Class Scheduling Automation Using Integer Programming: Aligning Professors, Courses, and Time Slots

MATH 499: Capstone Project

May 4, 2025

NAZARBAYEV
UNIVERSITY

Submitted for **MATH 499: Capstone Project** at Department of Mathematics, the School of Sciences and Humanities, Nazarbayev University

Student Name:

Maksat Manapkhan - ID: 201911345

Capstone Project
Instructor Name: **Adilet Otemissov**

In submitting this work we are indicating that we have read the University's Academic Integrity Policy. We declare that all material in this assessment is our own work except where there is clear acknowledgment and reference to the work of others.

1 Abstract

This project addresses the faculty-course-section scheduling problem in an academic institution by employing integer programming techniques to optimize professor assignments while satisfying various constraints. The problem is formulated as an Integer Linear Program (ILP) and solved using the PuLP optimization library.

The model considers multiple constraints including professor workload limits, course preferences, section availability, and time-slot assignments. The project is divided two phases: (1) assigning professors to course sections based on preference rankings and workload requirements and (2) scheduling these assignments into available classrooms and time slots while ensuring no conflicts occur. Numerical experiments validate the model's effectiveness in maximizing professor satisfaction while maintaining feasibility across various scheduling scenarios.

The approach provides a scalable and automated alternative to manual scheduling, reducing administrative effort and improving overall scheduling efficiency. Future extensions of this work could involve adding more constraints for better professors satisfaction, assigning recitations or other types of classes that university provide and adapting the model for broader academic scheduling applications.

2 Introduction

Scheduling problems The scheduling problems involves allocation of the resources such as workers, machines, roads and etc. to tasks and satisfying certain constraints such as work load, time, maximum worker number and etc [1]. This type of problems could be solved using different approaches, including exact solutions, heuristic methods, and metaheuristic methods. Each approach has its advantages and disadvantages, and the choice of method depends on factors such as the complexity of the problem, time constraints, and the required level of optimality. Exact methods such as linear programming, integer programming, and branch-and-bound are used to find the optimal solution for medium-sized problems but can be computationally expensive for large-scale problems [2]. Heuristic methods such as greedy algorithms and priority rules are employed when exact methods are too slow or infeasible; these methods provide quick solutions, although they may not always be optimal [2]. For highly complex, large-scale problems where a balance between computational efficiency and solution quality is required, metaheuristic methods are commonly used [2]. These methods, including genetic algorithms, simulated annealing, and ant colony optimization, are widely applied in real-world scenarios.

Project description In this project, the problem involves a significant number of variables and constraints, making it necessary to employ a robust and efficient optimization technique. The first part of the problem consists of 2,222 variables and 879 constraints, representing the assignment of 25 professors to 57 sections and 29 courses, each with multiple sections. The second part of the problem further scales in complexity, containing 3,035 variables and 7,075 constraints. Given the problem's size and nature, the PuLP library is used to define and PULP_CBC_CMD default open source CBC solver was used to solve the problem, offering several solution

algorithms, which automatically chooses the algorithm to use for solving the problem. For CBC solver uses for the Linear Program with only continuous variables simplex based method is applied and for the Mixed Integer Programs (MIP) that can contain either continuous and integer variables the Branch-and-Bound algorithm is used with simplex method for solving the LP relaxations [3].

Branch-and-Bound framework The Branch-and-Bound framework (B&B) is a family of the algorithms that are used to find exact solutions for the NP-hard problems such as MIP. B&B family contains lots of other algorithms that have main solving format as a base and each group is based on the choice of the search strategy (i.e. Depth First Search, Breadth First Search and etc.), branching strategy (i.e. Binary Branching, Wide Branching), prunning rules (Lower Bounds, Cutting Planes, Dominance Rules)[4]. Format which is shared by all the algorithms partions parent solution space into several subproblems and subspaces which are further solved by simplex algorithm and this process is repeated until the stopping criteria is met such as tolerance, all nodes fathomed, node limit and etc. To provide a clearer understanding of the algorithm, we now present its step-by-step formulation.

An optimization problem consists of a search space X containing all feasible solutions and an objective function $f : X \to \mathbb{R}$, which evaluates the quality of each solution. The objective is to determine an optimal solution x^* such that:

$$x^* \in \arg\max_{x \in X} f(x)$$

The B&B is a widely used method for solving such problems by systematically exploring subsets of the search space. It builds a search tree T, where each node represents a subproblem corresponding to a subset of X. To improve efficiency, the algorithm maintains a global best-known solution, termed the incumbent solution $\hat{x} \in X$.

During each iteration, a subproblem $S \subseteq X$ is selected from a list L of unexplored subproblems. If a solution $\hat{x}' \in S$ is found that improves upon the incumbent $f(\hat{x}') > f(\hat{x})$ for maximization problems, the incumbent is updated. If, however, it can be proven that no solution in S has a better objective value than $\hat{x}$ (i.e., $\forall x \in S, f(x) \geq f(\hat{x})$ or $f(x) \leq f(\hat{x})$), then S is pruned from further consideration.

If pruning is not possible, S is divided into smaller subproblems $S_1, S_2, \ldots, S_r$, which are then incorporated into T for subsequent exploration. This process continues until no unexplored subproblems remain.

Upon termination, the best incumbent solution is returned as the optimal solution. Since subproblems are only discarded when they are confirmed to contain no better solution, the final incumbent must satisfy:

$$\hat{x} \in \arg\min_{x \in X} f(x).$$

Cutting planes Cutting planes is an iterative method for solving the MIP problems, by solving LP relaxations of the MIP problem. If the optimal solution for the LP relaxation satisfies all the integrality constraints of the MIP then it is also an optimal solution for the MIP problem, however this is a very rare situation and we tighten

the constraints in order to have all vertices integer values. Since for any feasible region X of the initial LP problem there exists an ideal formulation and it could be formulated with the same size by the exponential number of the constraints the main goal is to find the best formulation of the feasible region.

