

**Birla Institute of Technology and Science,
GOA - 403726**



**Project Report
on
“AN APPLICATION OF GRAPH OPTIMIZATION”**

Submitted in partial fulfilment of the requirements for the III Semester

**Bachelor of Engineering
in
COMPUTER SCIENCE AND ENGINEERING
For the Academic Year
2023-2024
BY**

SAIKRISHNA	2021B5A71645G
PRATHIK SHETTY	2021B3A71317G
ADVAY BURTE	2021B5A72873G

**UNDER THE GUIDANCE OF
Snehanshu Saha
Department of CSE, BITS Pilani**



**Department of Computer Science and Engineering
BITS Pilani - K. K. Birla Goa CAMPUS
Zuarinagar, Sancoale, Goa 403726**

Birla Institute of Technology and Science
GOA - 403726



Project Report
on
“AN APPLICATION OF GRAPH OPTIMIZATION”

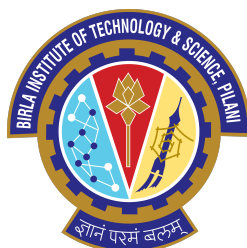
Submitted in partial fulfilment of the requirements for the III Semester

Bachelor of Engineering
in
COMPUTER SCIENCE AND ENGINEERING
For the Academic Year
2023-2024

BY

SAIKRISHNA	2021B5A71645G
PRATHIK SHETTY	2021B3A71317G
ADVAY BURTE	2021B5A72873G

UNDER THE GUIDANCE OF
Snehanshu Saha
Department of CSE, BITS Pilani



Department of Computer Science and Engineering
BITS Pilani - K. K. Birla Goa CAMPUS
Zuarinagar, Sancoale, Goa 403726

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to Birla Institute of Technology and Science, Goa, for providing the support that facilitated the successful completion of this project. The conducive academic environment at the institution played a pivotal role in the realization of this research endeavor.

We extend my heartfelt thanks to our mentor, Professor Snehanshu Saha, for his invaluable guidance and his insightful inputs throughout the duration of the project. We are grateful for the opportunity to learn under his mentorship.

ABSTRACT

This report explores the application of graph optimization techniques in the context of assigning courses to professors based on their preferences. The objective is to find optimal or suboptimal solutions that maximize overall happiness. Three different approaches are investigated to address this assignment problem. The first approach employs a brute force method, wherein the total happiness of each professor is calculated for every possible assignment, and combinations with the highest happiness are selected. The second approach formulates the problem as a binary integer program, and maximizes the objective function while adhering to given constraints. The third approach employs a greedy algorithm that assigns courses to professors based on their preference order.

Contents

1	Introduction	2
2	Problem Formulation	3
2.1	Method 1: Brute Force Approach	3
2.1.1	Objective:	3
2.1.2	Approach	3
2.2	Method 2: Binary Integer Linear Programming Model	4
2.2.1	Objective	4
2.2.2	Approach:	4
2.3	Method 3: Greedy Algorithm	6
2.3.1	Objective	6
2.3.2	Approach:	6
2.4	Dataset Generation for Testing	7
3	Results	8
3.1	Brute Force Method	8
3.2	Binary Integer Linear Programming	8
3.3	Greedy Algorithm	9
4	Crash Test	11
4.1	Brute Force Method	11
4.2	Binary Integer Linear Programming Model	12
4.3	Greedy Algorithm	13

Chapter 1

Introduction

This report explores the application of graph optimization techniques in the context of assigning courses to professors based on their preferences. Four distinct course types, namely "FDCDC," "HDCDC," "FDELE," and "HDELE," are considered, with professors categorized into three groups capable of handling 0.5, 1, or 1.5 courses per semester. The objective is to find optimal or suboptimal solutions that maximize overall happiness.

Three different approaches are investigated to address this assignment problem. The first approach employs a brute force method, wherein the total happiness of each professor is calculated for every possible assignment, and combinations with the highest happiness are selected. However, this method exhibits poor time complexity and becomes impractical for a large number of courses and professors.

The second approach formulates the problem as a binary integer program, utilizing the linear program solver of pulp library in python to maximize the objective function while adhering to given constraints. Despite providing an optimal solution, this method lacks the flexibility to yield multiple solutions, limiting its applicability.

The third approach employs a greedy algorithm that assigns courses to professors based on their preference order while ensuring all "CDC" courses are assigned. By randomly shuffling the order of professors, multiple possible suboptimal assignments are generated. This approach offers a balance between efficiency and the exploration of diverse solutions.

