IE-400 PRINCIPLES OF PROJECT MANAGEMENT

PROJECT REPORT

Abdullah Emir Öztürk: 21601852

Ömer Faruk Oflaz: 21602649

Erkut Alakuş: 21101413

Models

IP Problem

Parameters:

x_j : The amount of time for the j[th] student to finish the homework.

t_{ij}: Distance (as time) from i[th] person to j[th] person. (i = 0,1…N j=0,1…N)

Variables:

$$Y_{ij} = \begin{cases} 1\ , if\ the\ assistant\ goes\ from\ ith\ person\ to\ jth\ person \\ 0, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad otherwise \end{cases}$$

Model:

Object:

$$min \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} Y_{ij}(t_{ij} + x_j)$$

Subject to:

$$\sum_{i=1}^{\infty} Y_{ij} = 1\ for\ all\ j$$

$$\sum_{j=1}^{\infty} Y_{ij} = 1\ for\ all\ i$$

$$\sum_{i \in S}^{\infty} \sum_{j \notin S}^{\infty} Y_{ij} \geq 1 \quad S \subseteq N, S \neq \emptyset$$

i = 0,1…N

j = 0,1…N

$Y_{ij} \in \{0,1\}$

DP Problem

Parameters:

x_j : The amount of time for the j[th] student to finish the homework.

t_{ij}: Distance (as time) from i[th] person to j[th] person. (i=0,1…N j=0,1…N)

S: Cluster that represents N students so S= {1,2,3,4, ….,N}

DP function:

$$G(i,S) = \begin{cases} \min\limits_{K \in S}\{(t_{ij} + x_j) + G(K, S - \{K\})\} & if\ S \neq \emptyset \\ G(K,1)\ if\ S = \emptyset \end{cases}$$

Creating Random Study Time and Distances for N Student

In this part, we have used MATLAB in order to create N*ln(N) random matrices and to take their average. Also, we determined a random vector "x" that represents random study time for N student.

Note that we created vector "x" has 76 value. The first one is fixed for the professor, so it is zero. The other ones are random in the interval [300, 500].

In addition, random matrices are (N+1)*(N+1) because there is professor as well. Then, the average matrix is (N+2)*(N+1) because we add the first N+1 element of the "x" vector as a new row to the average matrix. We will use this last row in the next questions.

Finally, we export the data (matrices) as csv files. Therefore, there are 15 csv files. We will use these data in IBM CPLEX.

Solving the IP Model by using CPLEX and Running Time Plot

Firstly, in order to check whether our algorithm is working or not, we used the matrix which is given in the assignment (figure 1).

Travel Time	Professor	Student 1	Student 2	Student 3	Student 4	Student 5
Professor	-	23	15	42	30	51
Student 1	23	-	34	28	35	45
Student 2	15	34	-	48	62	27
Student 3	42	28	48	-	21	19
Student 4	30	35	62	21	-	36
Student 5	51	45	27	19	36	-

Figure 1 The givent matrix in the assginment

Then, we need to write this matrix into excel because our CPLEX algorithm will take the data from a ".xls" file. Hence, we will do the following processes (figure 2, 3).

	A	B	C	D	E	F
1	0	23	15	42	30	51
2	23	0	34	28	35	45
3	15	34	0	48	62	27
4	42	28	48	0	21	19
5	30	35	62	21	0	36
6	51	45	27	19	36	0
7	0	35	45	20	50	65
8						

Figure 2 The given matrix in excel

Note that the last row includes the study time of the students.

```
n = 6;
SheetConnection sheet("P5_given.xls");
Distance from SheetRead(sheet,"'Sayfa1'!A1:F6");
Time from SheetRead(sheet, "'Sayfa1'!A7:F7");
```

Figure 3 Code that takes data from xls file

If we run the code, we get the following result. This part taken exactly from the CPLEX command file diary part. The screen shot and the solution are as follows.

