An Application of Graph Optimization: Optimization of the University Course Assignment System

Ishaan Thakker, Parv Jain, Priyanshu Choubey

The research problem at hand revolves around the optimization of the University Course Assignment System. Within a department, there are ***N*** faculty members categorised into three distinct groups: ***X1***, ***X2*** and ***X3*** Faculty in each category are assigned different course loads, with ***X1*** handling 0.5 courses per semester, ***X2*** taking 1 course per semester, and ***X3*** managing 1.5 courses per semester.

In this system, faculty members have the flexibility to take multiple courses in a given semester, and conversely, a single course can be assigned to multiple faculty members. When a course is shared between two professors, each professor's load is considered to be 0.5 courses. Moreover, each faculty member maintains a preference list of courses, ordered by their personal preferences, with the most preferred courses appearing at the top. Importantly, there is no prioritisation among faculty members within the same category.

The primary objective of this research problem is to develop an assignment scheme that maximises the number of courses assigned to faculty while aligning with their preferences and the category-based constraints ***N*** faculty members categorised into three distinct groups: ***X1***,***X2***,***X3***. The challenge lies in ensuring that a course can only be assigned to a faculty member if it is present in their preference list.

This problem is unique due to the flexibility it offers regarding the number of courses faculty members can take, distinct from typical Assignment problems. Potential modifications may include adjusting the maximum number of courses "y" for each category of professors, instead of requiring exact adherence, or extending the number of professor categories beyond the existing three to devise a more generalised solution.

1. Objective

The objective is to assign professors ***X1***: Professors who want to take a course weight of **0.5.** ***X2***: Professors who want to take a course weight of **1**. ***X3***: Professors who want to take a course weight of **1.5**. The problem also states that the courses can be divided into 4 exhaustive sets, and the professor must also give his preference for the courses he wants to tech next semester

- **First Degree CDC's** (Atleast 4)
- **First Degree Electives** (Atleast 4)
- **Higher Degree CDC's** (Atleast 2)
- **Higher Degree Electives** (Atleast 2)

2. Constraints In Assignment

The problem states that the professors into 3 exhaustive sets ***X1***: Professors who want to take a course weight of **0.5.** ***X2***: Professors who want to take a course weight of **1**. ***X3***: Professors who

want to take a course weight of **1.5**. The problem also states that the courses can be divided into 4 exhaustive sets, and the professor must also give his preference for the courses he wants to tech next semester

- **First Degree CDC's** (Atleast 4)
- **First Degree Electives** (Atleast 4)
- **Higher Degree CDC's** (Atleast 2)
- **Higher Degree Electives** (Atleast 2)

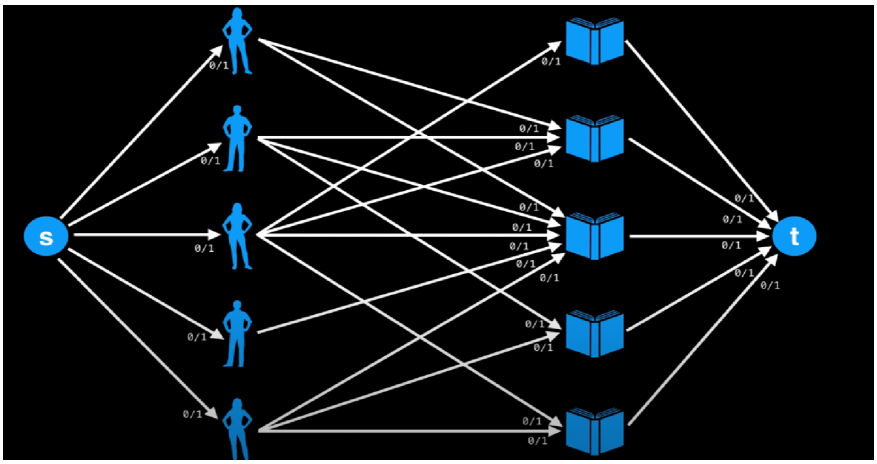
3. Methodologies Used

We tried to approach the problem using multiple methods, not all of them yielded the desired results. Here are the ones that we tried to implement

3.1. Maximum Matching-Maximum Flow

The initial solution considered by our team involved employing the maximum bipartite matching theorem, using unweighted bipartite matching to find the solution. To address the challenge of multiple course assignments to a single professor based on capacity, we assigned capacities to edges and utilized max-flow algorithms for the assignment. Refer to Figure 1 for a visual representation.