Gomory cuts Gomory fractional cuts is a class of the methods for solving MIP problems, first was introduced by Ralph E. Gomory in 1958 [5]. In the method proposed by him was created a new method of cutting planes which will satisfy not only the optimality and initial constraints but also the integrality constraints of the LP relaxation. The method involves adding and subtracting for all coefficients of the constraints in order to partion the integers and fractions.

$$\begin{aligned} \textbf{max} \quad & c^T x \\ \textbf{subject to} \quad & Ax = b \\ & x \geq 0, \quad x \in \{0\} \cup Z^+ \end{aligned}$$

For the IP problem above it is not required that any value in matrix A or vector b to be integers. Let A be $m \times n$ matrix and b be $m \times 1$ vector. Now for the better understanding lets consider first inequality constraint after converting to the standard form with added slack variables:

$$a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + a_{1,3} \cdot x_3 + \ldots + a_{1,n} \cdot x_n = b_1$$

Since, in Gomory cut, the coefficients are separated into fraction and integer parts, after separation the first constraint will look like:

$$(\lfloor a_{1,1} \rfloor + (a_{1,1} - \lfloor a_{1,1} \rfloor)) \cdot x_1 + \ldots + (\lfloor a_{1,n} \rfloor + (a_{1,n} - \lfloor a_{1,n} \rfloor)) \cdot x_n = \lfloor b_1 \rfloor + (b_1 - \lfloor b_1 \rfloor)$$

Where $a_{1,j} - \lfloor a_{1,j} \rfloor$ is the fraction part of the coefficient $a_{1,j}$ and it can be replaced by $f_{1,j} = a_{1,j} - \lfloor a_{1,j} \rfloor$. After opening the brackets and grouping all the fraction parts in the left side and integer parts in the right side we get:

$$f_{1,1} \cdot x_{1,1} + f_{1,2} \cdot x_{1,2} + \ldots + f_{1,n} \cdot x_{1,n} - f_{b,1} = \lfloor b \rfloor - \lfloor a_{1,1} \rfloor \cdot x_1 - \ldots - \lfloor a_{1,n} \rfloor \cdot x_n$$

From the equation above Gomory cut could be derived as follows:

$$f_{1,1} \cdot x_{1,1} + f_{1,2} \cdot x_{1,2} + \ldots + f_{1,n} \cdot x_{1,n} \geq f_{b,1}$$

By applying cut iteratively for one constraint at each iteration we obtain the optimal solution for our IP problem.

Example For the better understanding of how the Gomory cut works lets consider the following example (taken from the [6]):

$$\begin{aligned} \textbf{max} \quad & z = 8x_1 + 5x_2 \\ \textbf{subject to} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0, \quad x_1, x_2 \in \{0\} \cup Z^+ \end{aligned}$$

After applying simplex method to solve the initial LP problem we get optimal tableau:

		x_1	x_2	s_1	s_2
$-z$	-41.25	0	0	-1.25	-0.75
x_1	3.75	1	0	-1.25	0.25
x_2	2.25	0	1	2.25	-0.25

Optimal solution $z^* = 41.25$ and basic variables are $x_1 = 3.75, x_2 = 2.25$, since they are not integers we apply Gomory cut. For the first cut lets consider first constraint $x_1 - 1.25s_1 + 0.25s_2 = 3.75$. Get the fractional parts from the equation

$$(1+0)x_1 + (-2+0.75)s_1 + (0+0.25)s_2 = (3+0.75).$$

Gomory cut will look like:

$$0.75s_1 + 0.25s_2 \geq 0.75.$$

In order to rewrite it in terms of decision variables as an additional constraint we can substitute values of s_1 and s_2 from the first two constraints $s_1 = 6 - x_1 - x_2$ and $s_2 = 45 - 9x_1 - 5x_2$. The additional constraint after Gomory cut is $3x_1 + 2x_2 \leq 15$ and our problem will look like:

$$\begin{aligned} \textbf{max} \quad & z = 8x_1 + 5x_2 \\ \textbf{subject to} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & 3x_1 + 2x_2 \leq 15 \\ & x_1, x_2 \geq 0, \quad x_1, x_2 \in \{0\} \cup Z^+ \end{aligned}$$

By plotting the constraints we can solve the problem by graphical method and find optimal solution to our Integer Linear Program (ILP)[6].

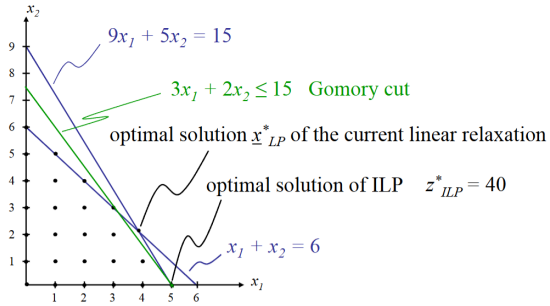

Figure 1: Gomory cut method.

3 Main body

Problem is created for application in Mathematics Department at Nazarbayev University which requires input data such as professors preferences for the courses

that they want to teach and time-slot rankings which will be collected from the survey. The files about professors load and required course list for next term will be received will be decided and received from department chair and file with list of rooms and available time slots with the capacities will be received from the Registrar office.

The problem is divided into two parts where the first part matches professors to the sections of the courses, and second matches the output of the first part and the classroom at certain time. This is done in order to have an option for human intervention between phases to make some adjustments.