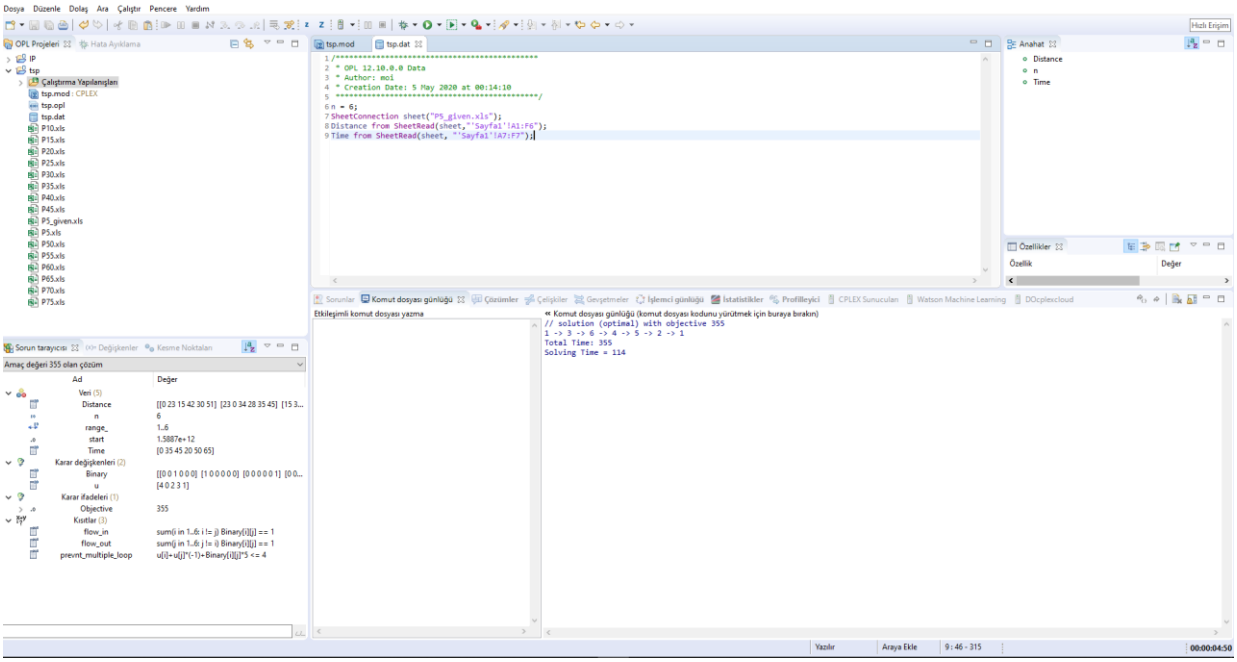

Figure 4 Screenshow of the solution for the given matrix

The solution for the given matrix:

N= 5 (The given Matrix in the assignment)

```
// solution (optimal) with objective 355
1 -> 3 -> 6 -> 4 -> 5 -> 2 -> 1
Total Time: 355
Solving Time = 114
```

In this solution, we can see the optimal path, total time which is the solution of the problem, and the solving time (printing the path is included).

Hence, solutions for the N=5, 10, 15…75 are as follows.

N=5

```
// solution (optimal) with objective 3096
1 -> 2 -> 6 -> 4 -> 3 -> 5 -> 1
Total Time: 3096
Solving Time = 99
```

N=10

```
// solution (optimal) with objective 6054
1 -> 11 -> 4 -> 3 -> 9 -> 7 -> 6 -> 5 -> 2 -> 8 -> 10 -> 1
Total Time: 6054
Solving Time = 122
```

N=15

```
// solution (optimal) with objective 8973
1 -> 16 -> 2 -> 15 -> 4 -> 9 -> 13 -> 11 -> 8 -> 6 -> 5 -> 12 -> 7 -> 10 -> 3 ->
14 -> 1
Total Time: 8973
Solving Time = 176
```