Through a comprehensive analysis, this report compares the strengths and limitations of each approach, considering factors such as time complexity, solution optimality, and flexibility in generating multiple solutions. The findings aim to guide decision-makers in selecting an appropriate method based on the specific requirements of their course assignment problem.

Chapter 2

Problem Formulation

2.1 Method 1: Brute Force Approach

2.1.1 Objective:

The primary aim of Method 1 was to explore a brute force strategy to solve the assignment problem. The problem involved assigning professors to courses based on their preferences, with the goal of maximizing overall happiness. This method is implemented using the `abc.py`, `def.py` and `fgh.py` files.

2.1.2 Approach

Data Preparation:

Input data was provided in a JSON format containing a dictionary of students, professors, and their preference orders. This JSON file was created through `abc.py`, wherein `— write method`. Then, in `def.py`, the subjects and professors were extracted from the input, creating separate lists for each. To accommodate professors' varying course loads (0.5, 1, or 1.5 courses), the subjects list was duplicated accordingly. The prof list was adjusted by making copies of profs according to their course loads.

Assignment Matrix:

A matrix was constructed, where rows represented all possible permutations of professors, and columns represented all possible permutations of subjects. Each professor was then assigned to a corresponding subject in the matrix.

Happiness Function:

A happiness function was defined to calculate the satisfaction of each professor based on their assigned subject. Maximum happiness was achieved when a professor was assigned their first preference, with a decrease in happiness for subsequent preferences.

Happiness Function:

A happiness function was defined to calculate the satisfaction of each professor based on their assigned subject. Maximum happiness was achieved when a professor was assigned their first preference, with a decrease in happiness for subsequent preferences.

Optimization:

The overall happiness for each permutation was computed by summing the individual happiness values. The program filtered out professor-course assignments yielding the maximum overall happiness.

2.2 Method 2: Binary Integer Linear Programming Model**2.2.1 Objective**

In this section, we present our second approach to solving the professor-course assignment problem using Binary Integer Linear Programming (BILP). We formulate the problem as a binary integer linear programming model where the objective is to maximize the overall satisfaction given all possible course assignments. This method utilizes the PuLP library in Python, which facilitates the formulation and solution of linear programming models.

2.2.2 Approach:**Binary Integer Program Implementation**

A Binary Integer Programming (BIP) problem is similar to a standard linear programming problem, with the exception that its decision variables are restricted to binary values, either 0 or 1. This restriction allows us to model discrete decisions, such as whether or not to assign a particular course to a particular professor.

To effectively model the professor-course assignment problem, we divide courses into four sets: FDCDC courses, HDCDC courses, FDELE courses, and HDELE courses. Each faculty member provides preference rankings for each type of course. In our model, faculty members provide four preferences for FDCDC courses, two preferences for HDCDC courses, four preferences for FDELE courses, and four preferences for HDELE courses. This setup allows us to incorporate constraints that ensure that faculty members teach a specified number of units. The number of units taught by a faculty member is represented by a weight ranging from 1 to 3. A weight of 1 indicates that the faculty member teaches 0.5 course, a weight of 2 indicates that the faculty member teaches 1 course, and a weight of 3 indicates that

the faculty member teaches 1.5 courses. Any special conditions or requirements can be incorporated into the model as additional constraints.

Decision Variables

The first step in formulating the BILP model is to define the decision variables. In this problem, we use decision variables of the form x_{ij} , where i represents the professor and j represents the course. If x_{ij} takes the value 0, it implies that professor i has not been assigned course j ; if x_{ij} takes the value 1, it implies that professor i has been assigned course j .

Objective Function

We use the previously defined happiness function to assign the profs a happiness score along with their course assignment.

$$\text{Happiness score} = \sum x_{ij} * \text{happiness}(x_{ij})$$

Constraints

The constraints are as follows:

1. Binary Constraints (taken care by pulp solver)
 - For every $x_{ij} \geq 0$
 - For every $x_{ij} \leq 1$
2. No prof gets exactly his assigned credits for every professor i ,

$$\sum x_{ij} = \text{assigned} - \text{credits}(i)$$
3. Every CDC gets exactly 2 profs (can be the same prof doing both halves of the course).

