# Using Visualization and Integer Linear Programming for University Class Scheduling

by

**Kristin Bushman**

Bachelor of Science, University of Pittsburgh, 2018

Submitted to the Graduate Faculty of

the Department of Computer Science in partial fulfillment

of the requirements for the degree of

**Master of Sciences**

University of Pittsburgh

2021

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This thesis was presented

by

Kristin Bushman

It was defended on

July 7th 2021

and approved by

Alexandros Labrinidis, Department of Computer Science

Panos Chrysanthis, Department of Computer Science

Kirk Pruhs, Department of Computer Science

Thesis Advisor: Alexandros Labrinidis, Department of Computer Science

**Using Visualization and Integer Linear Programming for University Class Scheduling**

Kristin Bushman, M.S.

University of Pittsburgh, 2021

University class scheduling is the task of assigning a room, instructor, and timeslot to each class in a university schedule. This is a highly-combinatorial task, with multiple constraints and goals. Often, these goals are in competition with each other, meaning improving one metric may deteriorate another. Traditional scheduling workflows have involved the use of spreadsheets and whiteboards to keep track of assignments. These tools may be adequate for small-size schedules, however, as the size and complexity of the schedule increases, these tools become increasingly difficult to use. In those cases, schedules may contain errors or undesirable assignments, such as room conflicts or class conflicts; it is also very difficult to evaluate the quality of the resulting schedule. In this thesis, we introduce a web-based tool that can be used to support the scheduling process. The tool includes multiple ways to visualize a schedule. These visualizations can help the user to quickly identify conflicts or problem areas. Users can make changes to instructor, room, or timeslot assignments and quickly reassess the quality of the resulting schedule. By utilizing the calendar paradigm, making changes to the current schedule is a very intuitive process. Further, we provide the ability for faculty to provide their teaching preferences (in terms of which courses to teach and also which days/times); this information is integrated with the rest of the schedule making it very easy to identify good assignments of instructors to courses. Finally, our tool has an automated scheduling feature, which allows the system to make all scheduling decisions rather than the user. The automated scheduler uses an integer linear programming model to describe the scheduling problem and its constraints. The linear programming model is optimized to reflect multiple scheduling goals that are required by the Computer Science Department of the University of Pittsburgh. We experiment with different ways to use the automated scheduler, specifically with respect to combining and prioritizing different metrics.

## Table of Contents

## List of Tables

## List of Figures

## Preface

I would like to thank my advisor, Alex Labrinidis, for providing me with the opportunity to work on this research and for his guidance and feedback throughout the process. Additional thanks to the other members of the committee, Panos Chrysanthis and Kirk Pruhs, for their time and helpful feedback on the presentation and write-up. I would also like to thank Heidi Davis and the countless other individuals at the University of Pittsburgh who participated in demos and provided feedback on the class scheduling tool throughout its development.

I am also grateful to the Pitt Smart Living project for providing me with the opportunity to work in the ADMT research lab and pursue my Master's degree. The Pitt Smart Living project is supported by NSF award CNS-1739413. Finally, I would like to thank my family and friends for their endless support throughout graduate school and the writing of this thesis.

## 1.0 Introduction

University class scheduling is the task of assigning the instructors, rooms, and timeslots to each of the classes offered in a university department. Creating these schedules by hand is a difficult and time-consuming task. There are many stakeholders whose preferences must be considered when building the schedule, including students, instructors, and the university or department. Organizing these preferences and incorporating them into the schedule is often a manual task. This manual effort can lead to errors in the schedule, which must be later caught and resolved.

Beyond ensuring the feasibility of a schedule, there are many different criteria which must be considered to evaluate the quality of the schedule. Due to the highly combinatorial nature of the class scheduling problem (NP-hard), there is no efficient way to compute all possible schedules and compare the quality. Another difficulty is that the scheduling metrics are often in competition with each other, thus improving one metric may worsen another. Since, there is not a single schedule that simultaneously optimizes all metrics, multiple pareto-optimal solutions exist. Schedule administrators must use their judgement to determine the relative importance of each metric in order to decide the overall best schedule.

Existing tools for building schedules in the University of Pittsburgh Computer Science department include spreadsheets, whiteboards, emails, and sticky-notes. With information in multiple locations, it is not uncommon for preferences to be missed and mistakes to be made. While minor adjustments of the schedule are possible using these tools, it is difficult to make large-scale changes. One of the main challenges is being able to visualize the schedule in a manner that allows for the easy identification of conflicts or problem areas.

Due to the complexity of the class scheduling problem, it can be beneficial to use software to automate schedule generation rather than creating schedules by hand. Integer linear programming (ILP) is a commonly used technique for solving scheduling problems in software. In the linear programming approach, a mathematical model is formulated to describe the constraints and objectives of the scheduling problem. The objective function and the constraints must be linear. ILP is a subset of linear programming that requires all variables

to be integer values. An optimization solver can find solutions (i.e. instructor, room, and instructor assignments) to the model such that the constraints are satisfied and the objectives are optimal.

The key contributions of this thesis are as follows:

- **Developed a web-based scheduling tool for visualizing and editing schedules:** The tool allows schedules to be visualized from several different perspectives or views. Each view is designed to allow the schedule administrator to evaluate a different aspect of the schedule quality. The administrator can make manual changes to a schedule and quickly reevaluate the quality of the resulting schedule.
- **Developed an automated scheduler feature:** The automated scheduler shifts some of the burden of making scheduling assignments from the user to the system. The automated scheduler uses integer linear programming to make assignments based on the objectives that are configured by the user. Our automated scheduler includes some objectives that were not considered in previously existing systems.
- **Experimentally evaluated the automated scheduler when working with multiple objectives:** We compare results from the automated scheduler under different configurations of objectives. These experiments help elucidate the trade-offs of the different approaches for optimizing multiple objectives. Additionally, the experiments show how certain objectives interact or conflict with one another.

The remainder of this thesis is organized as follows: Chapter 2 reviews related research. Chapter 3 describes the interface for visualizing and manually editing schedules. Chapter 4 describes the automated schedule generator, including its implementation and interface. Chapter 5 defines the constraints and objectives for the scheduling problem, as well as their linear programming formulations. In Chapter 6, the automated scheduling feature is evaluated using several experiments. Finally, Chapter 7 contains conclusions and future work.

## 2.0 Related work

There has been extensive research done on the university class scheduling problem [12]. Several variations of this problem exist, each reflecting the different workflows of educational institutions. In most cases, schedules are created by assigning rooms, timeslots, and instructors to courses [6, 3, 5, 10]. However, in some variations instructors are pre-assigned to courses, thus do not need to be assigned during the scheduling process [16]. University examination timetabling [7] and school class scheduling [4] are problems that are closely related to the university class scheduling problem. Although these problems have slightly different scheduling requirements, the approaches used to solve them are often similar to those used to solve the university class scheduling problem.

In the university class scheduling problem, the instructor, room, and timeslot assignments that are made for each course are subject to several hard constraints which must be adhered to. Ensuring that only one class occupies a room at a single time, ensuring that each instructor teaches only one class at a single time, and abiding by room capacity limits are constraints that are common to all institutions [12]. Other constraints such as observing instructor time availability [16] and avoiding time conflicts for courses that are often taken together [17], may also be included depending on the needs of the institution.

Additional scheduling goals are less strict. These are formulated as objectives rather than hard constraints. Common objectives include maximizing instructor course preferences, minimizing class conflicts for students, and ensuring good room utilization [12]. Since there are often multiple schedules that adhere to the hard constraints, these objectives are used to differentiate the feasible schedules in terms of quality.

### 2.1 Optimization techniques

Although there have been many different approaches [12] to solving the class scheduling problem, integer linear programming is the most commonly used technique in the litera-

ture [16, 17, 3, 13, 6, 19, 15, 4, 7]. These works differ in the scheduling objectives that are considered, however their general approach using ILP are very similar. Metaheuristic optimization methods have also been explored for the class scheduling problem. Examples of these include genetic algorithms [10], particle swarm optimization [14], and harmony search [1].

The most common ILP formulation for the class scheduling problem uses binary decision variables to represent assignments [3, 15, 16, 13]. Each decision variable corresponds to a course being assigned to a particular pattern. A pattern includes an assignment for each attribute, such as the timeslot, room, and instructor. Additional model variables are often created to help represent the various constraints and objectives in the linear programming model.

One of the challenges of the linear programming formulation is handling multiple objectives. This is often done by combining the multiple objectives into a single objective using a weighted sum. The weights for each objective correspond to its relative importance. In [16], weights were determined by a vote from the faculty to reflect their opinions on the relative importance of each objective. In [15], Analytic Hierarchy Process (AHP) and the Analytic Network Process (ANP) were used to assign weights. AHP and ANP allow various stakeholders to make a series of pairwise comparisons between metrics. These comparisons are then used to compute weights.

Another way to handle the complexities of multiple objectives is to decompose the scheduling process into multiple stages. This allows fewer metrics to be considered during a single stage and eliminates interaction between the metrics of different stages. Al-Qaheri [2] broke the scheduling problem into three separate stages: the faculty-course assignment stage, then the courses-timeslot assignment stage, and finally the timeslot-room stage. Goal programming was used to optimize multiple objectives relating to each stage. The decomposition of the problem into separate stages greatly reduced the complexity of the problem and thus significantly improved the computation time. Even with the reduced complexity, the authors were able to achieve high-quality schedules.

## 2.2 Decision support systems for university class scheduling

Software systems are often useful for supporting the scheduling process. Most existing work on class scheduling focuses on the optimization algorithms, however, some research has also been done on the development of interactive decision support systems (DSS) that allow the optimization algorithms to be utilized in practice. Early instances of such systems were PC-based applications [9, 7]. These early systems allowed the user to input information regarding courses, timeslots, etc. The system would then generate a schedule using integer linear programming and return a report of the results to the user.

A more recent system, the udpSkeduler [16], has a web-based interface. The system allows the user to manage the courses, classrooms, and instructors that will be the input for the solver. Additionally, they can set scheduling parameters, view previous schedules, and generate reports. Instructors are able to log in to the system to submit their time availability and preferences. Once the data is input into the system, the user can submit the model to the optimization module which finds a solution using integer linear programming. The results are returned to the user via a series of reports. The authors note that the udpSkeduler has significantly sped up the scheduling process, helped to eliminate human-errors, and simplified the exchange of information between instructors and the schedule administrators.

Other recent decision support systems have similar interfaces. The primary focus of these systems appear to be facilitating data input for the ILP model, while little is done in terms of schedule visualization. Some are PC-based applications [2], while others are web-based [18, 8]. One system [11] uses a Microsoft Excel spreadsheet as the user interface which connects to a Python backend.

The DSS presented in this thesis has several features that distinguish it from the previous work. First, the interface contains multiple views for visualizing schedules. Second, the ILP formulation includes additional metrics that were not considered in previous work. Lastly, the system allows for an iterative scheduling workflow that alternates between automated and manual scheduling modes.

## 2.3 Summary

In summary, the university class scheduling problem has been studied extensively over the last several decades. As a result of the varying workflows of different intuitions, there are many variations of the scheduling problem. Although the core idea is the same, different constraints and metrics can be included. Many optimization techniques have been utilized for this problem, however integer linear programming is the most common. Additional work has also been done on decision support systems, which integrate the optimizer into a nice user interface to help support the entire scheduling process.

## 3.0 Schedule visualization and interface for making manual changes

The class scheduling tool allows the user to visualize, as well as make changes to an existing schedule. The interface includes many different ways for the user to view a schedule, with each view highlighting a different aspect of the schedule. This allows them to easily identify problem areas or conflicts. The user can manually change the instructor, room, or timeslot for a class using forms that are accessible from any of the visualizations. After making a change, the visualization will automatically update to reflect the new state. This allows the user to quickly assess the outcome of the change. This chapter will describe the interface for working with an existing schedule. The automated schedule generation feature will be discussed in the next chapter.

### 3.1 Calendar-based views

The most useful views in the tool utilize a grid-like calendar to visualize the schedule. Classes appear as blocks on the grid. The grid makes it very easy to identify conflicts in the schedule. If class blocks are overlapping (i.e. occupying the same cell in the grid), this typically indicates that there is a conflict in the schedule, such as two classes scheduled to be in the same room at the same time. However, there are exceptions where overlap does not indicate conflict, such as with cross-listed courses. Classes that are causing a true conflict are highlighted with a red border to make them very easy to spot.

There are several calendar-based views that are available to the user. The department view shows all classes that are present in the schedule. A screenshot of this view is shown in Figure 1. The grid displays with time along the y-dimension and classrooms across the x-dimension. Each class appears as a block in the column for its assigned room and in the row based on its assigned timeslot. The height of the block corresponds to the class duration. This view can show either 1 day at a time, or the full 5-day week all at once.