N=20

```
// solution (optimal) with objective 11982
1 -> 21 -> 5 -> 12 -> 14 -> 8 -> 16 -> 9 -> 20 -> 7 -> 15 -> 2 -> 10 -> 18 -> 17 -
> 4 -> 11 -> 6 -> 19 -> 13 -> 3 -> 1
Total Time: 11982
Solving Time = 212
```

N=25

```
// solution (integer optimal, tolerance) with objective 14859
1 -> 11 -> 16 -> 21 -> 18 -> 10 -> 13 -> 14 -> 19 -> 9 -> 26 -> 2 -> 4 -> 22 -> 7
-> 8 -> 23 -> 17 -> 5 -> 6 -> 24 -> 12 -> 3 -> 20 -> 25 -> 15 -> 1
Total Time: 14859
Solving Time = 3344
```

N=30

```
// solution (integer optimal, tolerance) with objective 17832
1 -> 24 -> 17 -> 31 -> 4 -> 30 -> 15 -> 2 -> 28 -> 5 -> 26 -> 23 -> 20 -> 13 -> 27
-> 3 -> 12 -> 25 -> 9 -> 29 -> 21 -> 10 -> 11 -> 22 -> 16 -> 6 -> 8 -> 14 -> 7 ->
18 -> 19 -> 1
Total Time: 17832
Solving Time = 640
```

N=35

```
// solution (integer optimal, tolerance) with objective 20811
1 -> 22 -> 29 -> 30 -> 21 -> 35 -> 6 -> 24 -> 13 -> 12 -> 23 -> 36 -> 5 -> 28 -> 4
-> 16 -> 7 -> 27 -> 31 -> 18 -> 11 -> 34 -> 2 -> 26 -> 15 -> 8 -> 14 -> 10 -> 25 -
> 33 -> 3 -> 19 -> 17 -> 32 -> 20 -> 9 -> 1
Total Time: 20811
Solving Time = 741
```

N = 40

```
// solution (integer optimal, tolerance) with objective 23917
1 -> 33 -> 5 -> 25 -> 34 -> 32 -> 27 -> 20 -> 38 -> 18 -> 31 -> 7 -> 14 -> 15 ->
24 -> 21 -> 22 -> 3 -> 40 -> 26 -> 37 -> 12 -> 13 -> 17 -> 11 -> 4 -> 39 -> 8 ->
35 -> 19 -> 23 -> 41 -> 16 -> 9 -> 28 -> 30 -> 6 -> 29 -> 2 -> 36 -> 10 -> 1
Total Time: 23917
Solving Time = 1656
```

N=45

```
// solution (optimal) with objective 26755
1 -> 30 -> 19 -> 9 -> 12 -> 21 -> 11 -> 6 -> 13 -> 15 -> 40 -> 45 -> 37 -> 35 ->
28 -> 25 -> 26 -> 3 -> 41 -> 33 -> 39 -> 20 -> 42 -> 8 -> 46 -> 24 -> 34 -> 5 ->
16 -> 7 -> 27 -> 44 -> 29 -> 18 -> 17 -> 36 -> 2 -> 23 -> 38 -> 14 -> 31 -> 32 ->
22 -> 43 -> 4 -> 10 -> 1
Total Time: 26755
Solving Time = 750
```

N=50

```
// solution (optimal) with objective 29806
1 -> 7 -> 33 -> 29 -> 36 -> 51 -> 45 -> 34 -> 44 -> 16 -> 6 -> 47 -> 35 -> 23 -> 8
-> 20 -> 27 -> 31 -> 41 -> 15 -> 39 -> 5 -> 18 -> 38 -> 46 -> 12 -> 21 -> 42 -> 48
-> 4 -> 10 -> 26 -> 37 -> 25 -> 2 -> 17 -> 11 -> 32 -> 19 -> 40 -> 22 -> 24 -> 9 -
> 30 -> 43 -> 14 -> 50 -> 28 -> 49 -> 13 -> 3 -> 1
Total Time: 29806
Solving Time = 720
```

