IE400 Project

Can Savcı - 21300803
Osman Can Yıldız - 21302616

6 August 2020

1 Introduction

A cargo truck has to visit 30 locations on an island to deliver cargoes. There is only a single two-way road to travel from any of these two locations. These roads have no turning points i.e. they are linear. Suppose that the locations are on a tropical island where regional storms occur in circular shapes and surround certain regions of the island. If some part of the road is surrounded by the storm, the cargo truck prefers not to choose that road due to its possible dangers. Suppose that cargo truck is noticed by the exact locations of these storms before it starts to deliver the cargoes. Moreover, suppose that the cargo truck moves with a constant speed along the road. There are three types of roads on this island: asphalt, concrete and gravel.

Speed of the truck differs based on the road type:

Asphalt	Concrete	Gravel
100 unit/hour	65 unit/hour	35 unit/hour

Table 1: The Road Type

To give an example, let there be three locations A,B,C as in the following figure:

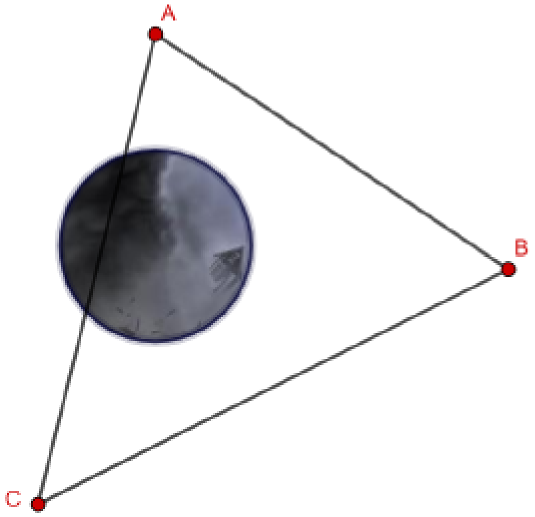


Figure 1: Example

- The road between A and C is blocked by a circular storm. It cannot be used by the cargo truck.
- The road between B and C can be used by the cargo truck.
- The road between A and B can be used by the cargo truck.

Now, let the Cartesian coordinate of A to be (0, 0) and B to be (3, -2). Then, the length of the road is ≈ 3.60555 units. If the type of the road is asphalt, the cargo truck travels this road in 0.0360555 hours.

2 Objective

In this project, you are given the Cartesian coordinates of 30 locations (first sheet of data.xlsx) and the storm regions defined by the center and radius of the storm (second sheet of data.xlsx). The type of the road between any of the two locations are also provided (the third sheet of data.xlsx).
Suppose that the cargo truck drops the cargoes immediately and leaves the location. Hence, no time delay occurs. The cargo truck's aim is to visit all the locations from a starting point (the first location in the first sheet of data.xlsx) and then turn back to its initial location while **minimizing the traveling time.**

3 Process of Modelling

To solve the Travelling Salesman Problem, we used IBM ILOG CPLEX Optimization Studio and we used Python programming language and Networkx library to simulate the path. We used data.xlsx file to extract the information of the Cartesian coordinates of 30 locations, the storm regions defined by the center, radius of the storm and the type of the road between any of the two locations.

3.1 Problem Analysis

The problem is the application of Travelling Salesman Problem or TSP. Because, the question implies that it is undirected graph and the cargo truck should visit all of the 30 locations on the island. The truck driver cannot pass through the same node twice. TSP can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e., each pair of vertices is connected by an edge). If no path exists between two cities, adding an arbitrarily long edge will complete the graph without affecting the optimal tour.

3.2 Travelling Salesman Problem

"The TSP can be formulated as an integer linear program. Even though the problem has the several formulations, two notable formulations are the Miller–Tucker–Zemlin (MTZ) formulation and the Dantzig–Fulkerson–Johnson (DFJ) formulation. The DFJ formulation is stronger, though the MTZ formulation is still useful in certain settings. We used Miller-Tucker-Zemlin formulation in our project." [1]

3.2.1 Miller-Tucker-Zemlin Formulation

Label the cities with the numbers 1, . . . , n and define:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

For i = 1, . . . , n, let u_i be a dummy variable, and finally take $c_{ij} > 0$ to be the distance from location i to location j. Then TSP can be written as the following integer linear programming problem:

$$
\begin{aligned}
&\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \colon && \\
&\quad x_{ij} \in \{0,1\} && i,j = 1,\ldots,n; \\
&\quad u_i \in \mathbf{Z} && i = 2,\ldots,n; \\
&\quad \sum_{i=1, i \neq j}^{n} x_{ij} = 1 && j = 1,\ldots,n; \\
&\quad \sum_{j=1, j \neq i}^{n} x_{ij} = 1 && i = 1,\ldots,n; \\
&\quad u_i - u_j + n x_{ij} \leq n - 1 && 2 \leq i \neq j \leq n; \\
&\quad 0 \leq u_i \leq n - 1 && 2 \leq i \leq n.
\end{aligned}
$$

The first set of equalities requires that each location is arrived at from exactly one other location, and the second set of equalities requires that from each location there is a departure to exactly one other location. The last constraints enforce that there is only a single tour covering all locations, and not two or more disjointed tours that only collectively cover all locations.

3.3 Parameters

The descriptions about parameters are provided in the table below.