The 1-day option is useful for finding conflicts and for identifying courses that are offered

at the same time. Often there are courses that students typically take during the same semester, thus they should not be offered at the same time. The 1-day view makes it easy to identify courses offered at the same time since they will appear in the same row of the grid. Additionally, the 1-day view can be used to spot under-utilized rooms or to find a room that is available at a certain time. Although the 5-day option is often overwhelming with information and difficult for seeing details, it can be useful for getting a high-level overview of the schedule. This provides a clear visualization of how the classes are balanced across different days and times. Typically, departments strive to achieve a good balance across days and times, in order to minimize class conflicts and allow maximum flexibility for the students.

The department view can be further customized using filters. The user can filter to include only certain rooms or class types (lectures, recitations, etc.). This can make it easier for the user to focus in on one aspect of the schedule without getting overwhelmed by other unrelated courses. Another customizable aspect of the calendar is the color scheme. By default, classes are colored blue for undergraduate lectures, green for graduate lectures, and purple for recitations or labs. However, the user is able to create a custom color scheme that best fits their visualization needs.

The other calendar-based views are the instructor view and room view. For these views, the grid displays with days across the x-axis and time across the y-axis. For the instructor view, the user will select an instructor then the calendar will show only the classes that are taught by this instructor. The instructor view makes it easy to find conflicts such as an instructor teaching two courses at the same time. Additionally, it can be used to identify if an instructor was assigned an undesirable schedule, such as too many classes back-to-back without a break. The room view is very similar to the instructor view. The user selects a room, then the calendar will display only the classes that were assigned to that particular room. Room conflicts can be easily caught using this view.

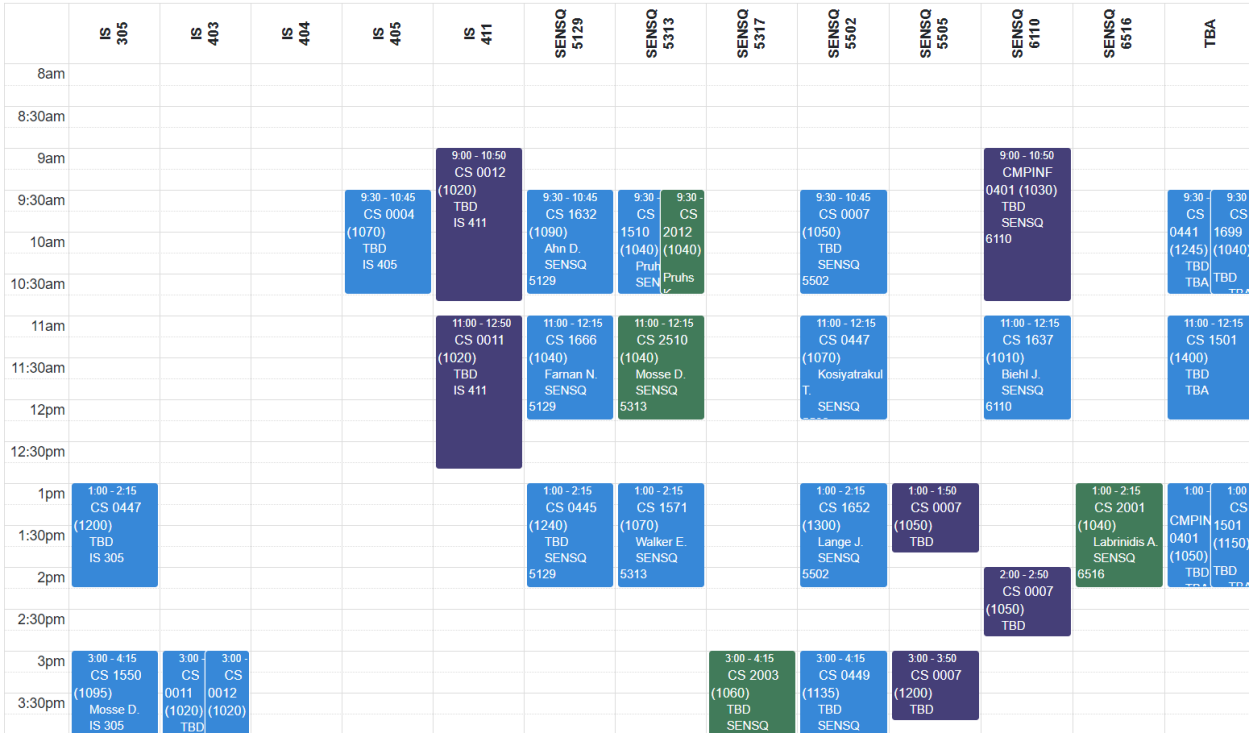


Figure 1: Calendar-based department view

### 3.2 Import and edit

A schedule can be imported into the tool from a CSV file. In the Pitt Computer Science department, the CSV is obtained by downloading a schedule from the SCI Courses site[1]. The CSV file contains a row for each class in the schedule. Important fields include subject, course number, class number, instructor, room, meeting days, and time. Once imported, the data is stored in a database so that the schedule can be accessed at a later time without re-uploading the file. Any changes that are made to the schedule are also stored in the database.

There are two ways to modify an existing schedule. The first is to double-click on a class to open the edit form modal, shown in Figure 2. This form can be used to change any of the class attributes. The other way to modify the schedule is to click on a class in

[1] `http://courses.sci.pitt.edu/`

the calendar-based view, then drag and drop it to a new location on the grid. The drag and drop method allows the user to quickly make changes to the room or time of the class. When a course is moved, any related courses will also move with it. For example, if the user drags the Tuesday instance of a Tu/Th class to a new room, the Thursday instance will also be moved to the same room. This capability also works for cross-listed classes, if the user configures them in the setup menu.

New classes can be added to the schedule via a form identical to the edit form. For bulk additions to the schedule, the user can upload a CSV file of classes. These classes are then appended to the existing schedule.

### 3.3 Interface for instructors

Instructors are able to log in to the system to view schedules. Once the administrator is done editing a schedule, they can enable instructors to view it. When enabled, the instructor will be able to see the classes that they are teaching, presented in the calendar-based instructor view. If an instructor finds any issue with their assigned schedule, they can contact the administrator to request a change.

Instructors are also able specify their teaching preferences within the tool. Some preferences apply to all semesters, while others are semester specific. When administrators are viewing an instructors individual schedule using the instructor view, the preferences for that instructor will appear on the same page as the calendar. This allows the administrator to quickly reference whether the preferences are adhered to in the schedule and make changes accordingly.

General preferences that apply to all semesters include time availability and preferred course length. The interface for specifying these preferences is shown in Figure 3. For time availability, instructors specify both the earliest and latest times they are able to teach. Additionally, they can specify the earliest and latest times that they would *prefer* to teach. For the preferred course length, instructors can indicate whether they prefer to teach courses that meet once per week or twice per week. There is also a comments field so that an

instructor can include any other preferences that they may have which were not explicitly covered in the form.

Semester-specific preferences for instructors include the courses they'd like to teach and the number of classes they are able to teach. The interface for specifying these preferences is shown in Figure 4. For preferred courses, the instructor constructs a list of all courses they typically teach and rank these from most to least preferred. For the number of classes, the instructor specifies the number of classes that they are expected to teach (by contract) and the number of additional classes that they would be willing to teach on top of their expected load. Since course offerings and teaching load can vary by semester, these preferences must be specified for each of the fall, spring, and summer terms.

### 3.4 Additional features

The class scheduling tool includes additional functionality that can help to assist in the scheduling process. Additional views include a table-based course summary view, a table of instructor assignments, and a table of instructor assignments that incorporates teaching preferences. Additional features include the ability to manage users and permissions, export data, view a log of changes, merge schedules, among others. Although this additional functionality is extremely useful for schedule administrators, it is beyond the scope of this thesis and will not be discussed in detail.

### 3.5 Implementation

The scheduling tool interface was built in Python using the Flask[2] web framework. It is backed by a MySQL database. The SQLAlchemy[3] object-relational mapper is used to

[2] `https://flask.palletsprojects.com/en/2.0.x/`
[3] `https://www.sqlalchemy.org/`

interface with the database from the Flask code. On the front-end, Bootstrap[4] is used for styling and jQuery/JavaScript are used for DOM manipulation, event handling, and AJAX requests. The FullCalendar[5] JavaScript library is used for the schedule visualizations. This library contained much of the required functionality for the visualisations already built in. The site is deployed on a server in the ADMT lab in the University of Pittsburgh Computer Science department. The code is available at `https://github.com/PittSmartLiving/class-scheduling`.

### 3.6 Summary

In this chapter, we discussed the user interface for our class scheduling tool. Several visualizations utilize a calendar-based view, which place classes on a grid based on their room and timeslot. These visualizations allow the user to quickly identify conflicts in the schedule. The user can easily make edits to the schedule from our interface. Additionally, instructors can log in to the tool and submit their teaching preferences. These preferences can be cross-referenced by the schedule administrator when building the schedule.

[4] `https://getbootstrap.com/`
[5] `https://fullcalendar.io/`

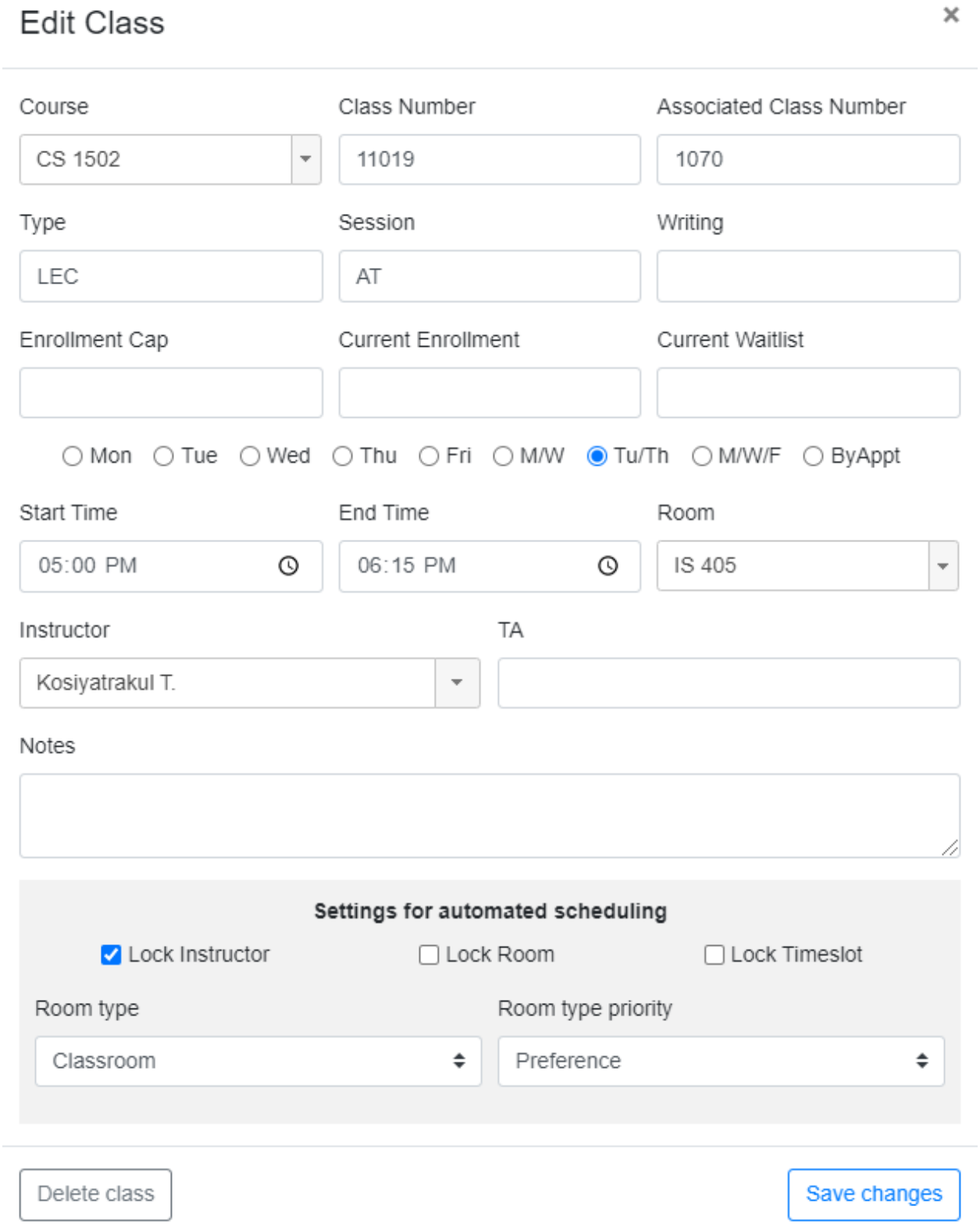


Figure 2: Edit modal

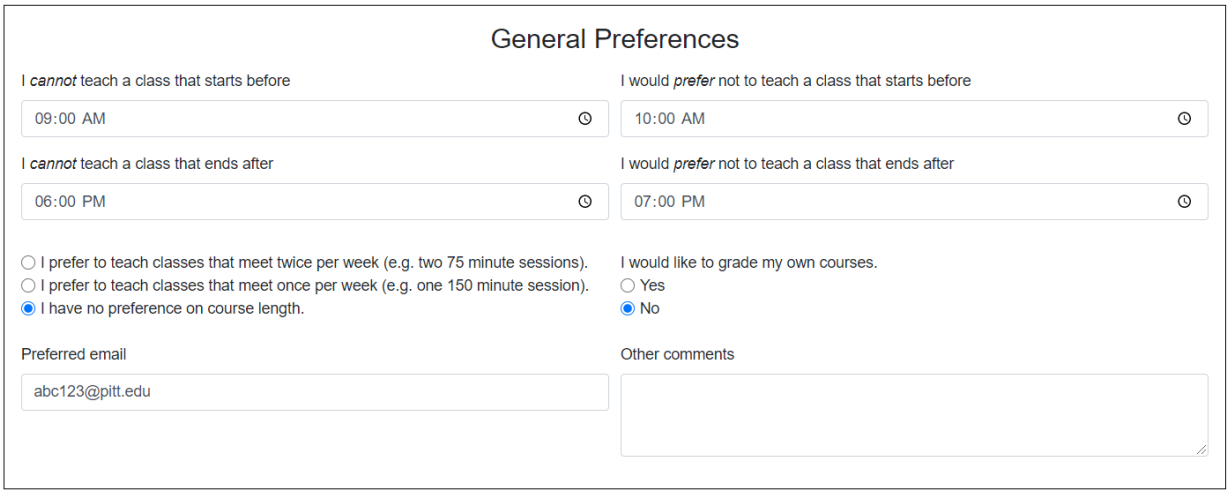


Figure 3: Interface for instructors to specify preferences. These preferences apply to all semesters.