3.1 Data collection

LP model takes as an input four excel files about professors, courses, room and time slots and professors preferences. File about professors contains information about the number of sections they are required to teach next term, points of each professor for courses and time slots and pre-assigned courses. An example is given in Table 1 where professor 1 has to teach three sections next term and has 100 course and time points which is given by default and further is increased or decreased based on their performance, moreover he is also pre-assigned for section 1L of MATH 161, which also be taken into account by the model so he will be assigned to two extra sections of the courses from his preference list, however professor 2 is not pre-assigned to any course so he will be assigned to three sections from his preference list.

professors	load 2	course points	time points	course 1	section 1
professor 1	3	100	100	MATH 161	1L
professor 2	3	100	100		

Table 1: Professors input file

Main input file that will contain preferences for courses, times and desire of teaching several sections of one course of all professor. In Table 2 professor 1 has put MATH 109 to the 3rd place and MATH 161 to the 8th place so first five courses will be counted and others dropped such as MATH 161. For the time slots all of them will be considered.

professor	MATH 109	MATH 161	...	MWF 9:00-10:00	...	TR 9:00-10:15	...
professor 1	3	8	...	4	...	8	...
professor 2	4	1	...	10	...	2	...

Table 2: Preferences input file

3.2 Professor to course section assigning

Objective function. Objective function of the first part is constructed in order to maximize preferences of the professors for the courses for which they are assigned. Which iterates over all professors (N_p), courses (N_c) and each section (N_s) of the course for which professor is assigned.

$$\max \quad \zeta = \sum_{p}^{N_p} \sum_{c}^{N_c} \sum_{s}^{N_s} x_{p,c,s} \cdot w_{p,c} - 100u_p - 100v_c - 100r_p - 100q_{p,c},$$

where $x_{p,c,s}$ is equal to 1 if professor p is assigned to section s of course c and 0 otherwise, and $w_{p,c}$ is preference weight of professor p for the course c the values of which are initialized based on survey data. $u_p, v_c, r_p, q_{p,c}$ are relaxation variables from the constraints in order to punish the problem based on some criteria which will be explained further.

Professor's load constraint. This constraint is formulated to ensure that no professors are assigned to more sections than their prescribed load. By summing professors assignings over all courses and sections we calculate for how many classes he is assigned and equalize it to professors load $load_p$, relaxation variable v_p is also added in order to avoid non-feasibility of the problem and then check for how many classes each professor was not assigned.

$$\sum_{c}^{N_c} \sum_{s}^{N_s} x_{p,c,s} + v_p = load_p$$

One professor for one section constraint. Constraint to avoid assigning several professors to one section of the course is constructed by summing assigning of all professors for the section and making it less or equal to 1.

$$\sum_{p}^{N_p} x_{p,c,s} \leq 1$$

All sections assigned constraint. Constraint to assign all section of the course was constructed by summing all assignings for that course and is equalized to the number of sections $section\_num_c$ for that course, relaxation variable u_c for courses was also added to avoid the problem being non-feasible and to get partial solution by assigning as much professors to courses.

$$\sum_{p}^{N_p} \sum_{s}^{N_s} x_{p,c,s} + u_c = section\_num_c$$

Professors teach preferred courses. The constraint that professor should teach a course from his preferences is written as a sum of professors assigning $x_{p,c_{pref},s}$ over his preferred courses equal to his load $load_p$. Where r_p is the relaxation variable which was added to have partial feasible solution if not all professors are assigned as they were supposed to and was added to the objective function to minimize it.

$$\sum_{c_{pref}}^{N_{c_{pref}}} x_{p,c_{pref},s} + r_p = load_p$$

Professors can not teach a non-preferred course. Additional inverse constraint that professor can not teach any course not from his preference list is added and written as sum of $x_{p,c_{not\_preferred},s}$ assigned courses not in the professors preference list $c_{not\_preferred}$ must be equal to zero.

$$\sum_{c_{not\_pref}}^{N_{c_{not\_pref}}} x_{p,c_{not\_pref},s} = 0$$

Several section preference constraint. For the professors who prefer not to teach several sections of one course there is an additional constraint by summing professors assigning for the course c for all sections s to the corresponding course and that should be less or equal to 1. The relaxation variable $q_{p,c}$ is also added for the situations when assigning the professor who choose "not preferred" for several sections of one course is the only solution for the problem.

$$\sum_{s}^{N_s} x_{p,c,s} + q_{p,c} \leq 1$$

3.3 Professor-section to time-slot-class assigning.

Objective function Objective function was formulated to maximize professors satisfaction for their assigned times to teach their courses.

$$\max \quad \zeta = \sum_{s_p}^{N_s} \sum_{t_r}^{N_t} x_{s_p,t_r} \cdot w_{p,t} - 100 \cdot j_{c,s},$$

where x_{s_p,t_r} is an LP variable that is equal to 1 if professor p who was predefined for section s from the first part is assigned to room r at time t and 0 otherwise, and where $w_{p,t}$ is professor p's preferred weight to teach at time t, N_s and N_t represent the total number of sections of the courses and timeslots for each room.

Assigning section only once constraint For assigning one section of course only once for all time slots this constraint is added. This constraint was defined as summation each section s of course c for all time slots t out of N_t for all rooms r out of N_r being equal to 1, meaning that each section of each course will be assigned only once. The relaxation variable j_s was added in order to avoid the non feasibility of the problem.

$$\sum_{t}^{N_t} \sum_{r}^{N_r} x_{s,t,r} + j_{c,s} = 1, \forall s \in S$$

Room capacity constraint Constraint to check if the sections capacity fits to the room was formulated with application of big-M constant as follows. Capacity of the course c multiplied to the x value for the professor-course pair c and time slot t should be less or equal to the capacity of room r plus the big-M constant multiplied to slack variable of course-professor pair. Big-M constant was applied in order if

the course c is not assigned to ignore the constraint, it should be big enough to be always true if $j_c = 1$ to deactivate the constraint and should not be too large to cause any numerical issues.