N=55

```
// solution (optimal) with objective 32801
1 -> 25 -> 47 -> 48 -> 9 -> 55 -> 52 -> 26 -> 15 -> 7 -> 2 -> 51 -> 45 -> 37 -> 13
-> 49 -> 46 -> 36 -> 3 -> 32 -> 34 -> 31 -> 10 -> 50 -> 14 -> 30 -> 21 -> 8 -> 40
-> 22 -> 43 -> 53 -> 27 -> 33 -> 6 -> 16 -> 38 -> 23 -> 29 -> 12 -> 56 -> 19 -> 42
-> 18 -> 11 -> 5 -> 20 -> 17 -> 39 -> 41 -> 44 -> 4 -> 54 -> 28 -> 35 -> 24 -> 1
Total Time: 32801
Solving Time = 2614
```

N=60

```
// solution (integer optimal, tolerance) with objective 35736
1 -> 20 -> 41 -> 40 -> 61 -> 33 -> 53 -> 56 -> 52 -> 19 -> 59 -> 50 -> 12 -> 21 ->
54 -> 25 -> 58 -> 23 -> 44 -> 42 -> 29 -> 60 -> 49 -> 51 -> 31 -> 36 -> 3 -> 32 ->
15 -> 34 -> 17 -> 43 -> 46 -> 30 -> 16 -> 28 -> 4 -> 55 -> 14 -> 2 -> 27 -> 11 ->
7 -> 45 -> 35 -> 39 -> 24 -> 6 -> 26 -> 38 -> 8 -> 5 -> 18 -> 48 -> 57 -> 37 -> 22
-> 10 -> 13 -> 9 -> 47 -> 1
Total Time: 35736
Solving Time = 2640
```

N=65

```
// solution (integer optimal, tolerance) with objective 38559
1 -> 11 -> 57 -> 2 -> 35 -> 30 -> 19 -> 8 -> 44 -> 63 -> 33 -> 37 -> 14 -> 17 ->
58 -> 32 -> 34 -> 20 -> 3 -> 18 -> 54 -> 59 -> 5 -> 12 -> 60 -> 45 -> 27 -> 50 ->
43 -> 65 -> 6 -> 55 -> 56 -> 38 -> 52 -> 64 -> 4 -> 53 -> 51 -> 15 -> 13 -> 42 ->
48 -> 46 -> 49 -> 26 -> 24 -> 62 -> 61 -> 31 -> 66 -> 10 -> 23 -> 7 -> 28 -> 22 ->
25 -> 41 -> 39 -> 40 -> 21 -> 47 -> 36 -> 16 -> 9 -> 29 -> 1
Total Time: 38559
Solving Time = 7103
```

N=70

```
// solution (integer optimal, tolerance) with objective 41574
1 -> 51 -> 15 -> 32 -> 55 -> 45 -> 58 -> 4 -> 6 -> 71 -> 38 -> 30 -> 10 -> 12 ->
17 -> 48 -> 44 -> 29 -> 7 -> 25 -> 49 -> 54 -> 34 -> 57 -> 27 -> 36 -> 13 -> 11 ->
9 -> 42 -> 69 -> 59 -> 40 -> 26 -> 18 -> 65 -> 56 -> 47 -> 5 -> 20 -> 67 -> 52 ->
8 -> 37 -> 62 -> 19 -> 61 -> 63 -> 64 -> 50 -> 35 -> 22 -> 46 -> 33 -> 66 -> 16 ->
60 -> 2 -> 21 -> 3 -> 24 -> 53 -> 28 -> 31 -> 23 -> 43 -> 70 -> 68 -> 41 -> 39 ->
14 -> 1
Total Time: 41574
Solving Time = 5195
```