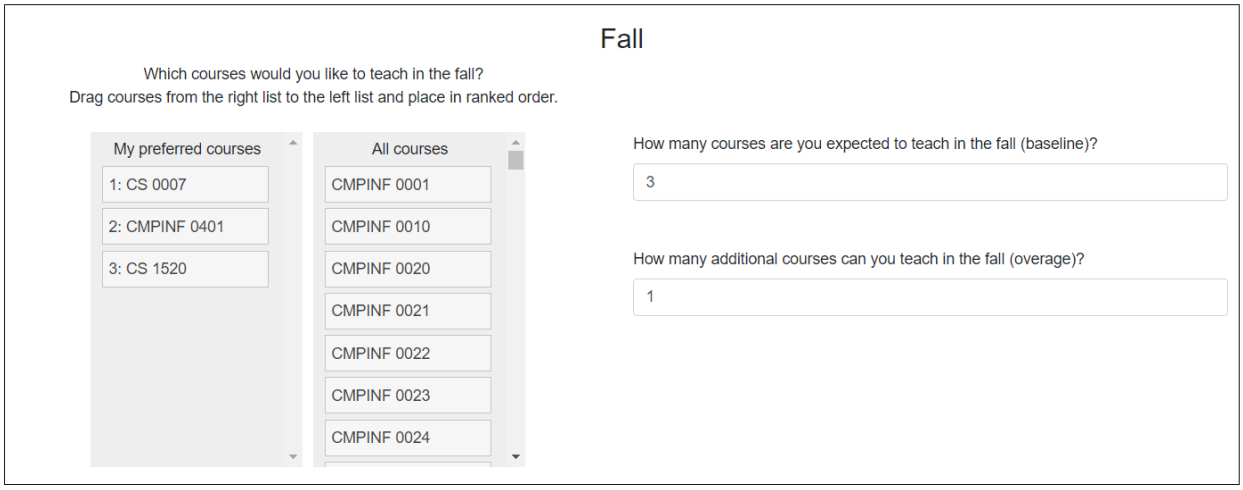


Figure 4: Interface for instructors to specify preferences. These preferences apply only to the fall semester. A similar interface is used to specify preferences for spring and summer semesters.

## 4.0 Automated scheduling feature

Rather than make manual changes to a schedule, the user can choose to use the automated scheduling feature instead. The automated scheduler will allow the system to make instructor, room, and timeslot assignments for each class such that the schedule is optimized to the metrics that are chosen by the user. This chapter will describe the interface and process for automated scheduling.

## 4.1 Optimization setup

The automated scheduler is implemented in Python and uses the Gurobi optimizer. Gurobi is a commercial optimizer that supports several different problem types, including linear programming, mixed integer linear programming, and quadradic programming. The class scheduling tool is supported under a Gurobi academic license. The Gurobi Python API is used to interface with the optimizer.

The different scheduling objectives and their linear programming formulations will be discussed in the next chapter. However, this section will cover the different approaches for working with multiple objectives. When objectives are competing, it becomes necessary to specify how to combine or prioritize these objectives during optimization. The automated scheduler supports two different approaches for combining objectives: blended or hierarchical. The blended approach requires the user to specify weights for each metric. This creates a new combined objective that is equivalent to a weighted sum of the individual metrics. The solver then optimizes this new combined objective. The hierarchical approach requires the user to specify a priority order for the different metrics. Then, the metrics are optimized one at a time in this order. Optimizing a lower priority metric does not necessarily find the overall best result for that metric. Rather, it finds the most optimal result such that the outcomes of the higher priority metrics do not degrade. The hierarchical and blended approaches can also be combined by creating a hierarchy of blended objectives.

When using the hierarchical approach, the user can introduce some flexibility by adding tolerance. Tolerance allows the result of a previously optimized metric to degrade to some degree in order to improve the result of a lower priority objective. The tolerance can be specified in absolute or relative (percentage) terms. However, it is important to note that the result of the higher priority metric will degrade if *any* improvement of the lower priority metric is possible, not just significant improvement.

Since there can be significant interaction between timeslot, instructor, and room assignments, it may be difficult for the solver to find an optimal solution for some of the metrics. The solver may spend several hours or even days trying find an optimal solution and prove its optimality. To avoid the scheduler getting stuck, we use a time limit for each metric. If the solver is unable to prove optimality before the time limit is reached, it will keep the best result that it was able to find within the time limit. In order for this to work well, the MIPFocus parameter of the solver must be configured correctly. The MIPFocus parameter defines the overall strategy for the solver. The default strategy contains a balance of finding new feasible solutions and proving whether the current solution is optimal. For the automated scheduler, we have set the MIPFocus to be 3, which means focus on trying to improve the bound of the objective, rather than proving optimality. In practice, this makes it so that the solver finds a good solution quicker.

Configuring the optimization parameters requires some intuition about each of the metrics. To set weights, the user must think about the expected magnitude of each of the metrics and how the metrics may interact. How much of a increase in metric A is equivalent to an increase in metric B? Additionally, the user must consider the complexity of the metric. Combining complex metrics together may make it difficult for the solver to find a good solution. When assigning priority levels, the user should think about the importance of the metric, but also how constraining it is. Optimizing a very specific metric first means there is little to no flexibility remaining when trying to optimize lower priority metrics. Adding tolerance can increase flexibility, however, it must be done with caution because the solver does not consider magnitudes of degradation/improvement (e.g. it may degrade a higher priority metric by 10% in order to make a 1% improvement in the lower priority metric). In Chapter 6, we will experimentally explore how to best configure these parameters.

### 4.2 Workflow and user interface

Our tool allows for an iterative scheduling workflow that includes both manual and automated scheduling phases. The suggested workflow is shown in Figure 5. The high-level idea is to allow the automated scheduler to find optimal schedules, while allowing the user to override certain assignments. Before using the automated scheduler, the user can "lock-in" certain assignments that they do not want the automated scheduler to change. After configuring the automated scheduler and allowing it to find an optimized schedule, the user can evaluate the new schedule using the visualization tools. They can then make manual changes, "lock-in" satisfactory assignments, and resubmit the schedule to the automated scheduler. This process is repeated until the user is satisfied with the schedule. The remainder of this section will go into more details for each step of the process.

First, instructors submit their preferences to the system. Next, the initial schedule is imported into the system. Typically, this will be the schedule from the same semester in the previous year. The user can make manual changes to the schedule, such as adding or removing class offerings or making changes to room, instructor, and timeslot assignments.

Once the user has imported the schedule, they must configure class-specific scheduling criteria. One such criteria is the room type. The room type can be either a classroom, computing lab, or no preference. This can be specified using the class edit modal shown in Figure 2. Additionally, the user must specify whether the room type should be considered as hard-constraint or a "nice-to-have" preference. Another criteria that must be configured is the enrollment cap for each class. This is also specified from the edit modal, as is necessary in order to find optimal-size rooms.

Next, the user can lock any assignments (instructors, rooms, or timeslots) that they definitely want to keep. Locking the assignment means that the automated schedule is not able to change it. The interface allows two different methods for locking assignments. The user can right click on a class which brings up a lock menu, as shown in Figure 6. Alternatively, the assignments can be locked from the edit modal as shown in Figure 2.

Once these class-specific preferences have been set, the user is ready to submit the schedule to the automated scheduler. To submit the schedule, the user must pick a name for the

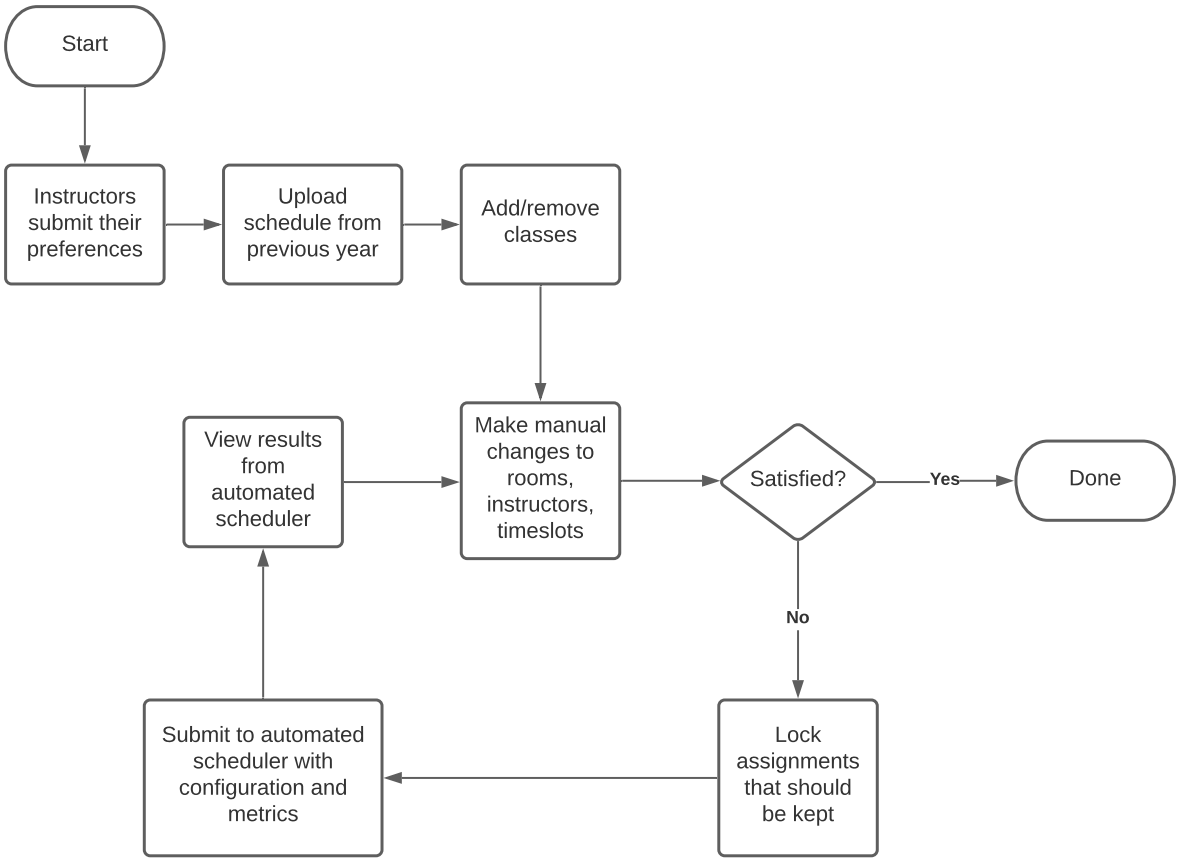


Figure 5: Workflow for using the scheduling tool. The workflow is an iterative process that alternates between manual and automated scheduling phases.

new schedule, specify which classes should be included in the new schedule (by subject), and how to handle the classes that are not included. This is done in the "Create Automated Schedule" modal that is shown in Figure 7. Filtering classes by subject is useful when the schedule is shared between multiple departments, but the user only wishes to change assignments for their specific department. Excluded classes can either be completely disregarded or can be considered for room availability. The later option is useful when rooms are shared between departments. For example, the CS department can use the automated scheduler to make changes to the CS classes only, but the scheduler will still take the INFSCI classes into

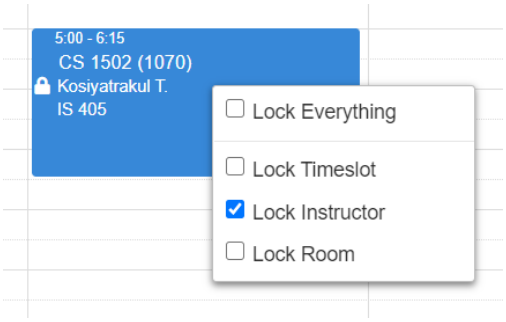


Figure 6: Right click on a class to open the lock menu. The user can lock the instructor, room, or timeslot to indicate that the automated scheduler should not change these assignments.