So, for every CDC course j , $\sum x_{ij} = 2$
4. every Elective gets either 0 or 2 profs (can be the same prof doing both halves of the course)
 - For every elective course j , $\sum x_{ij} \neq 1$
 - For every elective course j , $\sum x_{ij} \leq 2$

Note: Intermediate Code

Additionally, we have a Binary Integer Linear Program Intermediate Code, which only takes the random data generated by the Random Input Intermediate Code. Serving as a transitional step towards the Binary Integer Linear Program, this intermediate code focuses on obtaining a singular optimized solution for datasets where

the number of professors equals the number of courses. This condition ensures the feasibility of a valid assignment.

2.3 Method 3: Greedy Algorithm

2.3.1 Objective

We now approach the problem implementing a Greedy algorithm. This will allow us to find multiple suboptimal solutions to the professor- course assignment problem while adhering to the constraints.

2.3.2 Approach:

Greedy Algorithms

Greedy algorithms are a commonly used paradigm for combinatorial algorithms. Combinatorial problems intuitively are those for which feasible solutions are subsets of a finite set (typically from items of input). Therefore, in principle, these problems can always be solved optimally in exponential time (say, $O(2^n)$) by examining each of those feasible solutions. The goal of a greedy algorithm is find the optimal by searching only a tiny fraction.

Defining precisely what a greedy algorithm is hard, if not impossible. In an informal way, an algorithm follows the Greedy Design Principle if it makes a series of choices, and each choice is locally optimized; in other words, when viewed in isolation, that step is performed optimally.

Implementation

In our application of the greedy algorithm to the professor-course assignment problem, we adopt an iterative approach. Sequentially processing each professor, we assign their preferred course based on availability. If the first preference is available we assign it to them, if unavailable, subsequent preferences are considered until a viable assignment is made(Provided that such an assignment is possible)

By varying the order of professors, we generate multiple suboptimal solutions to the assignment problem.

Constraints

The following constraints are followed while making the assignments.

1. Every professor gets exactly their assigned credits
2. Every CDC course gets exactly 2 professors (can be the same professor doing both halves of the course)

3. Every elective course gets either 0 or 2 professors (can be the same professor doing both halves of the course) i.e., elective courses don't have to be assigned.

2.4 Dataset Generation for Testing

Our data generation process is seamlessly automated to provide a continuous influx of new datasets for testing our code, specifically for Method 2: Binary Integer Linear Programming and Method 3: Greedy Algorithm.

Initially, the code randomly generates x values representing the credits associated with each professor. Care is taken to ensure that the sum of all x values exceeds the total number of Core Discipline Courses (CDC) but remains less than the total number of courses. This guarantees the assignment of all CDCs while allowing flexibility in assigning elective courses, as our code prioritizes CDCs before electives. Consequently, elective courses may or may not be offered in a given semester.

The number of x values generated determines the corresponding number of professors to whom these credits are assigned. Subsequently, the code automatically generates preference orders for Full Duration CDCs (4 courses), Half Duration CDCs (2 courses), Full Duration Electives (4 courses), and Half Duration Electives (2 courses).

For the overall subjects, the data remains constant, assuming the same courses are assigned each semester. Our assumptions include three Full Duration CDCs and two Half Duration CDCs for the first year, four Full Duration CDCs, two Half Duration CDCs, six Full Duration Electives, and four Half Duration Electives for the second and third years, and six Full Duration Electives and four Half Duration Electives for the fourth year.

Note: In the case of Method 2, Binary Integer Linear Programming, we employ a distinct data set generation method tailored exclusively for the intermediate code. The key difference lies in generating the dataset such that the number of courses and professors are equal, ensuring a balanced assignment.

Chapter 3

Results

This section gives a brief overview of the results. As we used a randomized process to generate the various test cases, and ran our code on them, we can say with confidence that our results hold good for almost all test cases. The exceptions to this are discussed in the Crash Report section. Note: Detailed documentation for each of the codes is available on the GitHub repository, in the form of multiple README files.

3.1 Brute Force Method

This is not the primary aim of our report. It has been included to show our thought process. We started with the Brute Force method to get an idea of the size and complexity of the problem. While it served its purpose and gave us valuable insights, its impracticality for larger inputs due to exponential time complexity highlighted the need for more efficient algorithms. Subsequent methods, as discussed in the report, explored optimized approaches to address scalability issues.

3.2 Binary Integer Linear Programming

Our implementation using Binary Integer Linear Programming successfully optimizes the assignment of professors to courses, maximizing overall happiness. However, to address the requirement of obtaining multiple solutions, an alternative approach is necessary. The subsequent sections explore a third method to overcome this limitation and provide a comprehensive solution to the professor-course assignment problem. The form of the output can be seen below, in Figure 3.1.

