BIRLA INSTITUTE of TECHNOLOGY and SCIENCE, GOA
GOA - 403726

Project Report
on

"COURSE ALLOCATION USING ENUMERATING MAXIMUM MATCHINGS ALGORITHM"

Submitted in partial fulfilment of the requirements for the III Semester

Majoring
in
COMPUTER SCIENCE AND ENGINEERING
For the Academic Year
2023-2024
BY

cc

GROUP MEMBER A	Param Gandhi	2022A7PS0676G
GROUP MEMBER B	Soham Khadilkar	2022A7PS0277G
GROUP MEMBER C	Anish Shastry	2022A7PS0500G

UNDER THE GUIDANCE OF
Prof. Snehanshu Saha
DESIGNATION, Dept. of CSE, BITS Goa

Department of Computer Science and Engineering
BITS - GOA CAMPUS
Zuarinagar - 403726

ABSTRACT

Allocation of courses to professors is an endeavour common to all universities. Similar processes with the same underlying concept are applicable to other organizations as well. We attempt to automate this process in order to make it fast, efficient and trustworthy.

For this purpose, we use the Enumerating Maximum Matchings Algorithm (EMMA) and NetworkX library in python coding language. The technique we employ is that of One-One Maximum Unweighted Bipartite Matchings, by applying appropriate constraints to generate feasible allocations. We then assign Satisfaction Scores to these allocations based on the preference orders given by professors and display them in decreasing orders of Scores.

Thus, we present all possible allocations of courses to professors in descending order of preference.

Keywords: (EMMA), NetworkX, Unweighted Bipartite Matchings, Satisfaction Scores.

1 INTRODUCTION

1.1 Description

The following is a report on the project 'COURSE ALLOCATION USING ENUMERATING MAXIMUM MATCHING ALGORITHM' which attempts to optimize courses allocated to professors in C.S.E. department in BITS Goa.

1.2 Purpose

The purpose of this project is to maximize the number of courses allocated to professors, in accordance with the preference orders given by them.

1.3 Scope

Scope is wide due to the possibility of generalization, and use cases beyond University concerns.

2 IMPLEMENTATION

2.1 Overview

The method used for course optimization is Maximum Bipartite Matching. The code has been implemented in Python language, using the NetworkX library and Enumerating Maximum Matching Algorithm (EMMA). By applying necessary constraints to eliminate undesired matchings, we obtain and display the acceptable allocations, rank ordered by score which is based on preference order given by professors.

2.2 Detailed Explanation

The optimization problem at hand contains 3 categories of professors, accepting 0.5 (1 unit), 1 (2 units) and 1.5 (3 units) courses respectively. [Note: According to the problem statement, category 3 professors can accept either 1 or 1.5 course (either 2 or 3 units).] There are also 2 types of courses, CDCs and Electives, each having 2 units. This is shown in the following table:-

1. Professor	0.5 Course (Category 1)	1 Unit
	1.0 Course (Category 2)	2 Units
	1.5 Course (Category 3)	3 Units
2. Course	CDC	2 Units
	Elective	2 Units

The approach that we take to solve this problem is by finding bipartite graph matchings between 2 sets of nodes where one set of nodes represents professors while the other set represents courses. Since each professor can teach multiple courses, and each course can be taught by multiple professors, we have a many-many biparite matching problem. Tackling this problem in its present state is undesirably complex. Therefore, we convert this to a one-one matching problem by splitting individual professors and courses into nodes such that each unit represents a single node. This assignment of nodes is as follows:-

		cc	
1. Professor	0.5 Course(Category 1)	1 Unit	1 Node
	1.0 Course(Category 2)	2 Units	2 Nodes
	1.5 Course(Category 3)	3 Units	3 Nodes
2. Course	CDC	2 Units	2 Nodes
	Elective	2 Units	2 Nodes

Now that we have converted our problem into a one-one bipartite matching problem format, we proceed in the following fashion:

1. Generating all possible one-one maximum bipartite matchings using Enumerating Maximum Matchings Algorithm (EMMA).

   A Maximum Bipartite Matching is a matching in a bipartite graph that contains the maximum number of edges and where no more edges can be added.
   EMMA is an algorithm in the paper "Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs" by Takeaki Uno. According to Theorem 2 in this paper, time complexity of the alogrithm is $O(mn^{1/2} + nN_m)$ and space complexity is $O(m)$, where N_m is the number of maximum matchings in G. We use a python implementation of this algorithm by Guangzhi XU which makes use of NetworkX and NumPy modules.

2. We weed out undesirable matchings that do not satisfy the following constraints:

   (a) All nodes of professors of categories 1 and 2 should be assigned an edge.

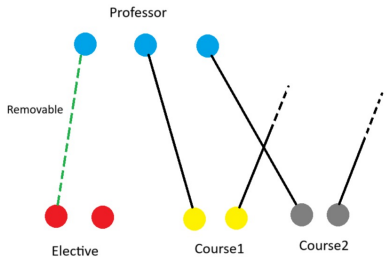
   (b) For all category 3 professors, 2 or 3 nodes should be assigned an edge.

   (c) Both nodes of all CDCs should be assigned an edge.

   (d) For electives, either both or no nodes should be assigned an edge.
   We need to check for certain eligible matchings which are not maximum but whose maximum version is eliminated due to this constraint. Such matchings occur because of some peculiar elective-professor edges. If an elective has only one unit assigned to a category 3 professor, whose

other 2 nodes are assigned to a single distinct course, or, separately assigned to two completely satisfied courses, then the edge joining the professor to the elective can be safely removed so as to incorporate the rest of the matching into the list of satisfactory matchings. We do this for all such electives to get the final desirable matching.

Figure 1: This figure demonstrates the edges to be removed.



3. Now that we have matchings that satisfy the aforementioned constraints, we convert them back into our original format of Professors and Courses.

   We do this to eliminate all the repetitions caused by intra-matching within the nodes of the same professor-course pairs.

4. We now assign Satisfaction Scores to each matching in accordance with the priority orders of Courses in the Professors' Preferences.

   For each edge in the matching we add $Max\_Points - k$ to get the Satisfaction Score of that matching, where k is the priority index of the course in the professor's preference order for respective professor-course pairs. Max_Points is the total number of courses. We add this term to ensure that the edge score always remains positive. Rank-ordering of desired matchings is performed based on the descending order of Satisfaction Scores. Matchings with higher Satisfaction Scores are preferred.

Thus, we present a list of all possible allocations of courses to professors, satisfying all constraints and maximizing preferences, in decreasing order of desirability.

Result Analysis

3 Test Results

3.1 Test Case 1

1. Courses

FDCDCs	FDElecs	HDCDCs	HDElecs
Course0	Course2	Course4	Course6
Course1	Course3	Course5	Course7

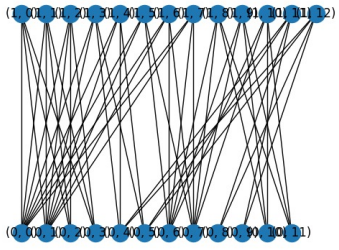
2. Professors

Category1	Category2	Category3
Prof0	Prof1	Prof3
	Prof2	Prof4
	Prof5	

3. Preferences

Prof0	Prof1	Prof2	Prof3	Prof4	Prof5
Course0	Course0	Course0	Course0	Course4	Course2
Course1	Course1	Course2	Course4	Course6	Course5

Figure 2: Bipartite Matching: Upper nodes are units of Professors while lower nodes are units of Courses.