$$cap_{c,s} \cdot x_{c,s,t} \leq cap_r + M \cdot j_{c,s}$$

No room double booking constraint To prevent booking of one room for one time slot several times the constraint is added and is constructed by summing x values for each course-professor c pair for one time slot t and making sure that it is less or equal to 1, meaning that it is either booked or not.

$$\sum_{c}^{N_c} x_{c,t} \leq 1$$

No professor-section double booking constraint In order to avoid situations where one professor is assigned to several rooms at one time slot the constraint is added. The constraint iterates over rooms r out of N_r at fixed time slot t for fixed professor p and must be less or equal to one, where $x_{p,t,r}$ is the LP variable that represents if the professor p is assigned to the room r at time t.

$$\sum_{r}^{N_r} x_{p,t,r} \leq 1$$

300/400/graduate level courses taught at different time constraints. Since there are restriction from the school that courses of 300, 400 and graduate level can not be taught at the same time for students to be able register to all of them this constraint is added. It is represented for each course level group as a sum of LP variable $x_{c_{level},t}$ over number of sections of course c_{level} less or equal to 1. Where $N_{s_{c300}}, N_{s_{c400}}, N_{s_{c_{grad}}}$ are the variables that represent number of sections for course from the list with exact course level such as 300, 400 or graduate:

$$\sum_{c300} \frac{x_{c300,t}}{N_{s_{c300}}} \leq 1, \qquad \sum_{c400} \frac{x_{c400,t}}{N_{s_{c400}}} \leq 1, \qquad \sum_{c_grad} \frac{x_{c_{grad},t}}{N_{s_{c_{grad}}}} \leq 1$$

3.4 Example

3.4.1 Input data

Let us formulate an IP problem on a small problem based on disscussions above.

professors	load	course points	time points	course 1	section 1
professor A	1	100	100	MATH 161	1L
professor B	1	100	130		
professor C	2	110	100		

Table 3: Example: professors file

professor	MATH 161	MATH 361	MATH 351	Several sections
professor A	2	1	3	-1
professor B	3	2	1	0
professor C	1	2	3	1

Table 4: Example: preferences file course part

course code	course title	section number	section type	capacity
MATH 161	Calculus I	1	L	50
MATH 161	Calculus I	2	L	30
MATH 351	Numerical Methods	1	L	60
MATH 361	Real Analysis	1	L	40

Table 5: Example: courses file

From the input tables we can get that professor A was preassigned to the first section of Calculus I class, and that professor C has a bit higher points for the course assigning while professor B has higher priority for the time assigning. In the following examples let $x_{A,161,1}$ be the binary LP variable which is 1 if professor A is assigned to the 1st section of MATH 161 and etc.

3.4.2 Part 1: Professor to course assigning

Professors load constraint:

$$\begin{aligned} x_{A,161,1} + x_{A,161,2} + x_{A,361,1} + x_{A,351,1} + p_A &= 1 \\ x_{B,161,1} + x_{B,161,2} + x_{B,361,1} + x_{B,351,1} + p_B &= 1 \\ x_{C,161,1} + x_{C,161,2} + x_{C,361,1} + x_{C,351,1} + p_C &= 1 \end{aligned}$$

One professor for one section constraint:

$$\begin{aligned} x_{A,161,1} + x_{B,161,1} + x_{C,161,1} &\leq 1 \\ x_{A,161,2} + x_{B,161,2} + x_{C,161,2} &\leq 1 \\ x_{A,361,1} + x_{B,361,1} + x_{C,361,1} &\leq 1 \\ x_{A,351,1} + x_{B,351,1} + x_{C,351,1} &\leq 1 \end{aligned}$$

All sections are assigned constraint:

$$\begin{aligned} x_{A,161,1} + x_{A,161,2} + x_{B,161,1} + x_{B,161,2} + x_{C,161,1} + x_{C,161,2} + course\_relax_{161} &= 2 \\ x_{A,361,1} + x_{B,361,1} + x_{C,361,1} + \text{course\_relax}_{361} &= 1 \\ x_{A,351,1} + x_{B,351,1} + x_{C,351,1} + course\_relax_{351} &= 1 \end{aligned}$$

Professors teach preferred courses constraint: For this constraint lets take top 2 choice and drop 3rd place for each professor from their preference table, so the

constraint equalities will look like:

$$\begin{aligned} x_{A,161,1} + x_{A,161,2} + x_{A,361,1} + r_A &= 1 \\ x_{B,361,1} + x_{B,351,1} + r_B &= 1 \\ x_{C,161,1} + x_{C,161,1} + x_{C,361,1} + r_C &= 2 \end{aligned}$$

Professors can not teach non-preferred course constraint: Respectively for this consider course on the 3rd place in professor preference as not preferred course.

$$\begin{aligned} x_{A,351,1} &= 0 \\ x_{B,161,1} + x_{B,161,2} &= 0 \\ x_{C,351,1} &= 0 \end{aligned}$$

Several section preference constraint: Since this constraint is included only for the professors who chose that they would not like to teach several sections of same course in the survey, lets look at the professors responses in Table 4. Professor A chose "do not prefer", B chose "does not matter" and professor C chose that he prefers, based on the responses the we consider only if professor does not want to teach so the constraint with relaxation variable $q_{A,c}$ will be added to the objective function:

$$\begin{aligned} x_{A,161,1} + x_{A,161,2} + q_{A,161} &\leq 1 \\ x_{A,361,1} + q_{A,361} &\leq 1 \\ x_{A,351,1} + q_{A,351} &\leq 1 \end{aligned}$$