The screenshot shows a code editor with the Explorer pane on the left. The Explorer pane shows a project structure with folders like FINAL, BinaryIntegerProgram, and files like Greedy, README.txt, and BruteForce. The main editor displays the output of a BILP program, showing a table with columns Professor, Subject, and Happiness. The table lists assignments for various professors and courses, such as Prof0, Prof1, Prof2, Prof3, Prof4, Prof5, Prof6, Prof7, Prof8, Prof9, and their corresponding happiness values.

Professor	Subject	Happiness
Prof0	HDCDC1-Y3-S1	94
Prof1	FDCDC3-Y1-S1	93
Prof1	FDE2-Y4-S1	94
Prof1	FDE2-Y4-S1	94
Prof2	HDCDC1-Y3-S1	94
Prof3	FDE5-Y2-S1	93
Prof4	FDCDC2-Y3-S1	94
Prof4	FDE4-Y4-S1	94
Prof4	FDE4-Y4-S1	94
Prof5	FDE4-Y3-S1	94
Prof6	FDCDC3-Y1-S1	93
Prof7	FDE3-Y2-S1	91
Prof7	FDE3-Y2-S1	91
Prof7	FDE5-Y3-S1	94
Prof8	FDCDC2-Y2-S1	94
Prof8	FDCDC4-Y3-S1	93
Prof8	HDCDC1-Y2-S1	94
Prof9	FDCDC1-Y1-S1	94
Prof9	FDCDC1-Y1-S1	94
Prof9	FDE4-Y2-S1	94

Figure 3.1: Example output for BILP code

3.3 Greedy Algorithm

In our application of the greedy algorithm to the professor-course assignment problem, we adopt an iterative approach. Sequentially processing each professor, we assign their preferred course based on availability. If the first preference is available we assign it to them, if unavailable, subsequent preferences are considered until a viable assignment is made (Provided that such an assignment is possible). To obtain a multiple solutions, our code executes this process approximately 10,000 times, repeatedly shuffling the order of professors and filtering out feasible solutions. The outcome comprises 50-80 distinct solutions for various professor-course assignments, acknowledging the potential recurrence of some solutions. The form of the output can be seen below in Figure 4.2. The output folder is opened in the Explorer on the left, and one of the output files is opened as an example.

Greedy > GreedyOutputs > Output3

1	+-----+-----+	
2	Professor	Subject
3	+-----+-----+	
4	Prof0	FDCDC3-Y2-S1
5	+-----+-----+	
6	Prof1	FDCDC1-Y1-S1
7	+-----+-----+	
8	Prof2	FDE5-Y4-S1
9	+-----+-----+	
10	Prof3	FDCDC2-Y2-S1
11	+-----+-----+	
12	Prof3	HDCDC1-Y2-S1
13	+-----+-----+	
14	Prof3	FDE5-Y3-S1
15	+-----+-----+	
16	Prof4	FDCDC3-Y3-S1
17	+-----+-----+	
18	Prof4	HDCDC1-Y1-S1
19	+-----+-----+	
20	Prof5	FDCDC1-Y3-S1
21	+-----+-----+	
22	Prof6	FDE4-Y4-S1
23	+-----+-----+	
24	Prof7	FDCDC3-Y3-S1
25	+-----+-----+	
26	Prof7	FDE1-Y4-S1
27	+-----+-----+	
28	Prof8	FDE4-Y3-S1
29	+-----+-----+	
30	Prof9	FDE4-Y2-S1
31	+-----+-----+	
32	Prof9	FDE3-Y2-S1
33	+-----+-----+	
34	Prof10	HDCDC1-Y3-S1
35	+-----+-----+	
36	Prof10	FDE6-Y3-S1
37	+-----+-----+	
38	Prof11	FDCDC1-Y1-S1
39	+-----+-----+	
40	Prof11	HDCDC1-Y1-S1