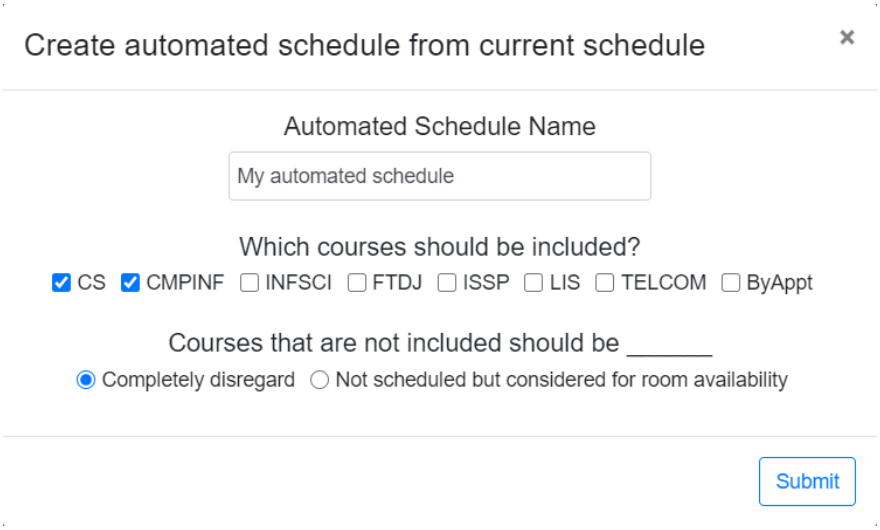


Figure 7: Modal for creating a new automated schedule.

consideration in order to avoid placing a CS class in a room that is already occupied by an INFSCI class.

The final step is to configure the automated scheduler. The user must select which instructors, timeslots, and rooms should be considered for scheduling from a list of all possible options that were added to the system. To allow for quick setup, instructors can be selected by department, rooms can be selected by building, and timeslots will default to the standard

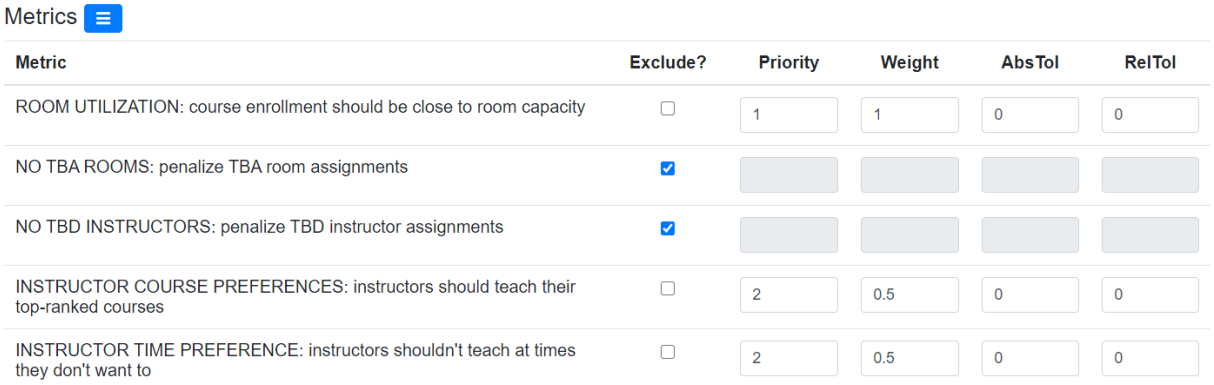


Figure 8: Interface for specifying metrics.

timeslots as specified by system admin. The user must also configure the metrics that should be included for scheduling. They select the priority, weight, and tolerance levels for each metric. Metrics can also be excluded completely. This is shown in Figure 8. Finally, the user configures additional settings, such as the optimization time limit, and the semester (needed to ensure the correct instructor preferences as used).

When the user submits the configuration for scheduling, some simple checks are done to ensure feasibility. This includes making sure there is at least one timeslot of the appropriate duration for each class, ensuring there is a room of an appropriate size for each class, and that each instructor has specified their preferences.

Once the solver has finished, the user can view the schedule using the visualization tools. They can then make manual changes to the schedule or "lock-in" any satisfactory assignments and resubmit to the automated scheduler. This process can be repeated multiple times until the user is satisfied with the schedule.

## 4.3 Implementation

The Flask server and database for the scheduling tool interface are hosted on server in the University of Pittsburgh Computer Science department. The automated scheduler is deployed on an AWS large-memory optimized EC2 instance with 128 GB of memory. The large-memory node is necessary in order to support the linear programming formulation for schedules that are the typical size for the University of Pittsburgh Computer Science department. Larger schedules will require even more memory. Information is relayed between our local server and EC2 instance through SCP of CSV files. This design choice was made because the EC2 instance is unable to access the database on our local server due to firewalls.

The scheduler uses the Gurobi Python API[1] to create the linear programming formulation. The process of creating the linear programming formulation is as follows. First, a model object is created. This object contains metadata such as the model name and objective sense (minimize or maximize). Next, variables are added to the model object. Variables are specified with a name and a type (binary, integer, etc). Variables can also be specified as a matrix that can be indexed into. Next, constraints and objectives are added to the model. Both constraints and objectives are written in terms of the model variables. Constraints are specified as a two-sided equation that must hold true. Objectives, on the other hand, are a single-sided equation which will be optimized based on the model sense. Objectives are also given metadata, including weights and priorities. Finally, the solver is configured with optimization parameters. The solve method is called on the model object, which submits the model to the Gurobi optimizer.

## 4.4 Summary

In this chapter, we discussed the automated scheduler feature that is part of our class scheduling tool. The automated scheduler uses integer linear programming to allow the system to suggest good assignments for classes. To use the automated scheduler, the user

[1] `https://www.gurobi.com/documentation/9.1/refman/py_python_api_overview.html`

must specify how to combine multiple metrics for optimization. The automated scheduler is not intended to be a one-step solution. Rather, it is intended to be used in conjunction with the manual scheduling and visualization interface.

# 5.0 Scheduling criteria and linear programming formulation

In this chapter, we discuss the linear programming formulation that is used by the automated scheduler. The linear programming formulation includes the variables, constraints, and metrics that define the scheduling problem.

## 5.1 Variables and notation

### 5.1.1 Decision variables

The primary variable that is used in our linear programming formulation is the class assignments matrix. This is a 4-dimensional binary matrix that stores the room, instructor, and timeslot assignments for each course. In the matrix, $x_{c,i,r,t} = 1$ if class $c$ is assigned to instructor $i$, room $r$, and timeslot $t$. Otherwise, $x_{c,i,r,t} = 0$. As will be seen in subsequent sections, constraints and objectives are created by summing over various sets of the decision variables.

### 5.1.2 Classes

$C = \{c_1, c_2, c_3, ...\}$ is the set of classes that should be scheduled. Each class $c$ is an instance of a course, such as CS 401. The attribute $c.course$ indicates the course for the class. The class type, $c.type$, defines the meeting type for the class, such as lecture, recitation, or lab. The associated class number, $c.assoc$, is an ID that connects different parts of the same course. For example, a recitation will have the same associated class number as its corresponding lecture. The duration, $c.duration$, and number of days, $c.days$, describe how long and how often the class should meet. The enrollment cap, $c.enroll$, is the maximum number of students that are able to enroll in the class. A class may request a specific room type which is specified by the $c.room\_type$ attribute. The $c.rm\_type\_priority$ attribute indicates whether this room type is a preference or a necessity (priority = 0 or 1 respectively).

In our tool, schedules are built from existing schedules (either from previous semesters or an earlier version of the same semester). As a result, each class has a previous instructor, room, and timeslot: $c.prev\_ins$, $c.prev\_room$, $c.prev\_ts$ respectively. These represent the assignments that were made for this class in the previous schedule. Additionally, the assignments may be "locked", indicating that the new assignment should be the same as the previous assignment. These are indicated with boolean attributes $c.ins\_lock$, $c.room\_lock$, and $c.ts\_lock$.

Classes may be cross-listed, meaning they exist as multiple courses in the schedule, however, meet and act as one class in practice. In the following sections of this chapter, cross-listed courses are ignored in order to simplify the notation of the constraints and metrics. However, in the implementation, these are handled by grouping all classes with the same crosslist ID into one class that has multiple course attributes.

### 5.1.3 Rooms, instructors, and timeslots

$R = \{r_1, r_2, r_3, ...\}$ is the set of rooms that are available for scheduling. Each room $r$ has a maximum occupancy ($r.occupancy$) and a classroom type ($r.type$). The classroom type is either a regular classroom or computing lab. There is one special room, $r_{TBA}$, which indicates that the room needs to be assigned by the registrar. Because $r_{TBA}$ is not a real room, it is not subject to the some of the same constraints as the other rooms (such as occupancy and room conflict constraints).

$T = \{t_1, t_2, t_3, ...\}$ is the set of timeslots that are available for scheduling. These timeslots are determined by the registrar or the department. Each timeslot $t$ consists of start time ($t.start$), end time ($t.end$), and meeting pattern ($t.days$). The meeting pattern may consist of one day or multiple days.

$I = \{i_1, i_2, i_3, ...\}$ is the set of instructors. Each instructor has a maximum number of courses that they are able to teach during a semester ($i.class\_count$). This value is equal to the instructors expected class count (baseline) plus the number of additional classes that they are willing to teach (overage). Each instructor also has a ranked list of courses that they would prefer to teach. The instructor can specify their time availability which should

not be violated as well as their preferred teaching hours which are not guaranteed to be adhered to. There is one special instructor, $i_{TBD}$, which is a placeholder that indicates that the instructor will be determined at a later date. Because $i_{TBD}$ is not a real instructor, it is not subject to some of the same constraints as the other instructors.

### 5.1.4 Additional sets and functions

Functions can be useful for defining instructors, rooms, or timeslots that meet certain criteria. The following functions will be used in the linear programming formulation.

- $pairs(list)$ takes in a list of elements. It returns the set of all possible combinations of size 2 from those elements.
- $ts\_overlap(t_1, t_2)$ takes in two timeslots $t_1$ and $t_2$. The function returns 1 if there is a conflict between the timeslots (i.e. they occur at least partially at the same time). Otherwise, it returns 0.
- $correct\_rm\_type(c, r)$ takes in a class $c$ and room $r$. The function returns 1 if $r$ is the correct room type for $c$. Otherwise, it returns 0.
- $correct\_duration(c, t)$ takes in a class $c$ and timeslot $t$. The function returns 1 if the timeslot has the correct duration and number of days that is required for the class. Otherwise, it returns 0.
- $ts\_align(t_1, t_2, a)$ takes in two timeslots $t_1$ and $t_2$ and an alignment $a$. The function returns 1 if $t_1$ has the given alignment relative to $t_2$. Otherwise, it returns 0.
- $prev\_room(c, r)$ takes in a class $c$ and room $r$. It returns 1 if $r$ was the assigned room for $c$ in the previous version of the schedule. Otherwise, it returns 0.
- $prev\_instructor(c, i)$ takes in a class $c$ and instructor $i$. It returns 1 if $i$ was the assigned instructor for $c$ in the previous version of the schedule. Otherwise, it returns 0.
- $prev\_timeslot(c, t)$ takes in a class $c$ and timeslot $t$. It returns 1 if $t$ was the assigned timeslot for $c$ in the previous version of the schedule. Otherwise, it returns 0.
- $pref\_course\_len(i, t)$ takes in an instructor $i$ and timeslot $t$. It returns 1 if the timeslot is the preferred length (1 or 2 days per week) for instructor $i$. It also returns 1 if the instructor has no preference. Otherwise, it returns 0.

- $overlap\_penalty(c_1, c_2)$ takes in two classes $c_1$ and $c_2$. The function returns the penalty for the two classes occurring at same time. The penalties are specified in a configuration file for the tool.
- $time\_avail(i, t)$ takes in an instructor $i$ and timeslot $t$. It returns 1 if the instructor is available during the given timeslot. Otherwise, it returns 0.
- $time\_pref(i, t)$ takes in an instructor $i$ and timeslot $t$. It returns 1 if the timeslot is within the instructor's preferred working hours. Otherwise, it returns 0.
- $rank(i, c)$ takes in an instructor $i$ and class $c$. It returns the rank that the instructor gave this course in their course preference list. Lower ranks are more preferred. If the course is not in the instructors list, then the function returns NULL.
- $ts\_day\_group(t, g)$ takes in a timeslot $t$ and a group number $g$. It returns 1 if the timeslot belongs to the given day-group. Otherwise, it returns 0. Group 1 is a timeslot that occurs on a Monday, Wednesday or Friday. Group 2 is a timeslot that occurs on Tuesday or Thursday.
- $ts\_time\_group(t, g)$ takes in a timeslot $t$ and a group number $g$. It returns 1 if the timeslot belongs to the give time-group. Otherwise, it returns 0. Group 1 is a timeslot that starts before 12pm. Group 2 is a timeslot that starts between 12pm and 3pm. Group 3 is a timeslot that starts between 3pm and 6pm. Group 4 is a timeslot that starts after 6pm.