4. Output

   Number of satisfactory matchings: 4

   63 : (('Prof0', 'Course1', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course1', 1), ('Prof2', 'Course2', 2), ('Prof3', 'Course0', 1), ('Prof3', 'Course4', 1), ('Prof4', 'Course4', 1), ('Prof4', 'Course6', 2), ('Prof5', 'Course5', 2))

   63 : (('Prof0', 'Course0', 1), ('Prof1', 'Course1', 2), ('Prof2', 'Course2', 2), ('Prof3', 'Course0', 1), ('Prof3', 'Course4', 1), ('Prof4', 'Course4', 1), ('Prof4', 'Course6', 2), ('Prof5', 'Course5', 2))

   62 : (('Prof0', 'Course1', 1), ('Prof1', 'Course1', 1), ('Prof1', 'Course0', 1), ('Prof2', 'Course2', 2), ('Prof3', 'Course4', 2), ('Prof3', 'Course0', 1), ('Prof4', 'Course6', 2), ('Prof5', 'Course5', 2))

   62 : (('Prof0', 'Course0', 1), ('Prof1', 'Course1', 2), ('Prof2', 'Course2', 2), ('Prof3', 'Course4', 2), ('Prof3', 'Course0', 1), ('Prof4', 'Course6', 2), ('Prof5', 'Course5', 2))

3.2 Test Case 2

1. Courses

FDCDCs	FDElecs	HDCDCs	HDElecs
Course0	Course2	Course4	Course6
Course1			

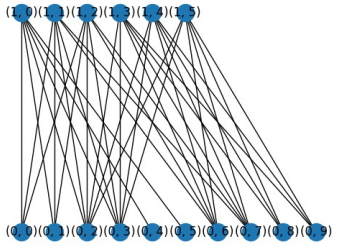
2. Professors

Category1	Category2	Category3
Prof0	Prof1	Prof3

3. Preferences

Prof0	Prof1	Prof3
Course0	Course0	Course1
Course1	Course1	Course4
Course2	Course4	Course6

Figure 3: Bipartite Matching: Upper nodes are units of Professors while lower nodes are units of Courses.



4. Output

Number of satisfactory matchings: 3

27 : (('Prof0', 'Course1', 1), ('Prof1', 'Course0', 2), ('Prof3', 'Course4', 2), ('Prof3', 'Course1', 1))

27 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course4', 1), ('Prof3', 'Course1', 2), ('Prof3', 'Course4', 1))

27 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course1', 1), ('Prof3', 'Course1', 1), ('Prof3', 'Course4', 2))

4 Crash Test/Consistency Report

1. Courses

FDCDCs	FDElecs	HDCDCs	HDElecs
Course0	Course2	Course4	Course3
Course1	Course5		Course6

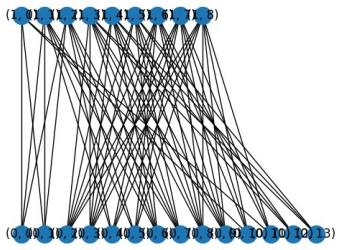
2. Professors

Category1	Category2	Category3
Prof0	Prof1	Prof2
		Prof3

3. Preferences

Prof0	Prof1	Prof2	Prof3
Course0	Course0	Course1	Course1
Course5	Course2	Course4	Course2
	Course3	Course3	Course4
		Course6	Course5

Figure 4: Bipartite Matching: Upper nodes are units of Professors while lower nodes are units of Courses.



4. Output

Number of satisfactory matchings: 15

52 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course2', 1), ('Prof2', 'Course4', 2), ('Prof2', 'Course1', 1), ('Prof3', 'Course2', 1), ('Prof3', 'Course1', 1))

52 : (('Prof0', 'Course0', 1), ('Prof1', 'Course2', 1), ('Prof1', 'Course0', 1), ('Prof2', 'Course4', 2), ('Prof3', 'Course2', 1), ('Prof3', 'Course1', 2))

51 : (('Prof0', 'Course0', 1), ('Prof1', 'Course2', 1), ('Prof1', 'Course0', 1), ('Prof2', 'Course1', 2), ('Prof2', 'Course4', 1), ('Prof3', 'Course2', 1), ('Prof3', 'Course4', 1))