Figure 3.2: Example output for BILP code

Chapter 4

Crash Test

4.1 Brute Force Method

The screenshot shows a code editor with two tabs: 'BruteForceInput.json' and 'BruteForce.py'. The JSON file contains a list of subjects and professors with their priority lists. The Python script, 'BruteForce.py', is running and has crashed with a memory error. The terminal output shows a traceback indicating a 'numpy.core._exceptions._ArrayMemoryError: Unable to allocate 181. TiB for an array with shape (7484400, 3326400) and data type float64'.

```

1  {
2    "Subjects" : ["subject1","subject2","subject3","subject4","subject5", "subject6"],
3    "Professors": {
4      "Prof1": {"X": 1, "Prioritylist": ["subject1","subject2", "subject3"]},
5      "Prof2": {"X": 2, "Prioritylist": ["subject3","subject2", "subject4"]},
6      "Prof3": {"X": 3, "Prioritylist": ["subject1","subject3", "subject5"]},
7      "Prof4": {"X": 1, "Prioritylist": ["subject4","subject5", "subject2"]},
8      "Prof5": {"X": 2, "Prioritylist": ["subject5","subject4", "subject1"]},
9      "Prof6": {"X": 3, "Prioritylist": ["subject1","subject3", "subject6"]}
10   }
11 }
12 }
13

```

```

Traceback (most recent call last):
  File "/Users/prathikshetty/Downloads/FINAL/BruteForce.py", line 66, in <module>
    obj1.max_happiness()
  File "/Users/prathikshetty/Downloads/FINAL/BruteForce.py", line 42, in max_happiness
    happiness = np.zeros((len(self.rows),len(self.columns)))
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 181. TiB for an array with shape (7484400, 3326400)
and data type float64

```

Figure 4.1: Brute Force Crash

When we try to run this code for more than 5 professors and 5 courses, it crashes as the number of permutations of professors and courses exceeds what the computer can store in its memory, as seen in Figure 4.1 above.

4.2 Binary Integer Linear Programming Model

```

BinaryIntegerProgramOutput.txt
1  CRASH! This Problem is not feasible and the program crashes. But assignment
2  has been done to the best of the program's ability
3  .
4  +-----+-----+-----+
5  | Professor | Subject   | Happiness |
6  +-----+-----+-----+
7  | Prof0     | FDCDC1-Y2-S1 | 94 |
8  +-----+-----+-----+
9  | Prof0     | FDCDC1-Y2-S1 | 94 |
10 +-----+-----+-----+
11 | Prof0     | HDCDC1-Y2-S1 | 94 |
12 +-----+-----+-----+
13 | Prof1     | FDCDC4-Y3-S1 | 94 |
14 +-----+-----+-----+
15 | Prof1     | HDCDC1-Y2-S1 | 93 |
16 +-----+-----+-----+
17 | Prof1     | HDCDC2-Y1-S1 | 94 |
18 +-----+-----+-----+
19 | Prof2     | FDCDC3-Y3-S1 | 94 |

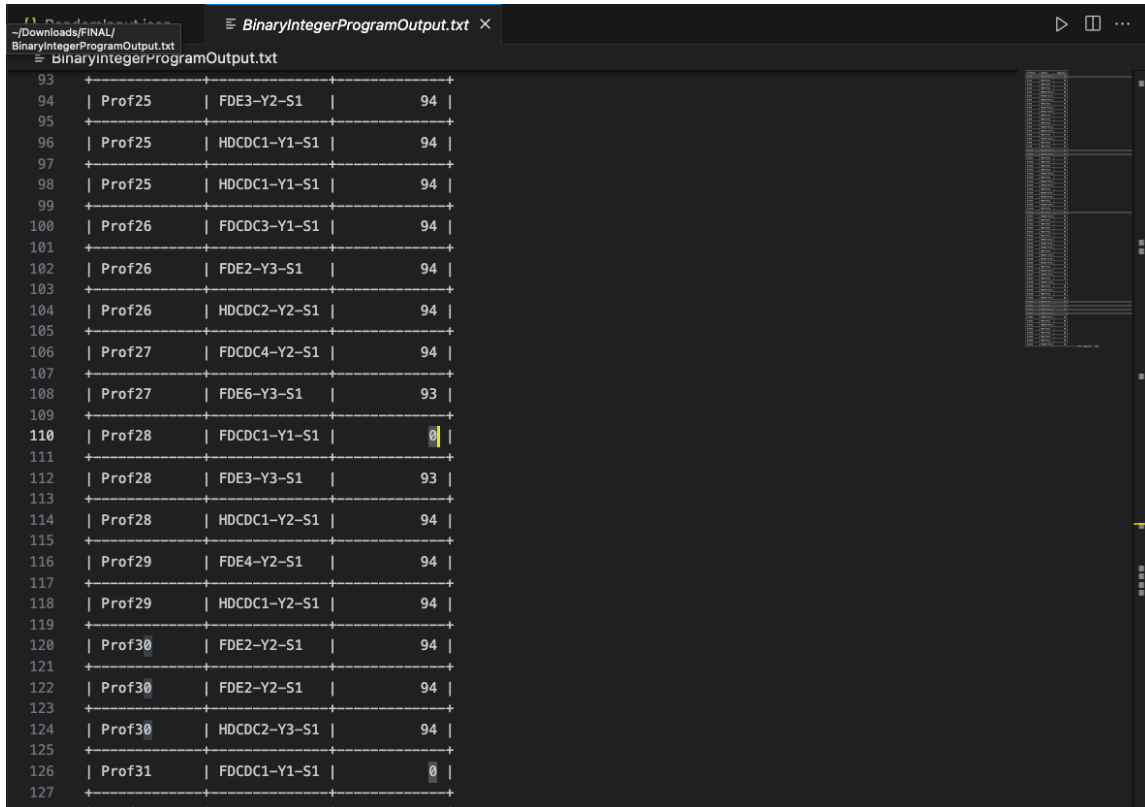
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Option for timeMode changed from cpu to elapsed
Problem is infeasible - 0.00 seconds
Option for printingOptions changed from normal to all
Total time (CPU seconds):      0.01  (Wallclock seconds):      0.03

-1