The following sets are also used in the linear programming formulation. These sets help to simplify notation of the constraints and metrics.

- Set of conflicting timeslot pairs
  $T_c = \{(t_1, t_2) : t_1 \in T \text{ and } t_2 \in T \text{ and } ts\_overlaps(t_1, t_2) = 1\}$
- Set of lectures
  $C_{lec} = \{c : c \in C \text{ and } c.type \neq REC \text{ and } c.type \neq LAB\}$
- Set of recitations
  $C_{rec} = \{c : c \in C \text{ and } c.type = REC \text{ or } c.type = LAB\}$
- Set of graduate lecture pairs
  $C_g = \{(c_1, c_2) : (c_1, c_2) \in pairs(C_{lec}) \text{ and both } c_1.course \text{ and } c_2.course \text{ are graduate level}\}$

- Set of undergraduate lecture pairs

  $C_u = \{(c_1, c_2) : (c_1, c_2) \in pairs(C_{lec})$ and both $c_1.course$ and $c_2.course$ are undergrad level$\}$
- Set of same-course lecture pairs

  $C_c = \{(c_1, c_2) : (c_1, c_2) \in pairs(C_{lec})$ and $c_1.course = c_2.course\}$
- Set of associated class pairs

  $C_a = \{(c_1, c_2) : (c_1, c_2) \in pairs(C)$ and $c1.course = c2.course$ and $c1.assoc = c2.assoc\}$
- Set of recitation alignments

  $A = \{-1, 0, 1\}$ representing that a recitation occurs before, between, or after the lecture sessions respectively.
- Set of blocked times

  $B = \{(r, t) : r \in R$ and $t \in T$ and $r$ is occupied during $t\}$.
- Set of locked-in assignments

  $L_i = \{(c, i) : c \in C$ and $i \in I$ and $c.prev\_ins = i$ and $c.ins\_lock\}$

  $L_r = \{(c, r) : c \in C$ and $r \in R$ and $c.prev\_room = r$ and $c.room\_lock\}$

  $L_t = \{(c, t) : c \in C$ and $t \in T$ and $c.prev\_ts = t$ and $c.ts\_lock\}$
- Set of lectures and their recitations

  $S = \{(lec, \{rec : rec \in C_{rec}$ and $rec$ is associated with $lec\}) : lec \in C_{lec}\}$

### 5.2 Constraints

In our work, we consider the following constraints.

- **Class Assignments**: each class must be assigned to exactly one instructor, room, and timeslot.

$$\forall c \in C \quad \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} = 1$$

- **No Room Conflicts**: a room cannot hold two classes at the same time (except the TBA room).

$$\forall r \in R \quad \forall (t_1, t_2) \in T_c \quad \sum_{c \in C} \sum_{i \in I} \sum_{t \in (t_1, t_2)} x_{c,i,r,t} \leq 1$$

- **No Instructor Conflicts**: an instructor cannot teach two classes at the same time (except the TBD instructor).

$$\forall i \in I \quad \forall (t_1, t_2) \in T_c \ \sum_{c \in C} \sum_{r \in R} \sum_{t \in (t_1, t_2)} x_{c,i,r,t} \leq 1$$

- **Room Occupancy**: the class enrollment capacity must not exceed the room occupancy.

$$\forall c \in C \quad \forall r \in R \quad \text{if } c.enroll > r.occupancy \text{ then } \sum_{i \in I} \sum_{t \in T} x_{c,i,r,t} = 0$$

- **Course Duration**: each class must be assigned a timeslot of the appropriate duration.

$$\forall c \in C \quad \forall t \in T \quad \text{if } correct\_duration(c, t) = 0 \text{ then } \sum_{i \in I} \sum_{r \in R} x_{c,i,r,t} = 0$$

- **No Associated Class Conflicts**: associated classes (e.g. a lecture and its recitation) cannot conflict in time.

$$\forall (c_1, c_2) \in C_a \quad \forall (t_1, t_2) \in T_c \ \sum_{c \in (c_1, c_2)} \sum_{i \in I} \sum_{r \in R} \sum_{t \in (t_1, t_2)} x_{c,i,r,t} \leq 1$$

- **Recitation Instructors**: recitations and labs do not have an instructor and should be assigned TBD. They will be assigned a TA at a later date.

$$\forall c \in C_{rec} \ \sum_{r \in R} \sum_{t \in T} x_{c,i_{TBD},r,t} = 1$$

- **Instructor Class Count**: each instructor has a maximum number of classes that they are able to teach.

$$\forall i \in I \ \sum_{c \in C} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} \leq i.class\_count$$

- **Instructor Time Constraints**: instructors are unable to teach at certain times.

$$\forall i \in I \quad \forall t \in T \quad \text{if } time\_avail(i, t) = 0 \text{ then } \sum_{c \in C} \sum_{r \in R} x_{c,i,r,t} = 0$$

- **Room Type Constraints**: certain classes can only be held in certain room types (e.g. computing lab).

$$\forall c \in C \quad \forall r \in R \quad \text{if } \mathit{correct\_rm\_type}(c, r) = 0 \text{ and } c.\mathit{rm\_type\_priority} = 1 \text{ then}$$

$$\sum_{i \in I} \sum_{t \in T} x_{c,i,r,t} = 0$$

- **Blocked Times**: some rooms are unavailable at certain times (e.g. they are occupied by a different department).

$$\forall (r, t) \in B \quad \sum_{c \in C} \sum_{i \in I} x_{c,i,r,t} = 0$$

- **Locked In Classes**: some classes are pre-assigned an instructor, room, and/or timeslot which should not be changed by the scheduler.

$$\forall (c, i) \in L_i \quad \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} = 1$$

$$\forall (c, r) \in L_r \quad \sum_{i \in I} \sum_{t \in T} x_{c,i,r,t} = 1$$

$$\forall (c, t) \in L_t \quad \sum_{i \in I} \sum_{r \in R} x_{c,i,r,t} = 1$$

- **Instructor Courses**: an instructor can only teach courses that are included on their preference list.

$$\forall i \in I \quad \forall c \in C \quad \text{if } \mathit{rank}(i, c) = \mathit{NULL} \text{ then } \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} = 0$$

### 5.3 Metrics

Metrics measure the quality of the schedule. For all metrics, lower values are better and zero is the most-optimal value. In this work, we consider the following metrics.

- **No TBA Rooms (TBA_RM)**: classes should be assigned to an actual classroom, not the TBA room. This metric counts the number of classes that are assigned to a TBA room.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{t \in T} x_{c,i,r_{TBA},t}$$

- **No TBD Instructors (TBD_INS)**: classes should be assigned an actual instructor, not the TBD instructor. This metric counts the number of classes that are assigned the TBD instructor.

$$\text{minimize} \sum_{c \in C} \sum_{r \in R} \sum_{t \in T} x_{c,i_{TBD},r,t}$$

- **Room Non-Utilization (RM_UTIL)**: class enrollment should be close to the room capacity. This metric calculates the percentage of the room occupancy that is not being used for each class and sums these values over all classes.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * (1 - c.enroll/r.occupancy)$$

- **Instructor Course Preferences (INS_COURSES)**: instructors should teach their top-ranked courses. This metric sums the course rankings that each instructor gave to the courses that they get assigned to teach. Lower rankings are more preferred.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * rank(i,c)$$

- **Instructor Time Preferences (INS_TIMES)**: instructors should teach during their preferred times. This metric counts the number of occurrences of an instructor teaching outside of their preferred times.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * (1 - time\_pref(i,t))$$

- **Room Type Preference (RM_TYPE)**: certain classes should be held in a specific room type (e.g. computing lab) if possible. This metric counts the number of classes that are assigned to a room that is not the preferred type.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * (1 - pref\_rm\_type(c, r)) * (1 - c.rm\_type\_priority)$$

- **Recitation Alignment by Section (REC_SECTION)**: all recitations that are associated with the same lecture should have the same relative placement (before/after/between the meetings of the lecture). This metric counts the number of pairs of recitations from the same section that do not have the same alignment.

  The LP formulation is defined in multiple steps. First, we create binary variables, *rec_align*, to indicate the alignment of each recitation. A constraint is used for creating these variables.

$$\forall (c_{lec}, recs) \in S \quad \forall c_{rec} \in recs \quad \forall a \in A \quad \forall t_{lec} \in T$$

$$rec\_align[c_{rec}, a, t_{lec}] + 1 \geq \sum_{i \in I} \sum_{r \in R} x_{c_{lec},i,r,t_{lec}} + \sum_{t_{rec} \in T} \sum_{i \in I} \sum_{r \in R} x_{c_{rec},i,r,t_{rec}} * ts\_align(t_{rec}, t_{lec}, a)$$

  The right-hand side of this constraint is testing for two conditions. The first condition is that the lecture $c_{lec}$ is held during timeslot $t_{lec}$. If this is true, the first summation will be equal to 1, otherwise 0. The second condition is that recitation $c_{rec}$ is held during a timeslot that has the alignment $a$ compared to $t_{lec}$. If this is true, the second summation will be equal to 1, otherwise 0. We want the *rec_align* variable to be 1 if and only if both of these conditions are true (i.e. the right hand side is equal to 2). We use an objective and minimize the sum of these *rec_align* variables in order to accomplish this.

$$\text{minimize} \sum_{c \in C_{rec}} \sum_{a \in A} \sum_{t \in T} rec\_align[c, a, t]$$

  Note that the result of this minimization will always be equal to the number of recitations, since each recitation must have exactly 1 alignment and its lecture can only occur at exactly 1 time. Although we are using an objective, the assignments that the scheduler makes cannot actually improve (or deteriorate) the result. Therefore, at this point, we

have not actually imposed any new constraints or done any optimization on the schedule. Rather, we are using constraints and objectives to create indicator variables.

Next, we create binary variables that will indicate whether two recitations have the same alignment. We want the binary $same\_align[c_1, c_2, a]$ variable to be 1 if and only if both of the recitations $c_1$ and $c_2$ have the alignment $a$.

$$\forall (c_{lec}, recs) \in S \quad \forall (c_1, c_2) \in pairs(recs) \quad \forall a \in A$$

$$2 * same\_align[c_1, c_2, a] \leq \sum_{t_{lec} \in T} rec\_align[c_1, a, t_{lec}] + rec\_align[c_2, a, t_{lec}]$$

$$\text{minimize} \sum_{(c_{lec}, recs) \in S} |pairs(recs)| - \sum_{(c_{lec}, recs) \in S} \sum_{(c_1, c_2) \in pairs(recs)} \sum_{a \in A} same\_align[c_1, c_2, a]$$

Again, we use an objective to enforce this constraint. We minimize the difference between the total number of recitation pairs (the first summation) and the number pairs that have the same alignment (the second summation). This is the same as maximizing the number of pairs with the same alignment or, in other words, maximizing the sum of the *same_align* variables. This optimization serves two purposes. First, it ensures that a *same_align* variable equals 1 when both recitations in the pair have the given alignment (i.e. the right-hand side of the constraint equals 2). Second, it encourages the scheduler to make assignments such that the recitation pairs do indeed have the same alignment. Unlike the objective from the previous step, the result of this objective is not constant. Therefore, the result is actually affected by the assignments that are made by the scheduler.

- **Recitation Alignment by Course (REC_COURSE)**: recitations for the same course (including different sections) should have the same recitation placement relative to its corresponding lecture. This metric counts the number of recitation pairs that do not have the same alignment.

  The LP formulation is almost identical to the Recitation Alignment Per Section metric that is shown above. The only difference is that the *same_align* variables are created for all recitation pairs of a given course rather than a section.

- **Recitations End of Week (REC_END)**: recitations should be at the end of the week (after the last lecture). This metric counts the number of recitations that occur before the first lecture or between the lectures. The formulation is similar to the Recitation Alignment by Section metric that is shown above. First, we create *rec_align* variables to indicate the alignments.

$$\forall(c_{lec}, recs) \in S \quad \forall c_{rec} \in recs \quad \forall a \in A \quad \forall t_{lec} \in T$$

$$rec\_align[c_{rec}, a, t_{lec}]+1 \geq \sum_{i \in I} \sum_{r \in R} x_{c_{lec},i,r,t_{lec}} + \sum_{t_{rec} \in T} \sum_{i \in I} \sum_{r \in R} x_{c_{rec},i,r,t_{rec}} * ts\_align(t_{rec}, t_{lec}, a)$$

Next, we minimize the sum of these variables. Unlike the REC_SECTION metric, we only include the "before" and "between" alignments (-1 and 0) in the minimization. The minimization serves two purposes. First, it enforces that a $rec\_align[c, a, t_{lec}]$ variable equals 1 if and only if the lecture is held at time $t_{lec}$ and the recitation $c$ has the alignment $a$. Second, it encourages the scheduler to make assignments such that recitations have the "after" alignment. Because we do not include the "after" alignment in the summation, the sum is not constant. Thus, it is dependent on the assignments that are made by the scheduler.

$$\text{minimize} \sum_{c \in C_{rec}} \sum_{a \in [0,1]} \sum_{t \in T} rec\_align[c, a, t]$$

- **Graduate Options (GRAD_OPT)**: allow scheduling options for graduate students by offering lectures at non-conflicting times. This metric sums the penalties for each conflict between graduate classes that occurs. Each penalty is dependent on the courses that are conflicting. Criteria for the penalties include number of sections offered, prerequisites, and core/elective course type. The penalties are specified in a configuration file for the system.

