IE 400

Principles of Engineering Management

Project Report

Faruk Şimşekli	21502464
Gülnihal Muslu	21502606
Tarık Emin Kaplan	21601737

Instructor: Özlem Karsu

IE400 Project Report
May 10, 2020
This report is submitted to the Department of Industrial Engineering of Bilkent University in partial fulfillment of the requirements of the Principle of Engineering Management course IE400.

Contents

1 Problem Definition

A homework has to be assigned and evaluated in these serious Corona times. An assistant with mask will sacrifice himself/herself by taking sterilized homeworks from the professor and distributing it to the students that are located in different places all over the country. There are N students in the course. Since they are doing social distancing they do not go anywhere and stay at home waiting for an homework desperately. The assistant has to distribute the homeworks to all the students one by one. The weird thing is that the assistant waits for the student to complete her/his homework and take it back. Student i needs x_i minutes to complete the homework. Since we do not have teleportation at the moment, it takes some time to go from one place to another. The assistant wants to minimize his time for doing this job, since time is precious. Therefore, s/he will take the homeworks from the professor, go to each student exactly once, wait there, and finally bring back together all the solutions (of the students) to the Professor at the very end. We do not know why, but in a mysterious way, we have ended up in a position where we have to help this assistant in this work to minimize his/her time for doing this duty.

This problem is quite similar to a problem in the literature that we have seen as Traveling Salesman Problem (TSP) [1]. TSP asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?". Since it is an NP-hard problem, it plays an important role in theoretical computer science and operations research.

In our original problem, we have students that are located in different locations and we want to visit them only once and return to the professor's place where we have started at the beginning in a minimum time. We will call the professor's place location 0 and we will treat it similar to student's locations. In addition to TSP, we also have waiting time for each student at their place. As you can see, the resemblance is uncanny.

2 Integer Programming Modelling

For Integer Programming, we will define some parameters and decision variables to model the problem as follows:

Parameters

C_{ij}: the travel time from location i to location j, $\quad i = 0$, 1, ..., N and $j = 0$, 1, ..., N

W_i: the waiting time at location i, $\quad i = 1$, ..., N

Decision Variables

$$X_{ij} = \begin{cases} 1 & \text{if the assistant goes directly from location } i \text{ to location } j, \quad i,\, j = 0, 1, ..., N \\ 0 & \text{otherwise} \end{cases}$$

U_i: Auxiliary variable for solving sub-tour eliminations, $\quad i = 0, 1, ..., N$

Model

$$min \quad \sum_{i=0}^{N} \sum_{j=0}^{N} C_{ij} X_{ij} + \sum_{i=1}^{N} W_i$$

s.t.

$$\sum_{j=0}^{N} X_{ij} = 1$$

$$\sum_{i=0}^{N} X_{ij} = 1$$

$$X_{ii} = 0, \qquad i = 0, 1, ..., N$$

$$U_i - U_j + (N-1) \times X_{ij} \leq N - 2 \qquad \forall_{i,j} \in \{0, 1, ..., N\} \quad [2]$$

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

$$U_i \geq 0 \qquad i = 0, 1, ..., N$$

3 Dynamic Programming Modelling

We will consider the problem as a multistage decision problem. Since the tour is a round trip, we will fix the origin at some location, say location 0 that is professor's place. Suppose that at a certain stage of our optimal tour starting at location 0, one has reached a location i and there remain k location $j_1,\ j_2,\ ...,\ j_k$ to be visited before returning to location 0. Then it is clear that, the tour being optimal, the path from location i through $j_1,\ j_2,\ ...,\ j_k$ in some order and then to location 0 must be of minimum length; for, if not the entire tour could not be optimal, since its total length could be reduced by choosing a shorter path from location i through $j_1,\ j_2,\ ...,\ j_k$ to location 0.

DP Function

$f(i;\ j_1,\ j_2,\ ...,\ j_k)$ = length of a path of minimum length from location i to location 0 which passes only once through each of the remaining k unvisited locations $j_1,\ j_2,\ ...,\ j_k$.

So, if we obtain $f(0;\ j_1,\ j_2,\ ...,\ j_N)$, and a path which has this length, the problem is solved.

Parameters

C_{ij}: the travel time from location i to location j, $i = 0$, 1, ..., N and $j = 0$, 1, ..., N

W_i: the waiting time at location i, $i = 1$, ..., N

Recursive Relationship

$$f(i;\ j_1,\ j_2,\ ...,\ j_N) \ = \ \min_{1 \leq m \leq N}\{D_{ij_m} \ + \ W_{j_m} \ + \ f(i;\ j_1,\ j_2,\ j_{m-1},\ j_{m+1},\ ...,\ j_N)\}$$

