach. The reason behind this is that even with a good priority function, our Python implementation of the priority queue algorithm can still do a great deal of backtracking. This makes the algorithm slow due to the large size of the search tree. We can mitigate this problem by limiting the branching factor during the branching phase of the algorithm.

4.3. Comparison to the Fall 2017 Science Division Schedule

Finally, we compare the results produced by our algorithms to the actual course timetable for the Rollins College Science Division in the fall semester of 2017. At Rollins College course scheduling is handled manually, each department constructing their own schedules. The conflicts that arise based on the division schedules such as room overlaps are handled by the College's registrar. Department chairs are knowledgeable about the conflicts within their own programs. However, they might overlook cross-departmental conflicts, and compact schedules. Because of this, constructing the final schedule needs manual rework by the registrar.

In order to recreate the Fall 2017 schedule as accurately as possible using our algorithms, we surveyed the members of the six Science Division departments—biology, chemistry, computer science, mathematics, physics, and psychology. Our medium for the survey was a combination of Excel surveys and physical copies. We gathered information on the courses offered as well as the list of acceptable timeslots and room assignments. We also gathered

information on courses that should not be scheduled in overlapping timeslots based on faculty's perception and used it for determining the conflicts between courses–or the edge weight between vertices.

The resulting data contained 124 total course sections taught by 44 faculty members in 65 different acceptable timeslots. The faculty surveys identified 750 total conflicts: 169 heavy conflicts, 532 total medium conflicts, and 49 light conflicts.

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

Table 4. **Comparison of actual and optimized Fall 2017 science schedules** (Myers and Yellen 2018)

Table 4 shows that all three scheduling algorithms produce results that improve upon the actual course schedule. All of our algorithms had runtimes that were less than 4 minutes. The optimal MIP schedule contains 7 heavy conflicts, compared to the published schedule, which contained 20 heavy conflicts. We also show that all of our coloring algorithms produced schedules with fewer conflicts than the actual schedule. The Beam-search scheduled a timetable with 46 total conflicts, compared to the published schedule that contained 80 total conflicts. This is an impressive result considering that our input contained 750 conflicts between different courses. The results suggest that our approach significantly reduced conflicts compared to the published schedule yet despite our best efforts, did not eliminate conflict between related courses. This could be the result of an underrepresentation of the suitable rooms and suitable timeslots.