```

Figure 4.2: NCrash - Not enough professors to cover CDCs

For the BILP code, the program crashes in two cases. The first one is when there are not enough professors to cover the CDCs. We have designed the code so that it throws an error but still tries to assign courses as best it can. The output in this case can be seen in Figure 4.2.



93			
94	Prof25	FDE3-Y2-S1	94
95			
96	Prof25	HDCDC1-Y1-S1	94
97			
98	Prof25	HDCDC1-Y1-S1	94
99			
100	Prof26	FDCDC3-Y1-S1	94
101			
102	Prof26	FDE2-Y3-S1	94
103			
104	Prof26	HDCDC2-Y2-S1	94
105			
106	Prof27	FDCDC4-Y2-S1	94
107			
108	Prof27	FDE6-Y3-S1	93
109			
110	Prof28	FDCDC1-Y1-S1	0
111			
112	Prof28	FDE3-Y3-S1	93
113			
114	Prof28	HDCDC1-Y2-S1	94
115			
116	Prof29	FDE4-Y2-S1	94
117			
118	Prof29	HDCDC1-Y2-S1	94
119			
120	Prof30	FDE2-Y2-S1	94
121			
122	Prof30	FDE2-Y2-S1	94
123			
124	Prof30	HDCDC2-Y3-S1	94
125			
126	Prof31	FDCDC1-Y1-S1	0
127			
128			

Figure 4.3: Crash - One CDC no professor gives a preference for

The other case where the program crashes is when there is one CDC which doesn't fall on any professor's priority list. The program does an assignment as best it can, but in such a situation, we see a happiness score of 0 given to one of the professors, as seen in Figure 4.3 below.

4.3 Greedy Algorithm

The Greedy Algorithm also crashes in the case where the number of professors is not sufficient to cover the CDCs. The program returns a 0, signifying that no solutions have been found. The other output is the time taken. This can be seen in Figure 4.4 below.

```

123     finalchk = True
124     for i in subjects_remaining_list:
125         """TO ENSURE 0.5 SUBJECT DOESNT GET ASSIGNED AND ALL
126         if "CDC" in i:
127             finalchk = False
128             break
129         if (subjects_remaining_list.count(str(i))%2 == 1):
130             finalchk = False
131             break
132     if(finalchk):
133         if None not in prof_assign.values():
134             solnmatrix = []
135             for key, value in prof_assign.items():
136                 solnmatrix.append([str(key.split("_")[0]),value])
137             finalsoln = sorted(solnmatrix, key=lambda x: int(x[1]))
138             table = tabulate(finalsoln, headers=["Professor", "Subject"])

```

PROBLEMS OUTPUT TERMINAL ... Python + - [] [X] ^ X

```

prathikshetty@Prathiks-MacBook-Air FINAL % /usr/local/bin/python3 /Users/prathikshetty/Downloads/FINAL/Greedy.py
0
19.51658554142341

```

Figure 4.4: Crash - Not enough professors to cover CDCs

These are the test cases that we found that gave us errors or crashes. In other cases, the code consistently gives us good results. We can be confident of this as the way we generated test cases was randomized.