The iterative procedure above is initiated through the base case as follows

$$f(i;\ j) = D_{ij} + D_{j0}$$

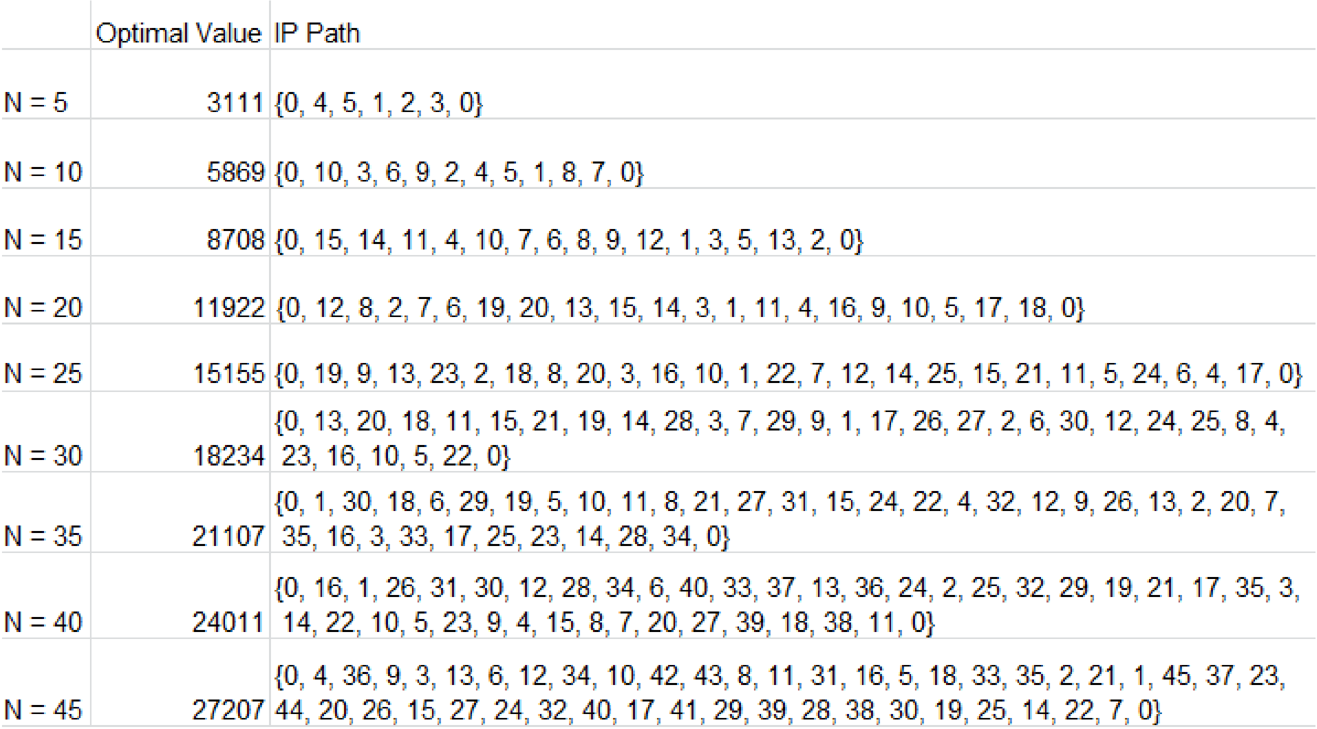
from which we obtain $f(i;\ j_1,\ j_2)$, which, in turn, through the recursive relationship yields $f(i;\ j_1,\ j_2)$, and so on until $f(0;\ j_1,\ j_2,\ ...,\ j_N)$ is obtained. The sequence of values of m which minimize the expression in the braces on the right-hand side of DP function gives a desired minimal path.

4 Randomly Generated Data and Their Solutions

We have used Python as our coding language and used Xpress API in Python as our solver. For simplicity we have limited our values to have no decimals after point (integer). (You can view our data in the appendix section A and text file.)

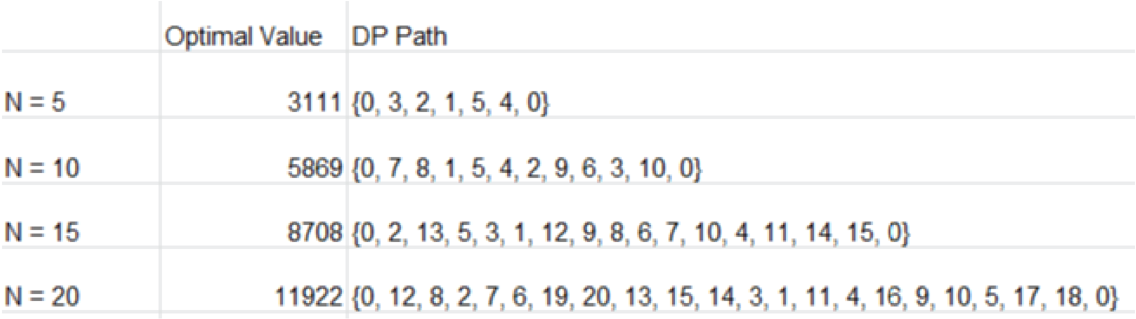
In our solutions, unfortunately, we couldn't find solutions for N values greater than 45, as our solver gave a licence related error indicating that we exceeded a certain space limit for a matrix (the error is not due to our code, rather a community license issue, see the Appendix B). Also, for DP solutions, we could not get the paths for N values greater than 20 as the execution time took more than the limit that is one hour.