  In the first step, we create binary *overlap* variables, which will indicate whether two classes occupy two conflicting timeslots. Next, we minimize the sum of the *overlap* variables. The minimization serves two purposes. First, it enforces that an $overlap[c_1, c_2, t_1, t_2]$ variable equals 1 if and only if the the classes, $c_1$ and $c_2$, are offered at times $t_1$ and $t_2$ respectively (i.e. the right hand side of the constraint is equal to 2). Second, it encourages

the scheduler to make assignments such that these important courses are not offered at the same time.

$$\forall(c_1, c_2) \in C_g \quad \forall(t_1, t_2) \in T_c \quad overlap[c_1, c_2, t_1, t_2] + 1 \geq \sum_{i \in I} \sum_{r \in R} x_{c_1,i,r,t_1} + \sum_{i \in I} \sum_{r \in R} x_{c_2,i,r,t_2}$$

$$\text{minimize} \sum_{(c_1,c_2) \in C_g} \sum_{(t_1,t_2) \in T_c} overlap[c_1, c_2, t_1, t_2] * overlap\_penalty(c_1, c_2)$$

- **Undergraduate Options (UNGRD_OPT)**: allow scheduling options for undergraduate students by offering lectures at non-conflicting times. This metric sums the penalty for each conflict between undergraduate classes that occurs. Each penalty is dependent on the courses that are conflicting. Criteria for the penalties include number of sections offered, prerequisites, and core/elective course type. The formulation is the same as the above metric for graduate students, but includes different courses.

$$\forall(c_1, c_2) \in C_u \quad \forall(t_1, t_2) \in T_c \quad overlap[c_1, c_2, t_1, t_2] + 1 \geq \sum_{i \in I} \sum_{r \in R} x_{c_1,i,r,t_1} + \sum_{i \in I} \sum_{r \in R} x_{c_2,i,r,t_2}$$

$$\text{minimize} \sum_{(c_1,c_2) \in C_u} \sum_{(t_1,t_2) \in T_c} overlap[c1, c2, t1, t2] * overlap\_penalty(c_1, c_2)$$

- **Recitation Options (REC_OPT)**: allow scheduling options for students by offering recitations at non-conflicting times. This metric sums the penalty for each conflict that occurs. Each penalty is dependent on the courses that are conflicting. Criteria for the penalties include number of sections offered, prerequisites, and core/elective course type. The formulation is the same as the above metric for graduate lectures, but considers conflicts that occur with recitations (either recitation/recitation or recitation/lecture conflict).

$$\forall c_1 \in C_{rec} \quad \forall c_2 \in C \quad \forall(t_1, t_2) \in T_c \quad overlap[c_1, c_2, t_1, t_2] + 1 \geq \sum_{i \in I} \sum_{r \in R} x_{c_1,i,r,t_1} + \sum_{i \in I} \sum_{r \in R} x_{c_2,i,r,t_2}$$

$$\text{minimize} \sum_{c_1 \in C_{rec}} \sum_{c_2 \in C} \sum_{(t_1,t_2) \in T_c} overlap[c_1, c_2, t_1, t_2] * overlap\_penalty(c_1, c_2)$$

- **Instructor Course Length (COURSE_LEN)**: instructors should teach courses that are their preferred length (1 day a week vs 2 days a week). This metric counts the number of times an instructor is assigned to teach a courses that is not their preferred length.

$$\text{minimize} \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * (1 - pref\_course\_len(i,t))$$

- **Multiple Sections Different Times (SEC_DIFF_TM)**: courses with multiple sections should be offered at different times. This metric counts the number of occurrences of the same course being offered in the same timeslot. First, we create binary *same_time* variables that will indicate whether two classes occur at the same timeslot.

$$\forall (c_1, c_2) \in C_c \quad \forall t \in T \quad same\_time[c_1, c_2, t] + 1 \geq \sum_{c \in (c_1, c_2)} \sum_{i \in I} \sum_{r \in R} x_{c,i,r,t}$$

Next, we minimize the sum of the *same_time* variables. The minimization serves two purposes. First, it enforces that a $same\_time[c_1, c_2, t]$ variable equals 1 if and only if the the classes, $c_1$ and $c_2$, are both at time $t$ (i.e. the right hand side of the constraint is equal to 2). Second, it encourages the scheduler to make assignments such that these courses are not offered at the same time.

$$\text{minimize} \sum_{(c_1, c_2) \in C_c} \sum_{t \in T} same\_time[c_1, c_2, t]$$

- **Load Balance TBA Rooms (BAL_TBA)**: balance the TBA room assignments across multiple timeslots. This helps to increase the likelihood that the registrar will be able to find a room for the class. If we request too many rooms at the same time, then it will be more difficult for the registrar to satisfy the request. This metric is equal to the maximum number of courses that are assigned to the TBA room during a single timeslot. First, we create integer variables that count the number of TBA rooms for each timeslot.

$$\forall t \in T \quad tba\_count[t] = \sum_{c \in C} \sum_{i \in I} x_{c,i,r_{TBA},t}$$

Next, we create an integer variable that is equal to the maximum of these counts.

$$\forall t \in T \quad maxTBACount \geq tba\_count[t]$$

Finally, we minimize this maximum count to encourage the scheduler to spread to TBA rooms across timeslots.

$$\text{minimize } maxTBACount$$

- **Shuffle Results (SHUF)**: instructor, room, and timeslot assignments should be different than the previous schedule. This metric is useful when doing multiple iterations of automatic scheduling. For example, a user can create a schedule using the automated scheduler then lock-in the assignments that they are satisfied with. They can then re-run the automated scheduler to find new assignments for anything that was not locked-in. If the same metric configuration is used the second time the scheduler is run, then the scheduler will return the same (unsatisfactory) assignments. The shuffle metric will encourage the scheduler to find new assignments. The metric is equal to the number of assignments that are the same as in the previous schedule.

$$\begin{aligned}\text{minimize } & \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * prev\_room(c,r) \\ & + \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * prev\_instructor(c,i) \\ & + \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * prev\_timeslot(c,t)\end{aligned}$$

- **Preserve Results (PRES)**: instructor, room, and timeslot assignments should be the same as the previous schedule. This metric is useful when fine-tuning an existing schedule. It encourages the scheduler to keep the current assignments unless making an alternative assignment would result in significant improvement in one of the other metrics. The metric is equal to the number of assignments that are different from the previous schedule.

$$\begin{aligned}\text{minimize } & \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * (1 - prev\_room(c,r)) \\ & + \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * (1 - prev\_instructor(c,i)) \\ & + \sum_{c\in C}\sum_{i\in I}\sum_{r\in R}\sum_{t\in T} x_{c,i,r,t} * (1 - prev\_timeslot(c,t))\end{aligned}$$

- **Spread Out Sections (SPREAD)**: Different sections of the same course should be offered on different days and at generally different times throughout the week. This allows students to have multiple options to fit into their schedules. This metric counts the number of pairs of lectures that occur on similar days or times. The timeslots are split into two groups based on days - those that include Monday, Wednesday, or Friday and those that include Tuesday or Thursday. We create binary variables $day\_group[c_1, c_2, g]$ that indicate whether a pair of classes occur in the same group of days. The same is done for times. Timeslots are split into four groups based on their start times - morning, midday, afternoon, and evening. The binary *time_group* variables indicate whether a pair of classes occur in the same group of times.

  $$\forall (c_1, c_2) \in C_c \quad \forall g \in [1, 2] \quad day\_group[c_1, c_2, g] + 1 \geq \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * ts\_day\_group(t, g)$$

  $$\forall (c_1, c_2) \in C_c \quad \forall g \in [1, 2, 3, 4] \quad time\_group[c_1, c_2, g] + 1 \geq \sum_{c \in C} \sum_{i \in I} \sum_{r \in R} \sum_{t \in T} x_{c,i,r,t} * ts\_time\_group(t, g)$$

  We minimize the sum of the *day_group* and *time_group* variables. This enforces that each *day_group* or *time_group* variable has the value 1 if and only if both classes occur within the given timeslot group (i.e. the sum of the right hand side of the constraint is equal to 2). Additionally, it encourages the solver to find a solution such that sections occur in different day and time groups.

  $$\text{minimize} \sum_{(c_1,c_2) \in C_c} \sum_{g \in [1,2]} day\_group[c_1, c_2, g] + \sum_{(c_1,c_2) \in C_c} \sum_{g \in [1,2,3,4]} time\_group[c_1, c_2, g]$$

## 5.4 Summary

In this chapter, we discussed the constraints and metrics that are built into our automated scheduler tool. These constraints and metrics reflect the scheduling needs of the University of Pittsburgh Computer Science department. For each constraint and metric, we explained how it is formulated in terms of integer linear programming.

# 6.0 Experimental evaluation

In this chapter, we will evaluate the performance of the automated scheduler. Through a series of four experiments, different metric configurations will be compared in order to elucidate best practices when working with automated scheduler.

## 6.1 Experimental setup

The following experiments are based on the Fall 2021 schedule for the University of Pittsburgh Computer Science department. The schedule contains 191 classes, 57 instructors (plus a TBD instructor), 7 rooms (plus a TBA room), and 140 timeslots. The instructors in the department submitted their teaching preferences through the class scheduling tool.

For all experiments, the solver was given a time limit of 10 minutes per metric. In some cases, it is possible that the solver will not be able to find an optimal solution within the time limit. When this occurs, the solver will keep the best solution that it was able to find, even though that solution was not yet proven to be optimal.

## 6.2 Experiment 1: Combining metrics

In this experiment, we explore the different approaches for combining multiple metrics. For simplicity, we only consider only two metrics - Room Non-Utilization (RM_UTIL) and Recitations at the End of the Week (REC_END). Intuitively, these two metrics are not completely independent. Forcing recitations to be at the end of the week limits room availability at the end of the week. Thus, in order for these classes to occur at the end of the week, they may need to be assigned to classrooms that are a sub-optimal fit for their enrollment capacities. The results of the different approaches for combining these two metrics are shown in Table 1.

Schedule A considers RM_UTIL as the only metric. It does not take REC_END into consideration at all. Schedule B is the opposite. It only considers REC_END and does not consider RM_UTIL. The results from these schedules show us the optimal value for each of the respective metrics.

Schedules C and D utilize the hierarchical approach. Schedule C has RM_UTIL as higher priority than REC_END. Comparing this schedule with schedule A, we see that the result for the RM_UTIL metric is still the optimal solution. However, there is some improvement for the REC_END metric compared to schedule A. Schedule D has REC_END as higher priority that RM_UTIL. Again, the result for the higher priority metric is the optimal solution, while there is some improvement on the lower priority metric compared to schedule B.

Schedule E uses the blended approach. In this schedule, the metrics are blended with equal weights. The result is that neither metric reaches its optimal solution. However, the results are closer to schedule D than schedule C. This makes sense because for a single class, there is a larger possible improvement in REC_END than RM_UTIL. Each change in REC_END will change the combined metric by 1.0, however, the worst possible change in the combined metric due to RM_UTIL is less than 1.0. For example, consider we are trying to decide whether to hold a recitation early in the week in a room that has 1% non-utilization, versus holding the recitation at the end of the week in a room that has 99% non-utilization. In the first case, the combined penalty is 1.01 (1 for REC_END + 0.01 for RM_UTIL). In the second case, the penalty is 0.99 (0 for REC_END + 0.99 for RM_UTIL). Thus, it is always better to improve REC_END even at the expense of RM_UTIL. The reason that the REC_END does not reach its optimal value (as it did in schedule D) is because combining the metrics creates a more difficult and complex metric to solve. The optimization process was not complete when the time limit was reached.

Schedules F and G also use the blended approach, however, they give a larger weight to RM_UTIL so that it is not completely dominated by REC_END. In schedule F, RM_UTIL has a weight that is double that of REC_END. The result is that both metrics reach a value that is somewhere between the values from the hierarchical schedules C and D. In schedule G, RM_UTIL has a weight that is 10 times larger than REC_END. This causes RM_UTIL to dominate the combined metric, thus the result is very similar to schedule C. Again, the

| | Room Non-Utilization (RM_UTIL) | | | | | Recitations End of Week (REC_END) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Sched | Priority | Weight | AbsTol | RelTol | Result | Priority | Weight | AbsTol | RelTol | Result |
| A | 1 | 1 | 0 | 0 | 50.03 | excluded | | | | 55 |
| B | excluded | | | | 165.43 | 1 | 1 | 0 | 0 | 0 |
| C | 2 | 1 | 0 | 0 | 50.03 | 1 | 1 | 0 | 0 | 38* |
| D | 1 | 1 | 0 | 0 | 63.99 | 2 | 1 | 0 | 0 | 0 |
| E | 1 | 1 | 0 | 0 | 63.72* | 1 | 1 | 0 | 0 | 3* |
| F | 1 | 2 | 0 | 0 | 54.52* | 1 | 1 | 0 | 0 | 18* |
| G | 1 | 10 | 0 | 0 | 50.27* | 1 | 1 | 0 | 0 | 33* |
| H | 1 | 1 | 0 | 0 | 60.26* | 2 | 1 | 8 | 0 | 8 |
| I | 2 | 1 | 0 | 0.1 | 54.97 | 1 | 1 | 0 | 0 | 14* |

Table 1: Results from combining RM_UTIL and REC_END metrics using different approaches. A star (*) indicates time limit was reached before an optimal solution could be found or proved.

combined metric is more difficult to optimize, thus the optimization process did not fully complete for either of these schedules.