N=75

```
// solution (integer optimal, tolerance) with objective 44513
1 -> 57 -> 26 -> 10 -> 69 -> 22 -> 75 -> 48 -> 43 -> 29 -> 19 -> 38 -> 33 -> 66 ->
37 -> 58 -> 34 -> 16 -> 42 -> 11 -> 55 -> 67 -> 14 -> 76 -> 21 -> 60 -> 63 -> 71 -
> 17 -> 47 -> 3 -> 35 -> 23 -> 2 -> 45 -> 56 -> 59 -> 74 -> 72 -> 64 -> 51 -> 30 -
> 9 -> 36 -> 8 -> 53 -> 28 -> 12 -> 41 -> 54 -> 32 -> 49 -> 24 -> 40 -> 31 -> 61 -
> 13 -> 27 -> 5 -> 65 -> 7 -> 50 -> 62 -> 68 -> 18 -> 52 -> 46 -> 25 -> 15 -> 6 ->
44 -> 39 -> 4 -> 73 -> 70 -> 20 -> 1
Total Time: 44513
Solving Time = 58988
```

Running time for the IP model is as follows.

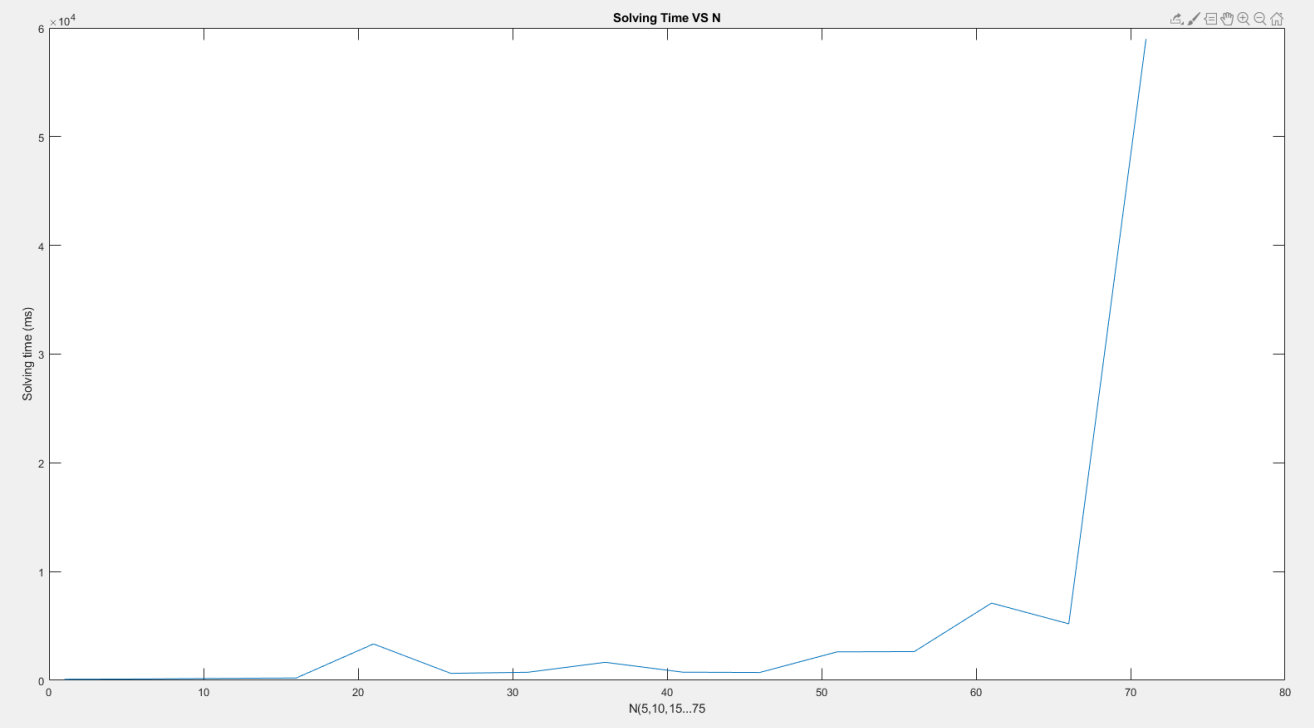

Figure 5 Running time (ms) for N (5,10,15...75) for IP Model

As it can be seen from N=70 to N=75, running time increased significantly. As far as we concerned, for N=100 it could take years to solve.