In the following table you can see the optimal values and the optimal paths we get for the matrices when we apply IP. (See Appendix A. You can find the rest in the averageMatrices.txt file)



	Optimal Value	IP Path
N = 5	3111	{0, 4, 5, 1, 2, 3, 0}
N = 10	5869	{0, 10, 3, 6, 9, 2, 4, 5, 1, 8, 7, 0}
N = 15	8708	{0, 15, 14, 11, 4, 10, 7, 6, 8, 9, 12, 1, 3, 5, 13, 2, 0}
N = 20	11922	{0, 12, 8, 2, 7, 6, 19, 20, 13, 15, 14, 3, 1, 11, 4, 16, 9, 10, 5, 17, 18, 0}
N = 25	15155	{0, 19, 9, 13, 23, 2, 18, 8, 20, 3, 16, 10, 1, 22, 7, 12, 14, 25, 15, 21, 11, 5, 24, 6, 4, 17, 0}
N = 30	18234	{0, 13, 20, 18, 11, 15, 21, 19, 14, 28, 3, 7, 29, 9, 1, 17, 26, 27, 2, 6, 30, 12, 24, 25, 8, 4, 23, 16, 10, 5, 22, 0}
N = 35	21107	{0, 1, 30, 18, 6, 29, 19, 5, 10, 11, 8, 21, 27, 31, 15, 24, 22, 4, 32, 12, 9, 26, 13, 2, 20, 7, 35, 16, 3, 33, 17, 25, 23, 14, 28, 34, 0}
N = 40	24011	{0, 16, 1, 26, 31, 30, 12, 28, 34, 6, 40, 33, 37, 13, 36, 24, 2, 25, 32, 29, 19, 21, 17, 35, 3, 14, 22, 10, 5, 23, 9, 4, 15, 8, 7, 20, 27, 39, 18, 38, 11, 0}
N = 45	27207	{0, 4, 36, 9, 3, 13, 6, 12, 34, 10, 42, 43, 8, 11, 31, 16, 5, 18, 33, 35, 2, 21, 1, 45, 37, 23, 44, 20, 26, 15, 27, 24, 32, 40, 17, 41, 29, 39, 28, 38, 30, 19, 25, 14, 22, 7, 0}

In the following table you can see the optimal values and the optimal paths we get for the matrices in the appendix and text file when we apply DP:

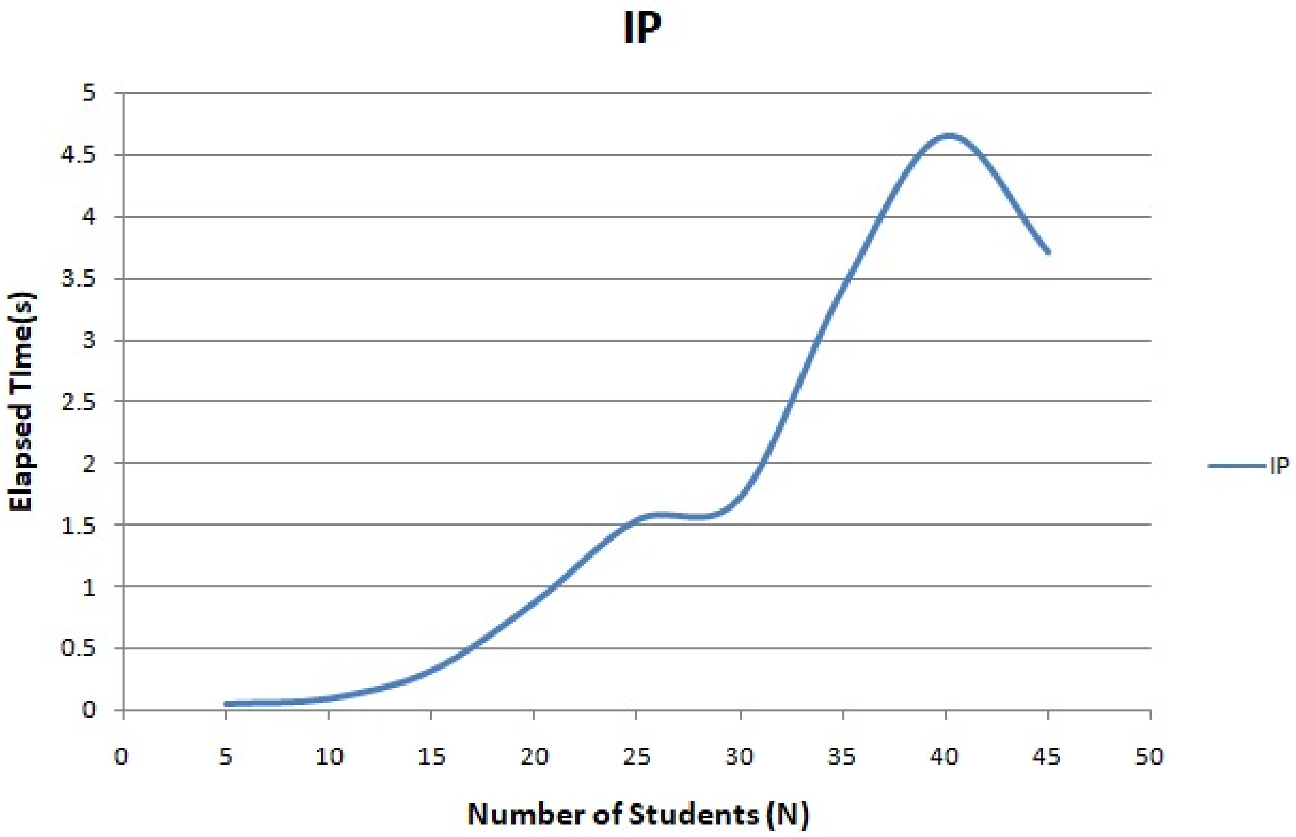


	Optimal Value	DP Path
N = 5	3111	{0, 3, 2, 1, 5, 4, 0}
N = 10	5869	{0, 7, 8, 1, 5, 4, 2, 9, 6, 3, 10, 0}
N = 15	8708	{0, 2, 13, 5, 3, 1, 12, 9, 8, 6, 7, 10, 4, 11, 14, 15, 0}
N = 20	11922	{0, 12, 8, 2, 7, 6, 19, 20, 13, 15, 14, 3, 1, 11, 4, 16, 9, 10, 5, 17, 18, 0}

One thing to mention is that in some cases, the optimal paths for the same matrices are the same except they are reversed of each other, which is taking the path other way around, that is the same logic after all. Also, in our test cases we have encountered some cases that IP have found multiple optimal solutions, yet does not show them all just declares the number of optimal solutions it founds, and that's why the path may differ from the path found by DP.