5. Graph Visualization

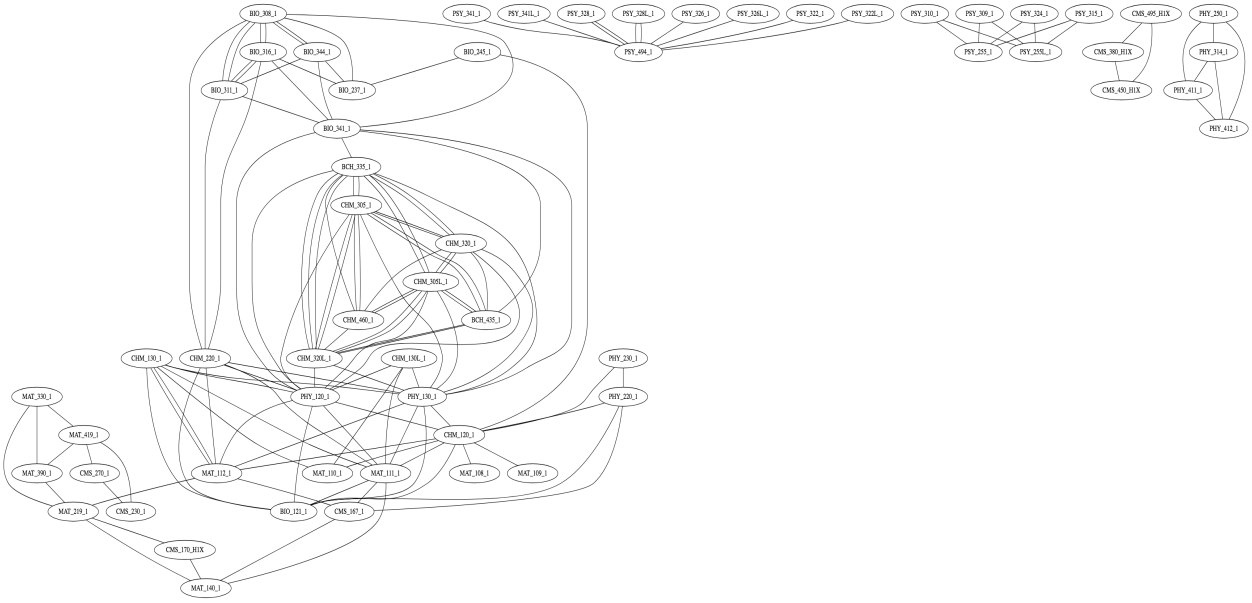

Figure 5. **Graph for Fall 2017 Science Division**

The graph above visualizes the Fall 2017 Science Division courses and conflicts. Different sections of the same course and their respective labs are consolidated into one vertex. Each edge represents a light, medium or heavy conflict. The graph is disconnected and contains 5 components. We have used the GraphViz graph visualization software to produce this graph based on our list of course conflicts. (Appendix) The graph shows that all 100-level science courses are connected, meaning that they have a conflict designation with at least one other course. Some of the Mathematics courses are required for most science majors, so this is not surprising. The smaller components are composed of higher-level courses within the same major. This shows that it is often harder to schedule intro level classes because they have a higher number of conflicts with other courses, and in result have a higher degree. By visualizing the graph, we have also shown the complexity of the problem and the need for our heuristic approach.

6. Conclusion

We have described the design and implementation of a course timetabling system. Our heuristic graph coloring approach has proved itself fruitful in the course-scheduling domain. MIP, one-pass, as well as the beam search approaches produced impressive results on our test problems. We have demonstrated that our heuristic approach is competitive with the exact Mixed-Integer solution in terms of the final conflicts and much faster in terms of runtime. We also showed that our system improved upon the actual Fall 2017 Science Division schedule, reducing the number of conflicts by roughly one half and creating more compact schedules for students and faculty.

One of the challenges that we ran into was the gathering of survey data from different departments across campus. It is not a simple task to get faculty and school to buy-in and collaborate on providing the data required for our scheduling system. The data-gathering procedure could possibly be improved upon by creating an enhanced survey. This could simplify the process of completing the survey, and the collection of our data. In regards to the timetabling algorithms, future research could focus on further improving our heuristic algorithms by exploring other approximate solution strategies.

References

Abramson, D. and Abela, J. (1991). 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.

Burke, E. K. and Newall, J. P. (1999). A multistage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation, 3(1):63–74.

Burke, E. K. and Petrovic, S. (2002). Recent research directions in automated timetabling. European Journal of Operational Research, 140(2):266–280.

Corne, D., Ross, P., and Fang, H.-L. (1994). Evolutionary timetabling: Practice, prospects and work in progress. In Proceedings of the UK Planning and Scheduling Special Interest Group Workshop, Strathclyde.

De Werra, D. (1985). Graphs, hypergraphs and timetabling. Methods Operational Research (Verlag A. Hain, Königstein), 49(ROSE-ARTICLE-1985-003):201–215.

Di Gaspero, L. and Schaerf, A. (2000). Tabu search techniques for examination

timetabling. In Proceedings of the International Conference on the Practice and

Theory of Automated Timetabling, pages 104–117. Springer.

Fortin F.A., De Rainville F.M., Gardner M.A., Parizeau M, & Gagné C. (2012) DEAP: Evolutionary Algorithms Made Easy, Journal of Machine Learning Research 13 2171-2175

Kiaer, L. and Yellen, J. (1992). Weighted graphs and university course timetabling. Computers & Operations Research, 19(1):59–67.