Solving the DP Model by using Python and Running Time Plot

The same matrices are being used in DP model solution as well. We used python for dynamic programming.

In this part, we have realized that runtime is much larger than IP model solution. For small N such as N=5, DP model solution is faster than IP model solution. However, when N increases, runtime increases significantly.

In this part, firstly, we have used recursion. Therefore, we were tracking all of the location multiple times so, it was inefficient. **Then, we changed our algorithm.** Now, we are storing the distances of the paths and we do not calculate it again. By doing that, we are able to find the solution for N=20. However, it is still inefficient and takes much more time compared to IP model solution. For N is larger than 20, we could not get a solution because it takes so many megabytes in the RAM (more than 25GB) so, our computers cannot compute the solution.

Here is the solution for the given matrix.

Firstly, in order to check whether our algorithm is working or not, we used the matrix which is given in the assignment (figure 6).

Travel Time	Professor	Student 1	Student 2	Student 3	Student 4	Student 5
Professor	-	23	15	42	30	51
Student 1	23	-	34	28	35	45
Student 2	15	34	-	48	62	27
Student 3	42	28	48	-	21	19
Student 4	30	35	62	21	-	36
Student 5	51	45	27	19	36	-

Figure 6 The given matrix

In this solution, we can see the optimal path, total time which is the solution of the problem, and the solving time (printing the path is included).

Hence, solutions for the N=5, 10, 15 and 20 are as follows.

Path for N=5: 1 -> 2 -> 6 -> 4 -> 3 -> 5 -> 1

Total Time: 3096

Runtime: 1ms

Path for N=10: 1 -> 10 -> 8 -> 2 -> 5 -> 6 -> 7 -> 9 -> 3 -> 4 -> 11 -> 1

Total Time: 6054

Runtime: 144ms

Path for N=15: 1 -> 14 -> 3 -> 10 -> 7 -> 12 -> 5 -> 6 -> 8 -> 11 -> 2 -> 15 -> 4 -> 9 -> 13 -> 16 -> 1

Total Time: 8973

Runtime: 13929ms

Path for N=20: 1 -> 3 -> 13 -> 19 -> 6 -> 11 -> 4 -> 17 -> 18 -> 10 -> 2 -> 15 -> 7 -> 20 -> 9 -> 16 -> 8 -> 14 -> 12 -> 5 -> 21 -> 1

Total Time: 11982

Runtime: 931484ms

As it can be seen from the solutions, they are same as the IP model solutions, but paths are different because there could be more than one optimal path for the random matrices.

Running time for the DP model is as follows.