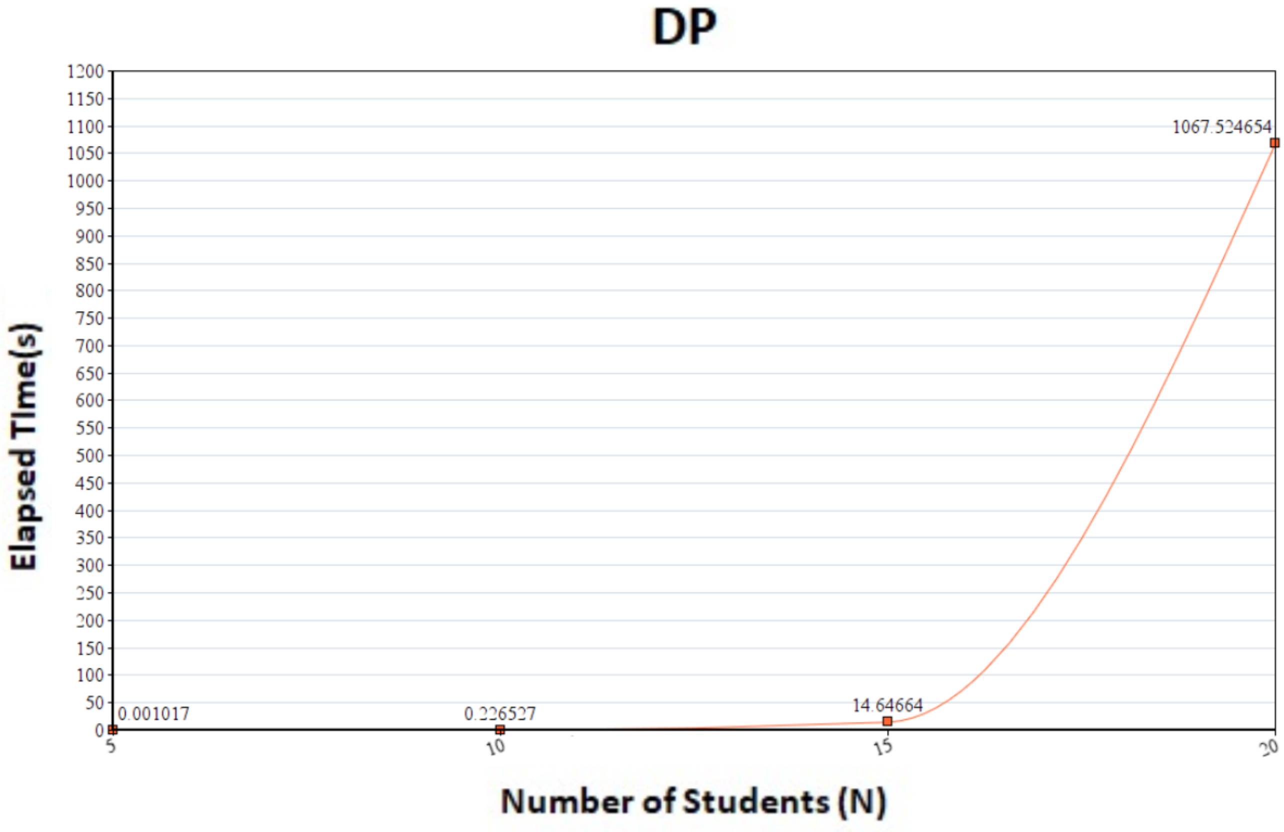
5 Graphical Illustrations and Interpretations

In the following graph you can see the execution time for running IP solution:



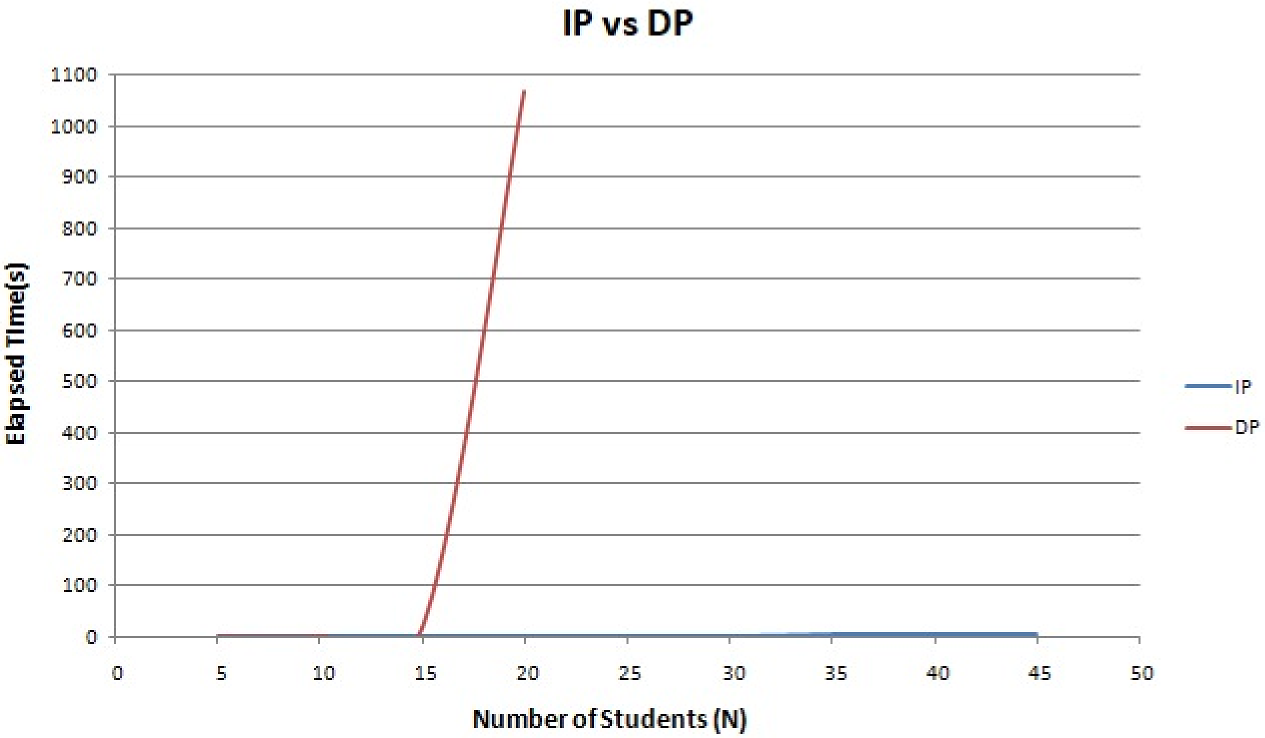
Our solver uses Branch and Bound (BB) and Cutting-Plane methods to solve the given problem. Normally, we would have had to try all the possibilities and the execution time would have been exponential. Using BB method, the solver only branches if there is a better possible solution. As a result, the execution time decreases a considerable amount. Yet, how many eliminations the solver does and at which stage these eliminations occur are still important factors, so the graph has fluctuations. One thing to add is that while N increases, the number of random matrix, that $NlogN$, we have created also increases. The values in the average matrix lie on the range [100 - 300]. Having said that we expect that when we try IP for N = 75, the time will be even smaller. The reason is that the values on the average matrix are even closer to 200, which makes the model converge easily and find the optimal solution earlier.

In the following graph you can see the execution time for running DP solution:



Without Dynamic Programming (DP), if we used brute-force approach, we would have to try all $n!$ different paths. With DP that is known as Bellman–Held–Karp Algorithm [3], we can reduce the complexity from factorial to exponential, if the size of n is relatively small, the problem can be solvable. Theoretically speaking, the time complexity of DP for TSP is $O(n^2 \times 2^n)$, where n is the number of students. Therefore, the execution time of DP for TSP follows an exponentially increasing trend as the number of students increases. As explained before, for N values above 20 the solution takes too long to see the results as we have expected since this question is NP-hard, so we consider those cases unsolvable with DP in our project. The reason that the execution time increases exponentially is that it tries every single possible travelling option.

In the following you can see both in the same graph:



Comparing IP and DP models, it can be said that IP model performs better in general. Although their approach to solve the problem is similar, which is trying available possibilities, using constraints to decide whether to try a possible travelling path to the end makes IP run faster compared to DP. Yet, these same constraints causes an overhead which makes IP run a little slower compared to DP for smaller N values.

6 Conclusion

Solving TSP requires trying all travelling paths to pick the least costly one. In this case the salesman is the assistant and the starting point is where the professor is. To solve this problem, we can consider it as an IP or DP problem. As mentioned before, we have written our code in Python and we used the Xpress library for it to solve the problem as an IP problem. Therefore, our solver optimized the problem using BB and Cutting-Plane methods to see which one is the best. On the other hand, when we used DP to solve the problem, at each stage we computed the possible directions and costs using the previously calculated values in the earlier stages. And at the end, we end up with an optimal value for the problem. However, it is not suitable for large N values. As a result, IP has performed better in general. Yet, there may be a case that there is no other way but to try all possibilities to the end regardless of the constraints. Although in such a case, using DP or IP will not make a huge difference, using DP would be slightly more sensible as IP solver has a constant factor overhead, whereas DP uses memoization and stores the results of previous calls to reach it faster. However, in our case, to continue branching does not make sense because it becomes obvious that continuing to do so will not give a better result, consequently, DP has exponentially increasing running time whereas IP has (roughly) constant running time. The execution time values we have presented in the report also suggests the same. Therefore, for the given problem, we claim that using IP solver is a better choice.