Myers, D. and Yellen J. (2018) A Multi-Objective Timetabling System That
Facilitates Scheduling Across Academic Programs. Manuscript for PATAT 2018

Norvig. P. (1992) Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Rickman J. and Yellen J. (2014) Course timetabling using graph coloring and a.i. techniques. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, Springer

Schaerf, A. (1999). A survey of automated timetabling. Artificial Intelligence Review, 13(2): 87–127.

Thompson, J. M. and Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. Computers & Operations Research, 25(7): 637–648.

Toffolo T (2015) Private communication.

Welsh, D. J. and Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal, 10(1): 85–86.

Wehrer A, Yellen J (2014) The design and implementation of an interactive course-timetabling system. Annals of Operations Research 218(1): 327–345

Xu Y, Fern, A (2007) On learning linear ranking functions for beam search. Proceedings of the 24th international conference on Machine learning - ICML '07

Yellen J (2016) Comparing Exact and Heuristic Algorithms for a Course-Timetabling Problem. Manuscript for PATAT 2016

Zhang A (1999) State-space search: Algorithms, complexity, extensions, and applications. Springer: New York.

Appendix

```
graph {
CHM_130_1 -- MAT_110_1;
CHM_130_1 -- MAT_111_1;
CHM_130_1 -- MAT_112_1;
CHM_130_1 -- PHY_120_1;
CHM_130_1 -- PHY_130_1;
CHM_130L_1 -- MAT_110_1;
CHM_130L_1 -- MAT_111_1;
CHM_130_1 -- MAT_112_1;
CHM_130L_1 -- PHY_120_1;
CHM_130L_1 -- PHY_130_1;
CHM_220_1 -- MAT_111_1;
CHM_220_1 -- MAT_112_1;
CHM_220_1 -- PHY_120_1;
CHM_220_1 -- PHY_130_1
CHM_220_1 -- BIO_121_1;
CHM_130_1 -- BIO_121_1;
MAT_111_1 -- BIO_121_1;
CHM_305_1 -- CHM_320_1;
CHM_305_1 -- CHM_320L_1;
CHM_305_1 -- CHM_460_1;
CHM_305_1 -- BCH_335_1;
CHM_305_1 -- BCH_435_1;
CHM_305_1 -- PHY_120_1;
CHM_305_1 -- PHY_130_1;
CHM_305L_1 -- CHM_320_1;
CHM_305L_1 -- CHM_320L_1;
CHM_305L_1 -- CHM_460_1;
CHM_305L_1 -- BCH_335_1;
CHM_305L_1 -- BCH_435_1;
CHM_305L_1 -- PHY_120_1;
CHM_305L_1 -- PHY_130_1;
CHM_320_1 -- CHM_305_1;
CHM_320_1 -- CHM_305L_1;
CHM_320_1 -- BCH_335_1;
CHM_320_1 -- PHY_120_1;
CHM_320_1 -- PHY_130_1;
CHM_320L_1 -- CHM_305_1;
CHM_320L_1 -- CHM_305L_1;
CHM_320L_1 -- BCH_335_1;
CHM_320L_1 -- PHY_120_1;
CHM_320L_1 -- PHY_130_1;
CHM_460_1 -- CHM_305_1;
CHM_460_1 -- CHM_305L_1;
CHM_460_1 -- CHM_320_1;
CHM_460_1 -- CHM_320L_1;
BCH_335_1 -- CHM_305_1;
BCH_335_1 -- CHM_305L_1;
BCH_335_1 -- CHM_320_1;
BCH_335_1 -- CHM_320L_1;
BCH_335_1 -- CHM_320L_1;
BCH_335_1 -- CHM_460_1;
BCH_335_1 -- PHY_120_1;
BCH_335_1 -- PHY_130_1;
BCH_435_1 -- CHM_305_1;
BCH_435_1 -- CHM_305L_1;
BCH_435_1 -- CHM_320_1;
BCH_435_1 -- CHM_320L_1;
BCH_435_1 -- CHM_320L_1;
PSY_341_1 -- PSY_494_1;
PSY_341L_1 -- PSY_494_1;
PSY_328_1 -- PSY_494_1;
PSY_328L_1 -- PSY_494_1;
PSY_326_1 -- PSY_494_1;
PSY_326L_1 -- PSY_494_1;
PSY_328_1 -- PSY_494_1;
PSY_328L_1 -- PSY_494_1;
PSY_324_1 -- PSY_255_1;
PSY_324_1 -- PSY_255L_1;
PSY_322_1 -- PSY_494_1;
PSY_322L_1 -- PSY_494_1;
PSY_315_1 -- PSY_255_1;
PSY_315_1 -- PSY_255L_1;
PSY_310_1 -- PSY_255_1;
PSY_310_1 -- PSY_255L_1;
PSY_309_1 -- PSY_255_1;
```