51 : (('Prof0', 'Course0', 1), ('Prof1', 'Course2', 1), ('Prof1', 'Course0', 1), ('Prof2', 'Course4', 1), ('Prof2', 'Course1', 1), ('Prof3', 'Course2', 1), ('Prof3', 'Course4', 1), ('Prof3', 'Course1', 1))

50 : (('Prof0', 'Course5', 1), ('Prof1', 'Course0', 2), ('Prof2', 'Course4', 2), ('Prof2', 'Course1', 1), ('Prof3', 'Course5', 1), ('Prof3', 'Course1', 1))

50 : (('Prof0', 'Course5', 1), ('Prof1', 'Course0', 2), ('Prof2', 'Course4', 2), ('Prof3', 'Course5', 1), ('Prof3', 'Course1', 2))

50 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course3', 1), ('Prof2', 'Course4', 2), ('Prof2', 'Course3', 1), ('Prof3', 'Course1', 2))

50 : (('Prof0', 'Course0', 1), ('Prof1', 'Course2', 1), ('Prof1', 'Course0', 1), ('Prof2', 'Course1', 2), ('Prof3', 'Course2', 1), ('Prof3', 'Course4', 2))

49 : (('Prof0', 'Course5', 1), ('Prof1', 'Course0', 2), ('Prof2', 'Course4', 1), ('Prof2', 'Course1', 2), ('Prof3', 'Course5', 1), ('Prof3', 'Course4', 1))

49 : (('Prof0', 'Course5', 1), ('Prof1', 'Course0', 2), ('Prof2',

'Course4', 1), ('Prof2', 'Course1', 1), ('Prof3', 'Course5', 1), ('Prof3', 'Course4', 1), ('Prof3', 'Course1', 1))

49 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course3', 1), ('Prof2', 'Course4', 1), ('Prof2', 'Course1', 1), ('Prof2', 'Course3', 1), ('Prof3', 'Course4', 1), ('Prof3', 'Course1', 1))

49 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course3', 1), ('Prof2', 'Course4', 1), ('Prof2', 'Course3', 1), ('Prof3', 'Course1', 2), ('Prof3', 'Course4', 1))

48 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course3', 1), ('Prof2', 'Course1', 2), ('Prof2', 'Course3', 1), ('Prof3', 'Course4', 2))

48 : (('Prof0', 'Course0', 1), ('Prof1', 'Course0', 1), ('Prof1', 'Course3', 1), ('Prof2', 'Course1', 1), ('Prof2', 'Course3', 1), ('Prof3', 'Course4', 2), ('Prof3', 'Course1', 1))

48 : (('Prof0', 'Course5', 1), ('Prof1', 'Course0', 2), ('Prof2', 'Course1', 2), ('Prof3', 'Course5', 1), ('Prof3', 'Course4', 2))

Time: 1690.72 seconds (28.18 mins)

The code is uniform in writing and consistent in its output. It may take time to run as time complexity is high, but will give all possible desirable maximum matchings without crashing. For example, the test case above took 28.18 minutes, but successfully gave us an output.

If a CDC is not a part of the preference order of any professor, the code lets us know without crashing. If there are no satisfactory matchings, then it still does not crash and informs us of the same.

5 Conclusion

Thus, the code for course allocation optimization has been implemented in python.

6 Future Scope

The concept behind course optimization has many use cases and shows great potential with scope for improvement. This includes :-

- Optimizing the code to increase efficiency and generalize it to larger data inputs.
- Exploring other algorithms like Hungarian algorithm for weighted bipartite graphs.
- Creating a proper application with well defined GUI and easy accessibility.
- Implementing CUDA so as to run the program on NVIDIA GPU for higher efficiency and distributing workloads over parellelized GPUs.

References

[1] *Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs*; Takeaki Uno

[2] https://github.com/Xunius/bipartite_matching/