However, despite our implementation efforts, the outputs received only included the most optimal solution, displaying partial allotments instead of all optimal solutions. The algorithm failed due to its incapability to accommodate the constraint of complete course allotment to professors using max-flow algorithms.



Upon implementation, the outputs we received could only include the most optimal solution, with partial allotments and not all the optimal solutions, the reason for failure of the algorithm was obvious, the max-flow algorithms obviously couldn't accommodate the constraint of complete allotment of courses to professors.

4. Detailed Explanation of Hungarian Algorithm Implementation

The provided Python code implements the Hungarian Algorithm, a combinatorial optimization technique used to solve the Linear Assignment Problem efficiently. The goal of this algorithm is to find the optimal assignment of elements given a cost matrix. Below is an in-depth explanation of the implementation:

4.1. ***Usage***

The following file takes input from an .txt file, the inputs are the professor and the their course preference list, as the constraints mentioned each professor must take a minimum of 12 courses that consist of 4 FD CDCs 4 FD Electives 2 HD CDCs and 2 HD Electives, and creates a matrix for the same where the rows represent the matrix and the columns are the courses

4.2. Functions Overview

1. **min_zero_row**: This function identifies the row in the matrix that contains the fewest number of zeroes and marks the corresponding row and column as false. It ensures the algorithm's progress towards finding the optimal assignment.
2. **mark_matrix**: This function converts the input matrix into a boolean matrix, where a zero value is represented as `True`, and other values are `False`. It then identifies marked rows and columns containing zeroes and checks for additional marked columns in unmarked rows. This step is crucial for subsequent adjustments.
3. **adjust_matrix**: Upon identifying marked positions, this function adjusts the matrix by subtracting the minimum non-zero element among uncovered elements and updating the covered elements.
4. **hungarian_algorithm**: The core of the algorithm, this function iteratively adjusts the input matrix for row and column reductions. It enters an iterative process to find the optimal assignment, continuing until all rows and columns are covered by marked positions or an optimal assignment is achieved.
5. **ans_calculation**: After finding the optimal positions using the Hungarian Algorithm, this function calculates the total cost of the assignment by summing up the costs of the elements assigned according to the obtained positions.
6. **solve**: This function handles the assignment problem by constructing the cost matrix based on input data. It applies the Hungarian Algorithm to find the optimal assignment of courses to professors and retrieves the resulting solution.

4.3. ***Usage and Constraint***

The following file takes input from an .txt file, the inputs are the professor and the their course preference list, as the constraints mentioned each professor must take a minimum of 12 courses that consist of 4 FD CDCs 4 FD Electives 2 HD CDCs and 2 HD Electives, and creates a matrix for the same where the rows represent the matrix and the columns are the courses, it then performs row and column reduction to find the optimum solutions, and outputs the optimal solution in an output.txt file, the constraint arises when the number of professors and number of courses are not equal as this is an implementation of Hungarian Algorithm that only works for NxN matrices, if not then the program gives a crash report.

4.4. ***References***

Maximum Unweighted Bipartite Matching: https://www.youtube.com/watch?v=GhjwOiJ4SqUt=584s
https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/tutorial/
https://www.geeksforgeeks.org/maximum-bipartite-matching/
https://www.geeksforgeeks.org/hungarian-algorithm-assignment-problem-set-1-introduction/amp/