Schedules H and I use the hierarchical approach with tolerance. When REC_END is optimized before RM_UTIL (schedule D), the result for the REC_END metric is 0. For schedule H, we use the same hierarchical order, however, we allow a tolerance of 8 for REC_END. By allowing the REC_END result to degrade by 8, the solver is able to find a better result for RM_UTIL. When RM_UTIL is optimized before REC_END (schedule C), the result for the RM_UTIL metric is 50.03. By adding 0.1 tolerance, we allow this value to degrade up to 10% (up to 55.03) in order to improve the REC_END result. The tolerance approach does not consider magnitudes of improvement between the two metrics. Rather, the higher priority metric will always degrade if doing so allows for any improvement in the lower priority metric.

### 6.3 Experiment 2: Identifying conflicting metrics

The goal of this experiment is to identify metrics that may conflict with each other. Metrics are optimized in a pairwise fashion. For each pair of metrics, A and B, the hierarchical optimizations of "A then B" versus "B then A" are compared. For non-conflicting metrics, the order of optimization will not matter thus the resulting outcomes will be similar for the two hierarchical orders. However, for metrics that do conflict, the two hierarchies will produce different outcomes. Due to the large number of combinations possible, this experiment will not consider all possible pairs of metrics, but rather a sampling of pairs which may be interesting. Additionally, it is important to note that this experiment is only considering two-way interactions. It is possible that three-way (or larger) interactions also exist.

The results of this experiment are shown in Table 2. Clearly, the SHUFFLE and PRESERVE metrics are conflicting, since the results are vastly different for the two orders of optimization. This makes sense because they are completely opposite objectives, meant for different workflows (and never to be used together). There also appears to be significant interaction between the room non-utilization (RM_UTIL) metric and the Recitation Alignment by Course (REC_COURSE) metric. Even though there was only a minor change in the RM_UTIL metric between the two orders, there was a significant difference in the REC_COURSE metric. Again, this makes sense because when the RM_UTIL metric is optimized first, it essentially locks the class into a particular room. This makes it more difficult to find a timeslot such that the room is available and the timeslot has the correct alignment with the lecture.

In all other cases explored, the results of the optimization depend very little or not at all on the order of the metrics. This indicates that there is little to no interaction between the metrics. However, as we will see in the next experiment, this trend does not always hold when using many metrics (more than two). This points to more complex interactions involving three or more metrics, which are more difficult to discern.

| Metric | | A then B results | | B then A results | |
|---|---|---|---|---|---|
| A | B | Metric A | Metric B | Metric A | Metric B |
| PRESERVE | SHUFFLE | 34 | 482 | 409 | 107 |
| INS_COURSES | RM_UTIL | 1063 | 50.03 | 1063 | 50.03 |
| INS_COURSES | SPREAD | 1063 | 26 | 1067 | 25 |
| RM_UTIL | INS_TIMES | 50.03 | 0 | 50.03 | 0 |
| RM_UTIL | REC_COURSE | 50.03 | 62 | 50.12 | 2 |

Table 2: Metric results from experiment 2.

### 6.4 Experiment 3: A real world example

In this experiment, the automated scheduler will be used to create a complete schedule, taking into consideration all important metrics. Several different configurations of metrics will be tested and compared. We will also compare the results to the true schedule (HUMAN), which was created manually by the schedule administrator.

The metric configurations that were tested are shown in Table 3, with the results shown in Table 4. For schedules AUTO1 through AUTO4, we grouped metrics into four general categories: room preferences, instructor preferences, student preferences, and recitation preferences. Using the hierarchical method for combining metrics, we tried different orders of these metric categories. AUTO1 uses the order: room, instructor, student, recitation. AUTO2 uses the order: instructor, recitation, room, student. AUTO3 uses the order: student, recitation, instructor, room. Finally, AUTO4 uses the order: recitation, room, student, instructor.

From these schedules, it is clear that with a large number of metrics, it is not possible to simultaneously optimize all of them. Generally, higher priority metrics perform better, however, the importance of having a high priority is dependent on the metric. Metrics such as SEC_DIFF_TM and GRAD_OPT are able to perform well even when placed as low priority. On the other hand, metrics such as RM_UTIL and REC_SECTION see a large difference in

results between high and low priority.

The metric configurations for schedules AUTO5 through AUTO7 were chosen based on intuition and previous experience. However, there is no true science to finding the best configuration. Trial and error is an expected and necessary part of the process. AUTO5 uses the hierarchical method for combining metrics. AUTO6 also uses the hierarchical method, but allows some tolerance in a few of the metrics. AUTO7 uses a blended approach to combining metrics. Each of these configurations result in a high-quality schedule that performs better than the HUMAN schedule for most metrics. The different configurations have slightly different results. Determining the overall "best" schedule will require a value-judgement by the schedule administrator.

When comparing the results of the automated schedules to the human-created schedule, it is important to note that there a few of the "constraints" that were not adhered to in the HUMAN schedule. In some cases, this is due to an error by the schedule administrator. For example, there is one instance where the room occupancy constraint is not adhered to. The course CS 0134 has a enrollment cap of 50, however, it was assigned to a room with a occupancy of 32.

In other cases, however, the schedule administrator is able to use their "expert-knowledge" to override a constraint and make an assignment that is still valid. For example, there are several violations of the Instructor Courses constraint in the HUMAN schedule. This means that instructors were assigned to teach a course that was not included on their preference list. In the automated scheduler, this is framed as a constraint in order to avoid an instructor being assigned to teach a course that is outside their area of expertise. However, the schedule administrator is able to work outside of this constraint because they have knowledge of the background and prior experience of each instructor. They may know that an instructor is qualified to teach a course, even if they may not have listed that course on their preference list. This is the same reason that none of the automated schedules were able to perform as well as the HUMAN schedule on the TBD_INS metric. There were certain courses that no instructor listed as a preferred course, thus the automated schedule was forced to assign TBD.

There were also two violations of the Instructor Time constraint in the HUMAN schedule.

Recall that instructors submitted both their hard-constraint availability and their preferred teaching hours. There was a part-time instructor that listed their hard-constraint availability as 6PM-9PM. Although it is outside of their listed availability, the instructor agreed to teach a class that started at 5:30PM after discussing with the schedule administrator. There was another instructor that listed their hard-constraint availability as 2PM-10PM. The schedule administrator knew that this individual was a full-time instructor who is normally on campus much earlier than 2PM. It is likely that this instructor listed the 2PM start time by mistake. The schedule administrator was able to recognize that this was a mistake, and assigned the instructor to teach a course before 2PM. Because the automated scheduler does not have this "expert-knowledge", it is unable to work outside of the constraints. This is yet another reason why the iterative workflow with both automated and manual scheduling portions can be useful.

In general, the metric results show that the automated schedules perform better than the human-created schedule. These are very promising results, and indicate that the automated scheduler could be very useful to the University of Pittsburgh Computer Science department. However, there may also be aspects to the schedule quality that are not well-reflected in the metrics. For this reason, the automated scheduler is meant to be a tool for administrators to use, but manual evaluation and editing is still a necessary part of the process. Again, there is no single "best" schedule, but rather the schedule administrator must use intuition and value judgements to evaluate and compare.

| | AUTO1 | AUTO2 | AUTO3 | AUTO4 | AUTO5 | AUTO6 | AUTO7 |
|---|---|---|---|---|---|---|---|
| RM_UTIL | 13 / 1 / 0 / 0 | 7 / 1 / 0 / 0 | 2 / 1 / 0 / 0 | 10 / 1 / 0 / 0 | 5 / 1 / 0 / 0 | 5 / 1 / 5 / 0 | 1 / 16 / 0 / 0 |
| BAL_TBA | 12 / 1 / 0 / 0 | 6 / 1 / 0 / 0 | 1 / 1 / 0 / 0 | 9 / 1 / 0 / 0 | 6 / 1 / 0 / 0 | 6 / 1 / 1 / 0 | 1 / 32 / 0 / 0 |
| TBA_RM | exclude | exclude | exclude | exclude | exclude | exclude | exclude |
| RM_TYPE | exclude | exclude | exclude | exclude | exclude | exclude | exclude |
| INS_COURSES | 11 / 1 / 0 / 0 | 13 / 1 / 0 / 0 | 5 / 1 / 0 / 0 | 3 / 1 / 0 / 0 | 11 / 1 / 0 / 0 | 11 / 1 / 10 / 0 | 1 / 1024 / 0 / 0 |
| INS_TIMES | 10 / 1 / 0 /0 | 12 / 1 / 0 / 0 | 4 / 1 / 0 / 0 | 2 / 1 / 0 / 0 | 3 / 1 / 0 / 0 | 3 / 1 / 1 / 0 | 1 / 4 / 0 / 0 |
| COURSE_LEN | 9 / 1 / 0 / 0 | 11 / 1 / 0 / 0 | 3 / 1 / 0 / 0 | 1 / 1 / 0 / 0 | 1 / 1 / 0 / 0 | 1 / 1 / 0 / 0 | 1 / 1 / 0 / 0 |
| TBD_INS | exclude | exclude | exclude | exclude | exclude | exclude | exclude |
| SEC_DIFF_TM | 8 / 1 / 0 / 0 | 5 / 1 / 0 / 0 | 13 / 1 / 0 / 0 | 8 / 1 / 0 / 0 | 13 / 1 / 0 / 0 | 13 / 1 / 0 / 0 | 1 / 4096 / 0 / 0 |
| GRAD_OPT | 7 / 1 / 0 / 0 | 4 / 1 / 0 / 0 | 12 / 1 / 0 / 0 | 7 / 1 / 0 / 0 | 12 / 1 / 0 / 0 | 12 / 1 / 0 / 0 | 1 / 2048 / 0 / 0 |
| UNGRD_OPT | 6 / 1 / 0 / 0 | 3 / 1 / 0 / 0 | 11 / 1 / 0 / 0 | 6 / 1 / 0 / 0 | 10 / 1 / 0 / 0 | 10 / 1 / 4 / 0 | 1 / 512 / 0 / 0 |
| SPREAD | 5 / 1 / 0 / 0 | 2 / 1 / 0 / 0 | 10 / 1 / 0 / 0 | 5 / 1 / 0 / 0 | 7 / 1 / 0 / 0 | 7 / 1 / 4 / 0 | 1 / 64 / 0 / 0 |
| REC_OPT | 4 / 1 / 0 / 0 | 1 / 1 / 0 / 0 | 9 / 1 / 0 / 0 | 4 / 1 / 0 / 0 | 2 / 1 / 0 / 0 | 2 / 1 / 1 / 0 | 1 / 2 / 0 / 0 |
| REC_SECTION | 3 / 1 / 0 / 0 | 10 / 1 / 0 / 0 | 8 / 1 / 0 / 0 | 13 / 1 / 0 / 0 | 9 / 1 / 0 / 0 | 9 / 1 / 0 / 0 | 1 / 256 / 0 / 0 |
| REC_COURSE | 2 / 1 / 0 / 0 | 9 / 1 / 0 / 0 | 7 / 1 / 0 / 0 | 12 / 1 / 0 / 0 | 8 / 1 / 0 / 0 | 8 / 1 / 0 / 0 | 1 / 128 / 0 / 0 |
| REC_END | 1 / 1 / 0 / 0 | 8 / 1 / 0 / 0 | 6 / 1 / 0 / 0 | 11 / 1 / 0 / 0 | 4 / 1 / 0 / 0 | 4 / 10 / 0 / 0 | 1 / 8 / 0 / 0 |
| SHUFFLE | exclude | exclude | exclude | exclude | exclude | exclude | exclude |
| PRESERVE | exclude | exclude | exclude | exclude | exclude | exclude | exclude |

Table 3: Metric configuration from experiment 3. Each metric has a Priority / Weight / Absolute Tolerance / Relative Tolerance, shown in that order.

| Metric | HUMAN | AUTO1 | AUTO2 | AUTO3 | AUTO4 | AUTO5 | AUTO6 | AUTO7 |
|---|---|---|---|---|---|---|---|---|
| RM_UTIL | 77.77 | **50.03** | **64.33** | 102.93* | **64.00** | **55.67*** | **69.26*** | **52.95*** |
| BAL_TBA | 3 | **2** | 6 | **3*** | 5 | **3*** | 4* | **3*** |
| TBA_RM | 57 | **19** | **27** | 81 | **27** | **23** | **35** | **19** |
| RM_TYPE | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| INS_COURSES | 1138 | **1063** | **1063** | **1067*** | **1083*** | **1063** | **1073** | **1063*** |
| INS_TIMES | 4 | **1** | **0** | **1*** | **1*** | **1*** | **0** | **1*** |
| COURSE_LEN | 3 | **3** | **3** | **3** | **3** | **3** | **1** | **3*** |
| TBD_INS | 1 | 3 | 3 | 3 | 4 | 3 | 3 | 3 |
| SEC_DIFF_TM | 0 | **0** | **0** | **0** | **0** | **0** | **0** | 1* |
| GRAD_OPT | 30 | **10** | **10*** | **10** | **10** | **10** | **10** | **10*** |
| UNGRD_OPT | 8 | **5*** | **5** | **5*** | **5*** | **5*** | **6*** | **6*** |
| SPREAD | 36 | 39* | **27** | **25** | **26*** | **26** | **30** | **29*** |
| REC_OPT | 4 | **0** | **0** | **0** | **0** | **0** | **1** | **0*** |
| REC_SECTION | 0 | 46* | **0** | **0** | **0*** | **0** | **0** | 4* |
| REC_COURSE | 0 | 256* | **0** | **0** | **0*** | 2* | **0** | 31* |
| REC_END | 28 | 55* | **0** | **0** | **0*** | 66 | **8*** | **28*** |
| SHUFFLE | N/A | 205 | 212 | 219 | 211 | 218 | 212 | 212 |
| PRESERVE | N/A | 311 | 304 | 297 | 305 | 298 | 304 | 304 |

Table 4: Metric results from experiment 3. A star (*) indicates time limit was reached before an optimal solution could be found or proved. Bold font indicates that the metric performed equally well or better than the HUMAN schedule.