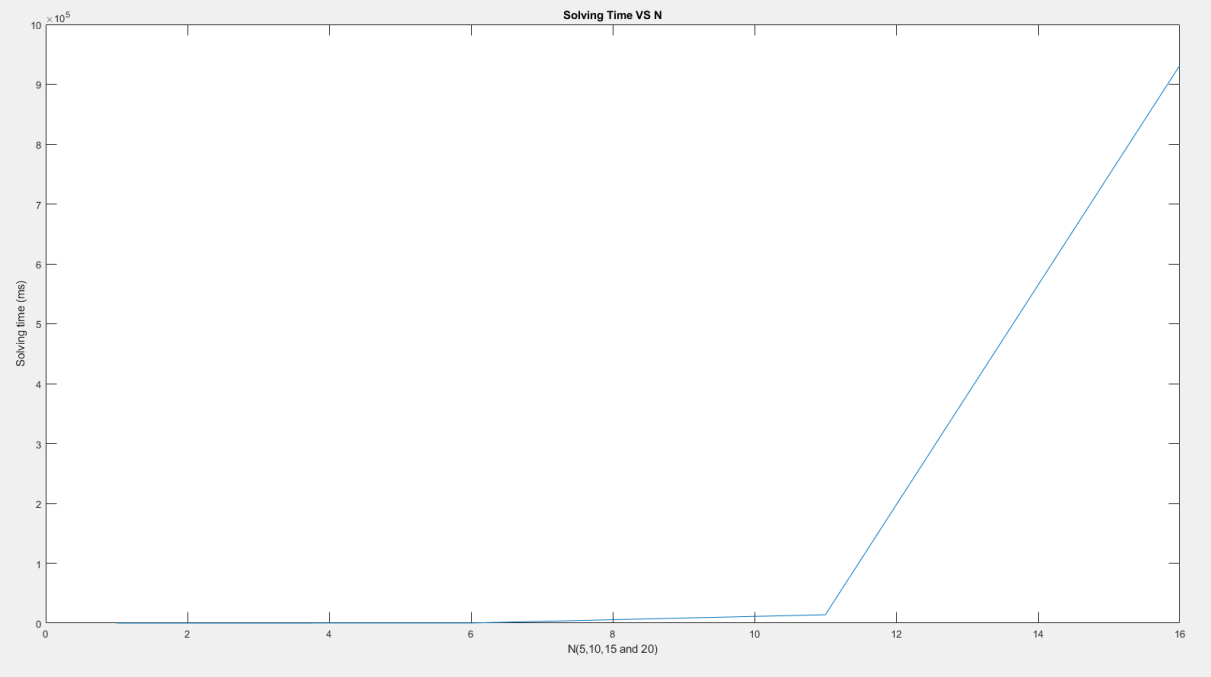

Figure 7 Running time for DP model

As it can be seen from the figure 7, runtime increases dramatically for N = 20. Therefore, using this algorithm for N=25 or more, is inefficient because it will take more than one hour. Hence, we should prefer IP model and solution.

Conclusion

At the beginning, we generate the data to use in the programs regarding specifications in the project description report. Then, we implement program to solve integer programming problem using IBM CPLEX. It is remarkably sufficient to fulfil this purpose with its libraries. We easily get the results from this training using out data. The result apparently increases as N value increases. At some points, it gives shorter path compared to previous one because of randomly generated values. However, general appearance seems increasing.

After that, we implement program to solve dynamic programming problem using Python. We have difficulties in this side of the project since there occur RAM problem to run the program for all N values. We get the results until N = 20; however, it is very hard to get results after this point because required space increases exponentially, and it eventually reaches the full capacity at for N = 20.

We conclude that using IP for this problem is more reasonably regarding our test results. Results can be obtained easily and accurately by this method.

Appendices

MATLAB CODE:

```
clear all, clc
%% Creating Random Study Time
x = zeros(1,76);
for k = 2:76
    x(k) = randi([300,500]);
end
%% Random Matrices for N=5,10,15 ... 75
for h = 1:15
M = zeros(h*5+1);
P = zeros(h*5+1);
N = h*5;
for t = 1: round(N*log(N))
    for k = 1:N+1
        for l = 1:N+1
            if k == l
                M(k,l) = 0;
            else
                M(k,l) = randi([100,300]); %%random integers
between 100 and 300
            end
        end
    end
    for k = 1:h*5+1
            M(k,:) = M(:,k);
    end
    P = P + M;
end
P = round(P/round(N*log(N)));% Average Matrix
P = [P; x(1:N+1)]; %Adding the first N+1 element of the random
vector.
s = strcat("P",int2str(h*5),".csv"); %Exporting the data as
csv file.
writematrix(P,s);
end
```

CPLEX CODE:

Mod Code:

```
/*********************************************
 * OPL 12.10.0.0 Model
 * Author: Emir Ozturk, Erkut Alakus, Omer Faruk Oflaz
 * Creation Date: 5 May 2020 at 00:14:10
 *********************************************/
int n = ...;
range range_ = 1..n;
int Distance[range_][range_] = ...;
int Time[range_] = ...;
float start;
```

```
execute{
var before = new Date();
start = before.getTime();
}

/*********************************************************************************
 * MODEL DECLARATIONS
 *********************************************************************************/
dvar int  Binary[range_][range_] in 0..1;

/*********************************************************************************
 * MODEL
 *********************************************************************************/
dexpr float Objective =(sum(i,j in range_) Binary[i][j]*(Distance[i][j] +
Time[j]));

minimize Objective;
dvar int+ u[2..n];
subject to{
      forall (j in range_)
          flow_in:
         sum (i in range_ : i!=j) Binary[i,j] == 1;

      forall (i in range_ )
        flow_out:
        sum (j in range_ : j!=i) Binary[i,j] == 1;

            forall (i in range_ : i>1, j in range_ : j>1 && j!=i)
                  prevnt_multiple_loop:
                  u[i]-u[j]+(n-1)*Binary[i,j]<=n-2;
  }

execute DISPLAY {
       var after = new Date();
       var check = false;
       var i = 1;
       write(1 + " -> ");
       for(var j = 1; (!check || i!= 1) && j<=n; j++ ){
             if (Binary[i][j] == 1) {
                   check = true;
                   i = j;
                   write(i + " -> ");
                   j = 1;
             }
       }

       writeln("1");
       writeln("Total Time: ", Objective);
       writeln("Solving Time = ", after.getTime()-start);
}
```

Dat Code:

Please note that for (N+1)x(N+2) matrix, n = N+1. Last row should include study time of the professor (it is zero) and students. Also, “A1:BX76” and “A77:BX77” should be changed according to the data inside .xls file.

```
/*********************************************
```

```
 * OPL 12.10.0.0 Data
 * Author: Emir Ozturk, Erkut Alakus, Omer Faruk Oflaz
 * Creation Date: 5 May 2020 at 00:14:10
 *********************************************/
n = 76;
SheetConnection sheet("P75.xls");
Distance from SheetRead(sheet,"'Sayfa1'!A1:BX76");
Time from SheetRead(sheet, "'Sayfa1'!A77:BX77");
```

PYTHON CODE:

```
import xlrd
import copy
import time
g = {}
p = []
def optimized_path(k, a, matrix, times):
    if (k, a) in g:
        return g[k, a]
    values = []
    all_min = []
    for j in a:
        set_a = copy.deepcopy(list(a))
        set_a.remove(j)
        all_min.append([j, tuple(set_a)])
        result = optimized_path(j, tuple(set_a), matrix, times)
        values.append(matrix[k][j] + times[k] + result)

    g[k, a] = min(values)
    p.append(((k, a), all_min[values.index(g[k, a])]))
    return g[k, a]
def get_matrix(n):
    workbook = xlrd.open_workbook(f'P{n}.xls')
    worksheet = workbook.sheet_by_index(0)
    matrix = []
    for i in range(n+1):
```

```
        row = []
        for j in range(n+1):
            row.append(int(worksheet.cell(i, j).value))
        matrix.append(row)
    return matrix
def get_study_times(n):
    workbook = xlrd.open_workbook(f'P{n}.xls')
    worksheet = workbook.sheet_by_index(0)
    matrix = []
    for i in range(n+1):
        matrix.append(int(worksheet.cell(n+1, i).value))
    return matrix
for n in range(5, 76, 5):
    distance_matrix = get_matrix(n)
    time_list = get_study_times(n)
    start = time.time()
    for x in range(1, n+1):
        g[x, ()] = distance_matrix[0][x] + time_list[x]
    total = optimized_path(0, tuple(range(1,n+1)), distance_matrix, time_list)
    print("\n\nBacktracking...")
    print(f'Path for N={n}: 1 -> ', end='')
    solution = p.pop()
    print(solution[1][0]+1, end=' -> ')
    for x in range(n):
        for new_solution in p:
            if tuple(solution[1]) == new_solution[0]:
                solution = new_solution
                print(solution[1][0]+1, end=' -> ')
                break
    print('1')
    print(f'Total Time: {total}')
    run_time = time.time() - start
```

```
print(f"Runtime: {int(run_time*1000)}ms")
g.clear()
del p[:]
```