Objective function:

$$\begin{aligned} \max \zeta = {} & x_{A,161,1} \cdot w_{A,161} + x_{A,161,2} \cdot w_{A,161} + x_{A,361,1} \cdot w_{A,361} + x_{A,351,1} \cdot w_{A,351} + \\ & + x_{B,161,1} \cdot w_{B,161} + x_{B,161,2} \cdot w_{B,161} + x_{B,361,1} \cdot w_{B,361} + x_{B,351,1} \cdot w_{B,351} + \\ & + x_{C,161,1} \cdot w_{C,161} + x_{C,161,2} \cdot w_{C,161} + x_{C,361,1} \cdot w_{C,361} + x_{C,351,1} \cdot w_{C,351} \end{aligned}$$

3.4.3 Part 2: Professor course pair assigning to the time slot room pairs

Suppose that from the first part we got output as in the table(table with assigned prof course section)

professor	course	section	capacity	id
professor A	MATH 161	1L	50	0
professor B	MATH 351	1L	40	1
professor C	MATH 161	2L	60	2
professor C	MATH 361	1L	30	3

Table 6: Example: assigned professor to course sections

professor	MWF 9:00-10:00	MWF 10:00-11:00	TR 9:00-10:15	TR 10:30-11:45
professor A	3	1	2	4
professor B	1	2	4	3
professor C	3	2	4	1

Table 7: Example: preferences file time part

day	time	room	capacity
M W F	9:00 AM - 9:50 AM	7.101	45
M W F	10:00 AM - 10:50 AM	7.101	45
T R	9:00 AM - 10:15 AM	7.102	70
T R	10:30 AM - 11:45 AM	7.102	70

Table 8: Example: time-room file

Assigning section only once constraint:

$$x_{161,1,9MWF,101} + x_{161,1,10MWF,101} + x_{161,1,9TR,101} + x_{161,1,10TR,101} + slack_{161_1} = 1$$
$$x_{161,2,9MWF,101} + x_{161,2,10MWF,101} + x_{161,2,9TR,101} + x_{161,2,10TR,101} + slack_{161_2} = 1$$
$$x_{351,1,9MWF,101} + x_{351,1,10MWF,101} + x_{351,1,9TR,101} + x_{351,1,10TR,101} + slack_{351_1} = 1$$
$$x_{361,1,9MWF,101} + x_{361,1,10MWF,101} + x_{361,1,9TR,101} + x_{361,1,10TR,101} + slack_{361_1} = 1$$

Room capacity constraint:

$$\begin{aligned} 50 \cdot x_{161,9MWF} &\leq 45 + M \cdot slack_{161_1} \\ 50 \cdot x_{161,10MWF} &\leq 45 + M \cdot slack_{161_1} \\ &\vdots \\ 40 \cdot x_{361,9TR} &\leq 70 + M \cdot slack_{361_1} \\ 40 \cdot x_{361,10TR} &\leq 70 + M \cdot slack_{361_1} \end{aligned}$$

No room double booking constraint:

$$\begin{aligned} x_{0,9MWF,101} + x_{1,9MWF,101} + x_{2,9MWF,101} + x_{3,9MWF,101} &\leq 1 \\ x_{0,10MWF,101} + x_{1,10MWF,101} + x_{2,10MWF,101} + x_{3,10MWF,101} &\leq 1 \\ x_{0,9TR,102} + x_{1,9TR,102} + x_{2,9TR,102} + x_{3,9TR,102} &\leq 1 \\ x_{0,10TR,102} + x_{1,10TR,102} + x_{2,10TR,102} + x_{3,10TR,102} &\leq 1 \end{aligned}$$

No professor double booking constraint:

$$\begin{aligned} x_{0,9MWF,161_1} &\leq 1 \\ x_{1,9MWF,161_1} &\leq 1 \\ x_{2,9MWF,161_1} + x_{3,9MWF,161_1} &\leq 1 \\ &\vdots \\ x_{0,10TR,361_1} &\leq 1 \\ x_{1,10TR,361_1} &\leq 1 \\ x_{2,10TR,361_1} + x_{3,10TR,361_1} &\leq 1 \end{aligned}$$

300/400/500+ courses at different time constraint: Since in the example there are no 400 and 500+ level courses the constraint will be constructed only for the 300 level courses.

$$\begin{aligned} \frac{x_{1,9MWF,351}}{1} + \frac{x_{3,9MWF,361}}{1} &\leq 1 \\ \frac{x_{1,10MWF,351}}{1} + \frac{x_{3,10MWF,361}}{1} &\leq 1 \\ \frac{x_{1,9TR,351}}{1} + \frac{x_{3,9TR,361}}{1} &\leq 1 \\ \frac{x_{1,10TR,351}}{1} + \frac{x_{3,10TR,361}}{1} &\leq 1 \end{aligned}$$

The denominator in the fraction is 1 since MATH 361 and MATH 351 each has one section.

Objective function:

$$\begin{aligned} \max \zeta = x_{0,0} \cdot 0.2 + x_{0,1} \cdot 0.4 + x_{0,2} \cdot 0.3 + x_{0,3} \cdot 0.1 + \\ + x_{1,0} \cdot 0.4 + x_{1,1} \cdot 0.3 + x_{1,2} \cdot 0.1 + x_{1,3} \cdot 0.2 + \\ + x_{2,0} \cdot 0.2 + x_{2,1} \cdot 0.3 + x_{2,2} \cdot 0.1 + x_{2,3} \cdot 0.4 + \\ + x_{3,0} \cdot 0.2 + x_{3,1} \cdot 0.3 + x_{3,2} \cdot 0.1 + x_{3,3} \cdot 0.4 \end{aligned}$$

4 Numerical Experiments

Through numerical experiments we check if the assigning professors to the courses part is working properly and check different variations of the input data for preference to teach several sections of same course and make sure that the constraint is working properly. Furthermore, we check which weighting type works better for the program to assign all the professors for their top preferences without having outliers that are assigned for their least preferred time slots or courses.