### 6.5 Experiment 4: Iterative workflow

This experiment will demonstrate the iterative workflow. The schedule will be created in a series of three stages. These stages will be similar to the stages that the department uses when scheduling manually. After each stage, certain assignments will be locked-in so that the automated scheduler cannot change them in later stages.

In the first stage, we scheduled the times for the graduate lectures. There are a limited number of graduate classes offered each semester, so it is very important that there are not conflicts between graduate courses. For this reason, we scheduled these classes first. This allows the automated scheduler to easily find optimal times to offer these classes without considering the additional complexity of the undergraduate classes. Since graduate students do not enroll in undergraduate courses and vice versa, it is not necessary to consider class conflicts between these two sets of courses.

A CSV file of only the graduate lectures was uploaded to the scheduling tool and the automated scheduler. The only metric that was used in this stage is the Graduate Options (GRAD_OPT) metric. Although the automated scheduler also makes instructor and room assignments, we are not concerned with these at this point. Making good instructor and room assignments requires a view of the entire schedule, in order to ensure that these assignments abide by global preferences and availability. Once the automated scheduler was finished, the timeslots for these graduate classes were locked-in.

In the second stage, we added the undergraduate classes to the schedule. This was done using the add classes from CSV file feature. At this point, since all lectures were included in the schedule, we were able to make instructor and room assignments as well. The schedule was sent to the automated schedule (with the timeslots locked-in for the graduate courses). The metrics were optimized using the hierarchical approach in the following priority order: SEC_DIFF_TM, SPREAD, UNGRD_OPT, INS_COURSES, BAL_TBA, RM_UTIL, and INS_TIMES.

In the third and final stage, the recitation and lab classes were added, again using the import from CSV feature. For most courses, there are multiple recitation options for a single lecture. Since there are multiple options for students, it is not super important to

consider how these classes may conflict with other classes. Additionally, since recitations are typically taught by TAs rather than instructors, these classes do not play a role in instructor availability or preferences. TAs are not assigned during this scheduling process, but rather at a later date closer to the start of the semester. Thus, the only important things to consider when scheduling recitations is 1) the timing of the class relative to the lecture and 2) room availability. The metrics are optimized using the hierarchical approach in the following priority order: REC_SECTION, REC_COURSE, RM_UTIL, BAL_TBA, REC_END, and REC_OPT. The metric values from the resulting schedule are shown in Table 5. Results from intermediate stages are not shown because they are not comparable to other results, given that they are not over the entire schedule.

Scheduling in an iterative manner can reduce the complexity at each stage since there are fewer classes/assignments being considered at a single time. This reduced complexity makes it easier for the scheduler to find optimal solutions at each step. However, the reduced complexity can come at a cost because there is less flexibility and assignments are made without a global view of the schedule.

For example, scheduling recitations after all lecture assignments are already locked-in reduces the complexity, but results in a poorer schedule due to the lack of flexibility. Since lecture assignments are locked-in, the scheduler has limited options for finding rooms/timeslots for the recitations. If all classes are scheduled at once, the scheduler can try to move both the lectures and the recitations in order to find a configuration that is optimal to both. However, with the lectures locked-in, the scheduler does not have this flexibility. As a result, the recitation placement metrics (REC_SECTION, REC_COURSE, REC_END) perform worse than the results from Experiment 3.

This experiment highlighted just one way to use the iterative workflow. Another option would be to consider all classes at once, but make different assignments at different stages (e.g. assign timeslots, then rooms, then instructors). This might work well for an institution that has relatively few constraints, however, for a large schedule like the University of Pittsburgh Computer Science department, this would surely lead to an inferior schedule due to the lack of global perspective.

One final option would be to consider all classes, assignments, and metrics at once. Once

the automated scheduler produces a schedule, the user can lock-in any assignments that they are satisfied with. They can then resubmit the schedule to the automated scheduler in order to change the assignments that they were not satisfied with. To ensure that the unsatisfactory assignment are different the second time around, the user must use a different configuration of metrics or use the SHUFFLE metric on top of the previous configuration.

## 6.6 Summary

In this chapter, we experimented with different configurations of the automated scheduler. In Experiment 1, we demonstrated different options for combining two conflicting metrics. In Experiment 2, we tried to identify conflicting metrics. We found that two-way conflicts are very minimal, pointing to more complex (3-way or larger) interactions. In Experiment 3, we used the automated scheduler to create a full schedule for the Pitt Computer Science department, using all relevant metrics. The results were very promising and outperformed the manually-created schedule for most metrics. In Experiment 4, we demonstrated one example of an iterative workflow, which uses multiple rounds of automated scheduling. The key takeaway from these four experiments is that the configuration of metrics for the automated scheduler can have a large impact on the quality of the results. Although some intuition can be used for setting the configuration, it ultimately requires some trial and error due to the complex nature of the problem.

| | Results |
|---|---|
| RM_UTIL | **52.57** |
| BAL_TBA | **2** |
| TBA_RM | **17** |
| RM_TYPE | N/A |
| INS_COURSES | **1077** |
| INS_TIMES | **2** |
| COURSE_LEN | 4 |
| TBD_INS | 4 |
| SEC_DIFF_TM | **0** |
| GRAD_OPT | **10** |
| UNGRD_OPT | **5** |
| SPREAD | **25** |
| REC_OPT | **0** |
| REC_SECTION | **0** |
| REC_COURSE | **2** |
| REC_END | 40 |
| SHUFFLE | N/A |
| PRESERVE | N/A |

Table 5: Metric results from experiment 4. Bold font indicates that the metric performed equally well or better than the HUMAN schedule.

# 7.0 Conclusions and future work

The University of Pittsburgh Computer Science department has found the manual scheduling interface extremely useful. Users of the system have noted that the system makes their jobs significantly easier and reduces the amount of time that it takes to create a schedule. Our tool has allowed them to create higher quality schedules, since it is easier to recognize problem areas in the schedule using our tool than it was using previous scheduling tools.

One of the current limitations of the tool is that it does not directly integrate with the existing software that is used at Pitt for course scheduling management (PeopleSoft). This means that when changes are made to a schedule within our tool, they do not get automatically reflected in the official schedule. The user must keep track of any changes that were made within our tool and apply them to the official schedule in PeopleSoft. Conversely, if changes are made to the official schedule, the user must remember to apply the changes within our tool. Keeping these versions synchronized has proven to be a challenge.

Discussions and work to integrate this tool with PeopleSoft and other existing systems has begun. However, one of the challenges that has been noted is fitting the tool into existing scheduling workflows. There are several stages to the scheduling process at Pitt, and a multitude of individuals involved at each stage. Additionally, different departments have different workflows, depending on the size and needs of the department. To integrate successfully, our tool needs to be an option for departments to use, while not disrupting or replacing existing workflows.

Although the results from the automated scheduler are very promising, the feature has not been used in practice yet. Because the existing workflow is to build off of previous version of the schedule (rather than a complete redesign), it will be a challenge to integrate the automated scheduler into the current workflow. Without an automated way to import a schedule into PeopleSoft, all changes will have to be made manually. Since the automated scheduler does not have any bias towards keeping the schedule as it was in previous terms, this means that every assignment could potentially be changed and need to be manually updated in PeopleSoft as well. To avoid this headache, the department is still using the

existing workflow of making small changes, rather than complete redesigns of the schedule.

Best practices for utilizing multiple metrics is still an open problem. The experiments in this thesis have shown that the hierarchical approach is often more effective because it reduces complexity. With the blended approach, the combined objective can become too complex and cause the solver to have difficulty finding an optimal solution. These takeaways are likely very dependent on the size and complexity of the schedule, as well as the metrics that are being considered. Ultimately, when using multiple objectives, there is no single solution and a value judgement must be made to determine the "best" schedule.

## Bibliography


[1] Mohammed Al-Betar, Ahamad Tajudin Khader, and Munir Zaman. University course timetabling using a hybrid harmony search metaheuristic algorithm. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42:664–681, 09 2012.

[2] Hameed Al-Qaheri, Mohamad Hasan, and Raed Al-Husain. A decision support system for a three-stage university course scheduler with an application to college of business administration, kuwait university. *International Journal of Data Analysis and Information Systems*, Vol. 3:95–110, 12 2011.

[3] M Bakır and Cihan Aksop. A 0-1 integer programming approach to a university timetabling problem. *Hacettepe Journal of Mathematics and Statistics*, 37, 01 2008.

[4] Natashia Boland, Barry D. Hughes, Liam T.G. Merlot, and Peter J. Stuckey. New integer linear programming approaches for course timetabling. *Computers & Operations Research*, 35(7):2209–2233, 2008. Part Special Issue: Includes selected papers presented at the ECCO'04 European Conference on combinatorial Optimization.

[5] S Daskalaki and T Birbas. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, 160(1):106–120, 2005. Applications of Mathematical Programming Models.

[6] S Daskalaki, T Birbas, and E Housos. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*, 153(1):117–135, 2004. Timetabling and Rostering.

[7] M. Dimopoulou and P. Miliotis. Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, 130(1):202–213, 2001.

[8] Najla'a Ateeq Mohammed Draib and A. B. Sultan. An autonomous software approach to enhance information sharing in university course timetable planning. 2015.

[9] L.R. Foulds and D.G. Johnson. Slotmanager: a microcomputer-based decision support system for university timetabling. *Decision Support Systems*, 27(4):367–381, 2000.

[10] Sehraneh Ghaemi, Mohammad Vakili, and Ali Aghagolzadeh. Using a genetic algorithm optimizer tool to solve university timetable scheduling problem. pages 1 – 4, 03 2007.

[11] Mehmet Güray Güler and Ebru GEÇİCİ. A spreadsheet-based decision support system for examination timetabling. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, 28:1584–1598, 05 2020.

[12] Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86, 07 2015.

[13] Melissa Humphrey and Amarjit Singh. Reducing class-scheduling conflicts using linear programming. *Journal of Professional Issues in Engineering Education and Practice*, 143(4):05017004, 2017.

[14] Sk. Imran Hossain, M.A.H. Akhand, M.I.R. Shuvo, N. Siddique, and H. Adeli. Optimization of university course scheduling problem using particle swarm optimization with selective search. *Expert Systems with Applications*, 127:9–24, 2019.

[15] Nergiz A. Ismayilova, Mujgan Sağir, and Rafail N. Gasimov. A multiobjective faculty–course–time slot assignment problem with preferences. *Mathematical and Computer Modelling*, 46(7):1017–1029, 2007. Decision Making with the Analytic Hierarchy Process and the Analytic Network Process.

[16] Jaime Miranda, Pablo A. Rey, and José M. Robles. udpskeduler: A web architecture based decision support system for course and classroom scheduling. *Decision Support Systems*, 52(2):505–513, 2012.

[17] Cristian Palma and Patrick Bornhardt. Considering section balance in an integer optimization model for the curriculum-based course timetabling problem. *Mathematics*, 8, 10 2020.

[18] Atiq W. Siddiqui, Syed Arshad Raza, and Zeeshan Muhammad Tariq. A web-based group decision support system for academic term preparation. *Decision Support Systems*, 114:1–17, 2018.

[19] Ahmed Wasfy and Fadi Aloul. Solving the university class scheduling problem using advanced ilp techniques. 01 2007.