PSY_309_1 -- PSY_255L_1;
CMS_167_1 -- MAT_140_1;
CMS_380_H1X -- CMS_450_H1X;
CMS_170_H1X -- MAT_140_1;
CMS_270_1 -- CMS_230_1;
CMS_495_H1X -- CMS_380_H1X;
CMS_495_H1X -- CMS_450_H1X;
MAT_111_1 -- CMS_167_1;
MAT_111_1 -- MAT_140_1;
MAT_112_1 -- MAT_219_1;
MAT_112_1 -- CMS_167_1;
MAT_219_1 -- MAT_140_1;
MAT_219_1 -- CMS_170_H1X;
MAT_330_1 -- MAT_219_1;
MAT_330_1 -- MAT_419_1;
MAT_330_1 -- MAT_390_1;
MAT_419_1 -- MAT_390_1;
MAT_419_1 -- CMS_230_1;
MAT_419_1 -- CMS_270_1;
MAT_390_1 -- MAT_219_1;
PHY_130_1 -- MAT_111_1;
PHY_130_1 -- MAT_112_1;
PHY_130_1 -- CHM_120_1;
PHY_130_1 -- BIO_121_1;
PHY_120_1 -- MAT_111_1;
PHY_120_1 -- MAT_112_1;
PHY_120_1 -- CHM_120_1;
PHY_120_1 -- BIO_121_1;
CHM_120_1 -- MAT_108_1;
CHM_120_1 -- MAT_109_1;
CHM_120_1 -- MAT_110_1;
CHM_120_1 -- MAT_111_1;
CHM_120_1 -- MAT_112_1;
CHM_120_1 -- BIO_121_1;
PHY_220_1 -- CMS_167_1;
PHY_220_1 -- CHM_120_1;
PHY_220_1 -- BIO_121_1;
PHY_230_1 -- CHM_120_1;
PHY_230_1 -- PHY_220_1;
PHY_250_1 -- PHY_314_1;
PHY_250_1 -- PHY_411_1;
PHY_250_1 -- PHY_412_1;
PHY_314_1 -- PHY_411_1;
PHY_314_1 -- PHY_412_1;
PHY_411_1 -- PHY_412_1;
BIO_341_1 -- BCH_335_1;
BIO_341_1 -- BCH_435_1;
BIO_341_1 -- BIO_308_1;
BIO_341_1 -- PHY_120_1;
BIO_341_1 -- PHY_130_1;
BIO_341_1 -- BIO_316_1;
BIO_308_1 -- CHM_220_1;
BIO_308_1 -- BIO_311_1;
BIO_308_1 -- BIO_316_1;
BIO_308_1 -- BIO_237_1;
BIO_308_1 -- BIO_344_1;
BIO_316_1 -- BIO_308_1;
BIO_316_1 -- BIO_311_1;
BIO_316_1 -- BIO_237_1;
BIO_316_1 -- CHM_220_1;
BIO_311_1 -- BIO_308_1;
BIO_311_1 -- BIO_341_1;
BIO_311_1 -- BIO_316_1;
BIO_311_1 -- CHM_220_1;
BIO_344_1 -- BIO_308_1;
BIO_344_1 -- BIO_311_1;
BIO_344_1 -- BIO_341_1;
BIO_344_1 -- BIO_237_1;
BIO_245_1 -- BIO_237_1;
BIO_245_1 -- CHM_120_1;
}