First of all to check if the constraint for professors preference for teaching several courses always works we first run the code with initial data and then randomly select 50% and then 100% percent of the professors as "do not want" for the question about preference of teaching several sections of same course. For the original data with only 1 professor chosen to not teach several sections of same course we have the following output.

```
Number of total unassigned section for professors = 0
Unassigned Professors: []
Unassigned professors section amount:  []


Number of total unassigned section for courses = 0
Unfilled Courses: []
Unassigned Courses section amount:  []
```

Figure 2: Output with original data

As it could be noticed from the output of the program in Figure2 there are no unassigned professor or courses, therefore with the original data received from the survey it works properly.

From the Figure 3 it is seem that after modifying data by randomly assigning 50% of the professors as 'do not prefer' that model is robust and maximizes number of assigned professors and course sections. However in order to check if the constraints are satisfied we should check the output to verify that no professor is teaching more than one section of each course. Figure 4 shows list of professors with chosen 'not prefer' and Figure 5 demonstrates that every professor is assigned for only one section of each course except Prof 4, the reason for that is that the input professor file has several professors pre-assigned for some courses and Prof 4 was assigned for sections 2 and 4 of MATH 162 course.

```
Number of total unassigned section for professors = 0
Unassigned Professors: []
Unassigned professors section amount:  []


Number of total unassigned section for courses = 0
Unfilled Courses: []
Unassigned Courses section amount:  []
```

Figure 3: Output with modified data(50%).

	Professor ID	Q4
1	Prof 2	-1
3	Prof 4	-1
6	Prof 7	-1
7	Prof 8	-1
9	Prof 10	-1
10	Prof 11	-1
11	Prof 12	-1
12	Prof 13	-1
13	Prof 14	-1
15	Prof 16	-1
23	Prof 24	-1
24	Prof 25	-1

Figure 4: Professors list (50%)

	Professor_ID	Course	Section	Capacity	Ranking
5	Prof 4	MATH 162	2L	200	1
6	Prof 4	MATH 162	4L	200	1
9	Prof 2	MATH 274	3L	50	1
10	Prof 2	MATH 350	1L	65	5
16	Prof 7	MATH 161	5L	70	1
17	Prof 7	MATH 263	1L	40	2
18	Prof 8	MATH 162	1L	200	1
19	Prof 8	MATH 251	2L	30	5
20	Prof 8	MATH 273	2L	60	3
21	Prof 10	MATH 423	1L	50	1
22	Prof 10	MATH 551	1L	20	2
23	Prof 11	MATH 322	1L	40	1
24	Prof 11	MATH 351	1L	40	2
25	Prof 12	MATH 322	3L	40	3
26	Prof 12	MATH 441	1L	50	4
27	Prof 13	MATH 361	1L	30	4
28	Prof 13	MATH 462	1L	50	3
29	Prof 14	MATH 310	1L	60	3
30	Prof 14	MATH 322	2L	40	4
35	Prof 16	MATH 302	1L	30	3
36	Prof 16	MATH 407	1L	50	2
52	Prof 24	MATH 161	1L	50	2
53	Prof 24	MATH 321	4L	90	4
54	Prof 24	MATH 411	1L	50	5
55	Prof 25	MATH 361	2L	30	5
56	Prof 25	MATH 403	1L	50	1

Figure 5: Professor-course assigning (50%)

After assigning all the professors as do not prefer, the number of unassigned sections and professors increased to 2 as it could be seen from the Figure 6. As a result after checking program for different type of input data the model is satisfies

all constraints and gives partial solution minimizing the not assigned professors and course sections.

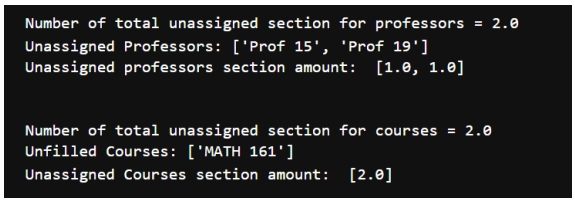

Figure 6: Output with modified data (100%).

5 Future Work

While this project successfully optimized class scheduling task, there are still several areas that should be explored. The current implementation focuses on constructing integer programming problem with linear constraints, however some constraints such as preference for consecutive time slots could be added by enabling absolute value which will make problem non-linear.

Consecutive time slot preference. In order to keep linearity of the problem in this project the constraint about preference of consecutive time slots for professors is not added. However if the main goal is not having linear problem, but having a program that will assign professors with preferences, this constraint could be added in many variations.

Recitation assigning. The problem application could be increased by assigning also the recitations along with the lectures, however in order to implement that all the conditions of assigning professors to the recitations must be taken into account such as if there is another load for each professor for recitations or if the 2 recitations weight same as 1 lecture teaching and etc.

Day preference constraint. An additional preference question could be added to the survey to account for professors' preferences to teach only on MWF (Monday, Wednesday, Friday) or TR (Tuesday, Thursday). The linear programming (LP) formulation should ensure that the total teaching assignments for each professor are either entirely on the preferred group of days or not at all. For example, if a professor prefers to teach on the TR group, the summation of their assignments on MWF should be zero, and the total load should be allocated to TR.

$$\sum_{t}^{t_{MWF}} \sum_{p}^{N_p} x_{p,t} = 0 \qquad \sum_{t}^{t_{TR}} \sum_{p}^{N_p} x_{p,t} = \text{load}_p$$

Not preferred time slot constraint. For further improvement considering time slots at which professors can not teach would be useful. Its implementation could be

as summation of professors allocations for the list of time slots must be equal to 0.

$$\sum_{p_{c_i}}^{N_p} \sum_{t_{r_j}}^{N_{t_r}} x_{p_{c_i}, t_{r_j}} = 0$$

Where t_{r_j} iterates in the list containing all available rooms for time slots which were chosen by professor p as can not teach and p_{c_i} iterates over all courses at which professor p was assigned. Even if the programs main goal is to assign each professor to their most preferred time slot there could be some cases when professor would be assigned for least preferred time, however by adding this constraint program would take into account time when professors for any reason can not teach at some times for any reason.