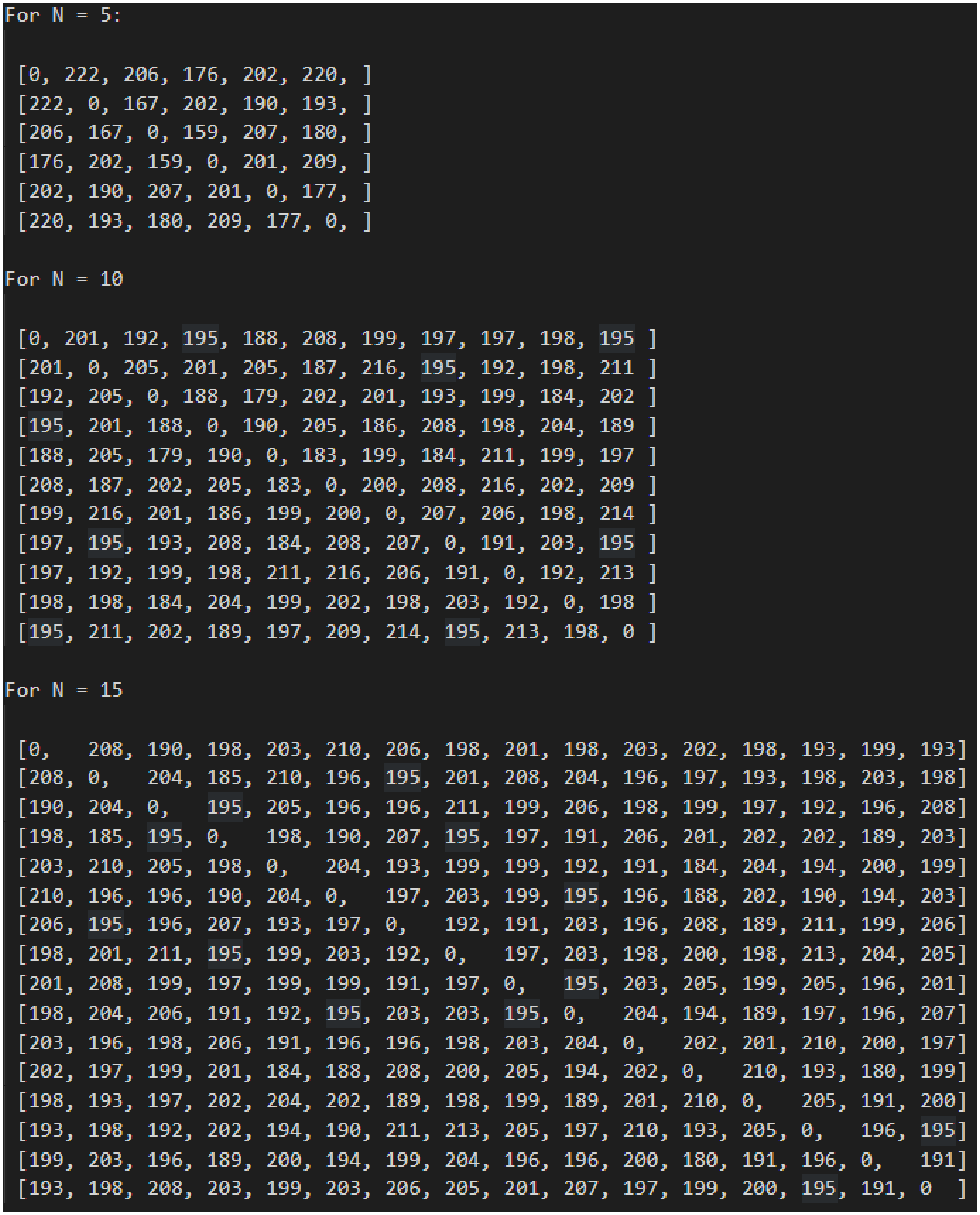
References

[1] "Travelling salesman problem," https://en.wikipedia.org/wiki/Travelling_salesman_problem/, Accessed: 2020-05-03.