Optimal class assigning. An important direction for future research is to incorporate room size constraints more effectively into the course scheduling model, ensuring that small courses are assigned to small rooms and large courses to larger ones. This refinement is critical for optimizing the utilization of physical space within university, which could face limitations in classroom availability with increased amount of the courses. By explicitly modeling and enforcing room-capacity compatibility, future scheduling systems can increase number of assigned courses while satisfying time preferences of the professors.

Several output. In order to get several slightly different outputs from one program small randomly distributed epsilon coefficients could be added. By running same code each time due to small influence of the epsilon coefficients it is possible to get different outputs, however since this values are not big enough to change the original weight distribution the output still will be same with slight changes.

$$\textbf{max} \quad z = (w_1 + \epsilon_1)x_1 + (w_2 + \epsilon_2)x_2 + (w_3 + \epsilon_3)x_3 + \dots$$

6 Conclusion

This project successfully formulated and implemented an integer programming approach for optimizing faculty-course-section assignments within an academic institution. By leveraging the PuLP library the proposed model efficiently handled the assignment of professors to courses while adhering to workload constraints, professor preferences, and scheduling limitations.

The experimental results demonstrated that the optimization model effectively maximized professor satisfaction while ensuring fair workload distribution. Furthermore, constraints such as avoidance of multiple sections per professor, time-slot preferences, and room capacity were successfully incorporated into the scheduling process. The robustness of the model was validated through multiple test cases, showing its ability to generate feasible and near-optimal solutions within a small computational time.

Despite these achievements, there are areas for further improvement. The inclusion of additional constraints, such as professors' preferences for consecutive time slots

and specific unavailability periods, will be done in further modification of converting initial linear IP problem to nonlinear IP problem.

Overall, this research highlights the potential of integer programming in solving complex scheduling problems in academic institutions. By automating the course assignment process, the proposed model can significantly reduce administrative workload, improve fairness in faculty workload distribution, and enhance overall scheduling efficiency. Future advancements in optimization techniques and computational efficiency will further expand the applicability of such models in broader scheduling and resource allocation domains.

References

[1] R. Kunwar and H. P. Sapkota, "Introduction to linear programming problems with some real-life applications," *European Journal of Mathematics and Statistics*, vol. 3, no. 2, pp. 21–27, 2022. DOI: 10.24018/ejmath.2022.3.2.108. [Online]. Available: https://doi.org/10.24018/ejmath.2022.3.2.108.

[2] S. Nesmachnow, "An overview of metaheuristics: Accurate and efficient methods for optimisation," *International Journal of Metaheuristics*, vol. 3, no. 4, p. 320, 2014. DOI: 10.1504/ijmheur.2014.068914. [Online]. Available: https://doi.org/10.1504/ijmheur.2014.068914.

[3] C.-O. Foundation, *PuLP: A linear programming toolkit for Python*, Retrieved from https://coin-or.github.io/pulp/. [Online]. Available: https://coin-or.github.io/pulp/.

[4] M. Fischetti and D. Salvagnin, "Strengthening mixed-integer programming formulations via bound tightening," *Discrete Optimization*, vol. 19, pp. 39–55, 2016. DOI: 10.1016/j.disopt.2016.01.002. [Online]. Available: https://doi.org/10.1016/j.disopt.2016.01.002.

[5] Ş. İ. Birbil, *03_Cut_Plan_Octave.html*, https://personal.eur.nl/birbil/ie606/03_Cut_Plan/03_Cut_Plan_Octave.html, Accessed: April 8, 2025.

[6] E. Amaldi, *ILP: Cutting Planes*, https://amaldi.faculty.polimi.it/SlidesFOR-19-20/ILP-cutting-planes-FOR-18.pdf, University Presentation, Accessed: April 9, 2025.