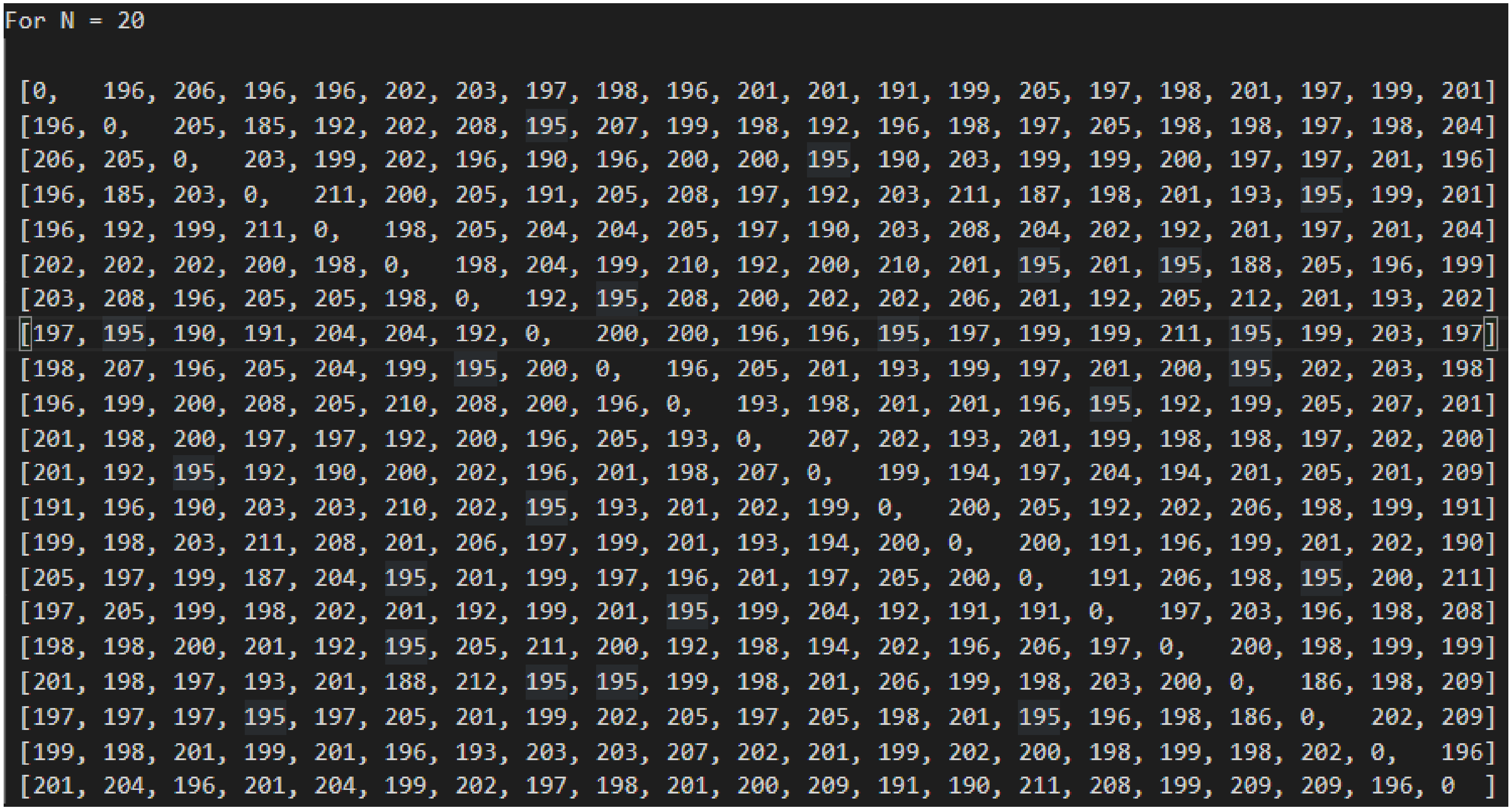
[2] "Integer Programming Formulation of Traveling Salesman Problems," https://dl.acm.org/doi/pdf/10.1145/321043.321046, Accessed: 2020-05-05.

[3] "Held–Karp algorithm," https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm, Accessed: 2020-05-09.

A Appendix - Data

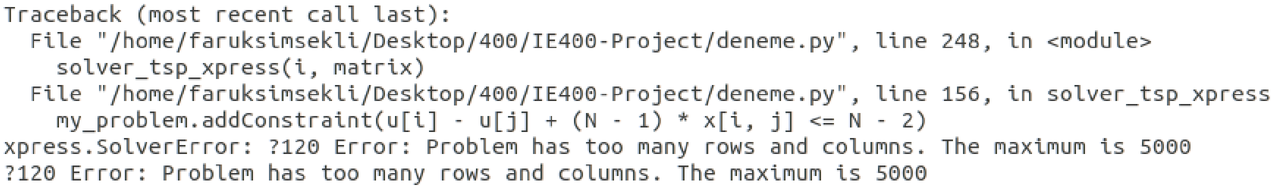


```
For N = 5:

 [0, 222, 206, 176, 202, 220, ]
 [222, 0, 167, 202, 190, 193, ]
 [206, 167, 0, 159, 207, 180, ]
 [176, 202, 159, 0, 201, 209, ]
 [202, 190, 207, 201, 0, 177, ]
 [220, 193, 180, 209, 177, 0, ]

For N = 10

 [0, 201, 192, 195, 188, 208, 199, 197, 197, 198, 195 ]
 [201, 0, 205, 201, 205, 187, 216, 195, 192, 198, 211 ]
 [192, 205, 0, 188, 179, 202, 201, 193, 199, 184, 202 ]
 [195, 201, 188, 0, 190, 205, 186, 208, 198, 204, 189 ]
 [188, 205, 179, 190, 0, 183, 199, 184, 211, 199, 197 ]
 [208, 187, 202, 205, 183, 0, 200, 208, 216, 202, 209 ]
 [199, 216, 201, 186, 199, 200, 0, 207, 206, 198, 214 ]
 [197, 195, 193, 208, 184, 208, 207, 0, 191, 203, 195 ]
 [197, 192, 199, 198, 211, 216, 206, 191, 0, 192, 213 ]
 [198, 198, 184, 204, 199, 202, 198, 203, 192, 0, 198 ]
 [195, 211, 202, 189, 197, 209, 214, 195, 213, 198, 0 ]

For N = 15

 [0,   208, 190, 198, 203, 210, 206, 198, 201, 198, 203, 202, 198, 193, 199, 193]
 [208, 0,   204, 185, 210, 196, 195, 201, 208, 204, 196, 197, 193, 198, 203, 198]
 [190, 204, 0,   195, 205, 196, 196, 211, 199, 206, 198, 199, 197, 192, 196, 208]
 [198, 185, 195, 0,   198, 190, 207, 195, 197, 191, 206, 201, 202, 202, 189, 203]
 [203, 210, 205, 198, 0,   204, 193, 199, 199, 192, 191, 184, 204, 194, 200, 199]
 [210, 196, 196, 190, 204, 0,   197, 203, 199, 195, 196, 188, 202, 190, 194, 203]
 [206, 195, 196, 207, 193, 197, 0,   192, 191, 203, 196, 208, 189, 211, 199, 206]
 [198, 201, 211, 195, 199, 203, 192, 0,   197, 203, 198, 200, 198, 213, 204, 205]
 [201, 208, 199, 197, 199, 199, 191, 197, 0,   195, 203, 205, 199, 205, 196, 201]
 [198, 204, 206, 191, 192, 195, 203, 203, 195, 0,   204, 194, 189, 197, 196, 207]
 [203, 196, 198, 206, 191, 196, 196, 198, 203, 204, 0,   202, 201, 210, 200, 197]
 [202, 197, 199, 201, 184, 188, 208, 200, 205, 194, 202, 0,   210, 193, 180, 199]
 [198, 193, 197, 202, 204, 202, 189, 198, 199, 189, 201, 210, 0,   205, 191, 200]
 [193, 198, 192, 202, 194, 190, 211, 213, 205, 197, 210, 193, 205, 0,   196, 195]
 [199, 203, 196, 189, 200, 194, 199, 204, 196, 196, 200, 180, 191, 196, 0,   191]
 [193, 198, 208, 203, 199, 203, 206, 205, 201, 207, 197, 199, 200, 195, 191, 0  ]
```



```
For N = 20

 [0,   196, 206, 196, 196, 202, 203, 197, 198, 196, 201, 201, 191, 199, 205, 197, 198, 201, 197, 199, 201]
 [196, 0,   205, 185, 192, 202, 208, 195, 207, 199, 198, 192, 196, 198, 197, 205, 198, 198, 197, 198, 204]
 [206, 205, 0,   203, 199, 202, 196, 190, 196, 200, 200, 195, 190, 203, 199, 199, 200, 197, 197, 201, 196]
 [196, 185, 203, 0,   211, 200, 205, 191, 205, 208, 197, 192, 203, 211, 187, 198, 201, 193, 195, 199, 201]
 [196, 192, 199, 211, 0,   198, 205, 204, 204, 205, 197, 190, 203, 208, 204, 202, 192, 201, 197, 201, 204]
 [202, 202, 202, 200, 198, 0,   198, 204, 199, 210, 192, 200, 210, 201, 195, 201, 195, 188, 205, 196, 199]
 [203, 208, 196, 205, 205, 198, 0,   192, 195, 208, 200, 202, 202, 206, 201, 192, 205, 212, 201, 193, 202]
 [197, 195, 190, 191, 204, 204, 192, 0,   200, 200, 196, 196, 195, 197, 199, 199, 211, 195, 199, 203, 197]
 [198, 207, 196, 205, 204, 199, 195, 200, 0,   196, 205, 201, 193, 199, 197, 201, 200, 195, 202, 203, 198]
 [196, 199, 200, 208, 205, 210, 208, 200, 196, 0,   193, 198, 201, 201, 196, 195, 192, 199, 205, 207, 201]
 [201, 198, 200, 197, 197, 192, 200, 196, 205, 193, 0,   207, 202, 193, 201, 199, 198, 198, 197, 202, 200]
 [201, 192, 195, 192, 190, 200, 202, 196, 201, 198, 207, 0,   199, 194, 197, 204, 194, 201, 205, 201, 209]
 [191, 196, 190, 203, 203, 210, 202, 195, 193, 201, 202, 199, 0,   200, 205, 192, 202, 206, 198, 199, 191]
 [199, 198, 203, 211, 208, 201, 206, 197, 199, 201, 193, 194, 200, 0,   200, 191, 196, 199, 201, 202, 190]
 [205, 197, 199, 187, 204, 195, 201, 199, 197, 196, 201, 197, 205, 200, 0,   191, 206, 198, 195, 200, 211]
 [197, 205, 199, 198, 202, 201, 192, 199, 201, 195, 199, 204, 192, 191, 191, 0,   197, 203, 196, 198, 208]
 [198, 198, 200, 201, 192, 195, 205, 211, 200, 192, 198, 194, 202, 196, 206, 197, 0,   200, 198, 199, 199]
 [201, 198, 197, 193, 201, 188, 212, 195, 195, 199, 198, 201, 206, 199, 198, 203, 200, 0,   186, 198, 209]
 [197, 197, 197, 195, 197, 205, 201, 199, 202, 205, 197, 205, 198, 201, 195, 196, 198, 186, 0,   202, 209]
 [199, 198, 201, 199, 201, 196, 193, 203, 203, 207, 202, 201, 199, 202, 200, 198, 199, 198, 202, 0,   196]
 [201, 204, 196, 201, 204, 199, 202, 197, 198, 201, 200, 209, 191, 190, 211, 208, 199, 209, 209, 196, 0  ]
```

B Appendix - Error



```
Traceback (most recent call last):
  File "/home/faruksimsekli/Desktop/400/IE400-Project/deneme.py", line 248, in <module>
    solver_tsp_xpress(i, matrix)
  File "/home/faruksimsekli/Desktop/400/IE400-Project/deneme.py", line 156, in solver_tsp_xpress
    my_problem.addConstraint(u[i] - u[j] + (N - 1) * x[i, j] <= N - 2)
xpress.SolverError: ?120 Error: Problem has too many rows and columns. The maximum is 5000
?120 Error: Problem has too many rows and columns. The maximum is 5000
```