Appendix

Part 1: Variables, objective function, constraints code

```
# Defining decision variables
# x[i, j, k] - professor i teaches section k of course j
x = LpVariable.dicts("assigned", ((i, j, k) for i in professors_list
for j in course_list for k in range(course_dict[j])
if (i, j, k) not in assigned_slots) , cat="Binary")

# y[i, j] - professor i teaches multiple sections of course j
```

```
y = LpVariable.dicts("multiple_sections", ((i, j)
for i in professors_list for j in course_list))

# Defining relaxation variables
p = LpVariable.dicts("prof_relax", professors_list, lowBound=0)
s = LpVariable.dicts("section_relax", course_list, lowBound=0)
r = LpVariable.dicts("preferred_relax", professors_list, lowBound=0)
q = LpVariable.dicts("multi_section_relax",
    ((i, j) for i in professors_list
    for j in course_list
    if two_section_preference.loc[i, two_section_preference.columns[0]] == -1),
    lowBound=0)

#Helper functions
# for a professor
def get_total_assignments(prof):
    pre = prof_preassigned_count[prof]
    new = lpSum(x[prof,j,k] for j in course_list
    for k in range(course_dict[j]) if (prof,j,k) not in assigned_slots)
    return pre + new

# for a course
def get_course_assignments(course):
    pre = course_preassigned_count[course]
    new = lpSum(x[i,course,k] for i in professors_list
        for k in range(course_dict[course])
        if (i,course,k) not in assigned_slots)
    return pre + new

def get_prof_course_assignments(prof, course):
    pre = len(pre_assignments.get(prof, {}).get(course, []))
    new = lpSum(x[prof,course,k]
        for k in range(load_dict[course])
        if (prof,course,k) not in assigned_slots)
    return pre + new

# sum of not preferred = 0
def get_non_preferred_assignments(prof):
    new = lpSum(x[prof,j,k]
                for j in course_list
                if j not in preferred_courses_for_professors[prof]
                for k in range(course_dict[j])
                if (prof,j,k) not in assigned_slots)
    return new

# sum of preferred section = professors load
```

```
def get_preferred_assignments(prof):
    pre = sum(len(sections)
              for course, sections in assigned_dict.get(prof, {}).items()
              if course in preferred_courses_for_professors[prof])

    # Count new preferred sections
    new = lpSum(x[prof,j,k]
                for j in preferred_courses_for_professors[prof]
                for k in range(course_dict[j])
                if (prof,j,k) not in assigned_slots)
    return pre + new

prob += (
    # Original preference maximization with modified preferences
    lpSum(modified_preferences.loc[i,j]*x[i,j,k]
            for i in professors_list
            for j in course_list
            for k in range(course_dict[j])
            if (i, j, k) not in assigned_slots)
            # Original relaxation penalties
            -100*lpSum(p[i] for i in professors_list)
            -100*lpSum(s[j] for j in course_list)
            -100*lpSum(r[i] for i in professors_list)
            # New multi-section relaxation penalty
            -10*lpSum(q[i,j]
                        for i in professors_list
                        for j in course_list
                        if two_section_preference.loc[i,
                        two_section_preference.columns[0]] == -1)
)

# Each section taught only by 1 professor
for j in course_list:
    for k in range(course_dict[j]):
        if not any((p, j, k) in assigned_slots for p in professors_list):
            prob += lpSum(x.get((i, j, k), 0) for i in professors_list
                        if (i, j, k) not in assigned_slots) <= 1

# Each professor teaches their load sections with relaxation
for i in professors_list:
    prob += get_total_assignments(i) + p[i] == load_dict[i]

# Each course must have all sections assigned with relaxation
for j in course_list:
    prob += get_course_assignments(j) + s[j] == course_dict[j]
```

```
# Professor must teach course which is in his preference
for i in professors_list:
    prob += get_preferred_assignments(i) + r[i] == load_dict[i]

# Professor cannot teach course which is not in his preference
for i in professors_list:
        prob += get_non_preferred_assignments(i) == 0

#two_section_preference
for i in professors_list:
    if two_section_preference.loc[i, two_section_preference.columns[0]] == -1:
        for j in course_list:
            # Sum of all sections for each course should be <= 1 (plus relaxation)
            prob += (lpSum(x.get((i,j,k), 0)
                        for k in range(course_dict[j])
                        if (i,j,k) not in assigned_slots)
                   + q[i,j] <= 1)
```

Part 2: Variables, objective function, constraints code

```
prob = LpProblem("Course_Scheduling", LpMaximize)

# Create sets for indices
courses = prof_courses_df.index.tolist()
timeslots = timeslots_df.index.tolist()
professors = prof_courses_df['professor'].unique().tolist()

time_groups = timeslots_df.groupby('timeslot', observed=True).apply(
    lambda x: x.index.tolist()).to_dict()

# Get course information
course_levels = {c: get_course_level(prof_courses_df.loc[c, 'course'])
                 for c in courses}
base_courses = {c: prof_courses_df.loc[c, 'course']
                for c in courses}
course_sections = {c: prof_courses_df.loc[c, 'section']
                   for c in courses}

# Create full course identifiers (course + section)
full_course_ids = {c: f"{base_courses[c]}-{course_sections[c]}"
                   for c in courses}

# Group courses by level
courses_300 = [c for c in courses if course_levels[c] == 300]
courses_400 = [c for c in courses if course_levels[c] == 400]
courses_500 = [c for c in courses if course_levels[c] >= 500]
```

```
# Decision variables
x = LpVariable.dicts("schedule",
                     ((c, t) for c in courses for t in timeslots),
                     cat='Binary')

# Slack variables for capacity constraints
slack = LpVariable.dicts("slack",
                         (c for c in courses),
                         lowBound=0,
                         cat='Binary')

# Objective function
objective = 0

# Add professor preferences to objective
for c in courses:
    professor = prof_courses_df.loc[c, 'professor']
    for t in timeslots:
        timeslot = timeslots_df.loc[t, 'timeslot']
        preference_weight = prof_preferences_df.loc[professor, timeslot]
        objective += preference_weight * x[c,t]


# Add course scheduling and slack terms
objective += lpSum(x[c,t] for c in courses for t in timeslots) -
            100 * lpSum(slack[c] for c in courses)

prob += objective

# 1. Each course section must be scheduled exactly once or use slack
for c in courses:
    prob += lpSum(x[c,t] for t in timeslots) + slack[c] == 1

# 2. Room capacity constraint with slack
for c in courses:
    for t in timeslots:
        room = timeslots_df.loc[t, 'room']
        room_capacity = timeslots_df.loc[t, 'room_capacity']
        course_capacity = prof_courses_df.loc[c, 'capacity']
        prob += course_capacity * x[c,t] <= room_capacity # + M * slack[c]

# 3. No room double-booking
for t in timeslots:
    for c in courses:
        prob += x[c, t] <= 1
```

```
# 4. No professor double-booking
for p in professors:
    prof_courses = prof_courses_df[prof_courses_df['professor'] == p].index
    for time_period, time_indices in time_groups.items():
        prob += lpSum(x[c,t] for c in prof_courses for t in time_indices) <= 1

# 5. Allow different sections of same 300-level course at the same time
for time_period, time_indices in time_groups.items():
    prob += lpSum(x[c,t] for c in courses_300 for t in time_indices) <= 1

# 6. Allow different sections of same 400-level course at the same time
for time_period, time_indices in time_groups.items():
    prob += lpSum(x[c,t] for c in courses_400 for t in time_indices) <= 1

# 7. Allow different sections of same 500-level course at the same time
for time_period, time_indices in time_groups.items():
    prob += lpSum(x[c,t] for c in courses_500 for t in time_indices) <= 1
```