Name	Type	Description
r1	range	Number of locations in the given problem
Locations_X_Coordinate	float array	X coordinates of the locations
Locations_Y_Coordinate	float array	Y coordinates of the locations
r2	range	The number of storms in the island
Storms_X_Coordinate	float array	X coordinates of the center of circular storms
Storms_Y_Coordinate	float array	Y coordinates of the center of circular storms
Storms_Radius	float array	Radii of the circular storms
r3	range	Number of roads in the island
RM_roadMaterial	string array	The types of each road in the island
edge	tuple	The tuple with the beginning and the end nodes
edges	set	The collection of edges
distances	float array	The distances of the edges
averageTime	float array	The average time spent when travelling the road
boolStorm	int array	Stores if there is a storm in any road
count	int	Mapping key from the roads to the road types

Table 2: Parameter Description

The information about the number of parameters are delivered in the table below.

Parameter Name	Parameter Type	Number of Parameters
r1	range	30
Locations_X_Coordinate	float array	30
Locations_Y_Coordinate	float array	30
r2	range	20
Storms_X_Coordinate	float array	20
Storms_Y_Coordinate	float array	20
Storms_Radius	float array	20
r3	range	870
RM_roadMaterial	string array	870
edges	set	870
distances	float array	870
averageTime	float array	870
boolStorm	int array	870
count	int	870

Table 3: The Number of Parameters

Parameter **r1** represents the number of locations and it is known from the problem statement, we stored the parameter as the type range. Parameter **r2** represents the number of storms and it is found using the data.xlsx file and we stored the parameter **r2** as the type range. Parameter **r3** represents the number of storms and it is found using the data.xlsx file and we stored the parameter **r3** as the type range. Parameters **Locations_X_Coordinate** and **Locations_Y_Coordinate** is declared as the float arrays with the size equal to the number of locations and they are assigned the numbers in the corresponding columns at the Locations sheet. Parameters **Storms_X_Coordinate**, **Storms_Y_Coordinate** and **Storms_Radius** is declared as the float arrays with the size equal to the number of storms and they are assigned the numbers in the corresponding columns at the Storms sheet. Parameter **RM_roadMaterial** represents the road type and it is declared as the string array with the size equal to the number of locations and **RM_roadMaterial** is assigned the numbers in the corresponding columns at the Road Material sheet. These parameters in the above are obtained from the excel file data.xlsx.

The tuple **edge** is declared to store the edges between the locations, it has two elements. The first one is **i**, it represents the beginning location of the edge; whereas, the second one is **j**, it represents the end location of the edge. The set **edges** is the collection of all edges whose beginning location and end location are any locations in the island. The beginning and the end locations of any edge cannot be the same; therefore, using the range **r1**, we could create the set of edges which starts with (1,2) and ends with (30,29). To make it compatible with our range **r1**, we decided to start enumerating our locations with 0.

The float array **distances** is declared to store the distance between the beginning and the end locations of any edge. The function **getDistance** is created to find the distance and it is created in the execute block. **getDistance** method takes two parameters: the beginning location i and the end location j, returns the distance as a floating point number. After that, the set **edges** is traversed. The edges and the distances are mapped through distance float array. After the distances are stored in the **distances** array, the float array **averageTime** is declared to find the average time spent at travelling the edge, to do that the distance of any edge is obtained from **distances** array and the distance is divided by the velocity which is shown in Table 1 with respect to the road type which can be obtained from the corresponding index in **RM_roadMaterial** string array. The mapping is actualized by initializing the **count** integer parameter with 0 and incrementing when the mapping occurs between an element of **averageTime** array and the average time found by implementing the process mentioned above.

The binary integer array **boolStorm** is declared to store if some part of any edge is surrounded by a circular storm. **boolStorm** array can take only binary values 0,1. First, all edges are considered supposedly as available, namely, no storms occur in any road. All elements of the **boolStorm** array is initialized with ones. After the initialization step, the edges are checked whether they are overlapping with the location of any circular storm with the helper function **isOverlapping** which is created in execute block and takes three parameters: the beginning location and the end location of the edge and the index of storm whose information are stored in **Storms_X_Coordinate**, **Storms_Y_Coordinate** and **Storms_Radius** and the function returns boolean value which represents if there is an overlapping circular storm in the particular edge.

The implementation details of the **isOverlapping** can be expressed as follows. Suppose that we have a edge with the edge formula $y = m \cdot x + n$ with m being a slope and n being the bias. In addition to this, we have a circle with the center $C(c_1, c_2)$ with the radius r; therefore, the formula of the circle can be expressed as $(x - c_1)^2 + (y - c_2)^2 = r^2$. In order to claim that they are overlapping, the orthogonal projection of the center $C(c_1, c_2)$ on the line can be found.

Suppose that the projected point of the center is called C' with the coordinates $C'(c_1', c_2')$. C' should be between the locations of the particular edge. To check whether the condition is satisfied, **isBetween** method is created. Suppose that we have the locations $A(x_1, y_1)$ and $B(x_2, y_2)$ of the edge. **isBetween** method is checking that

$$min(x_1, x_2) \leq c_1' \leq max(x_1, x_2) \tag{1}$$

$$min(y_1, y_2) \leq c_2' \leq max(y_1, y_2) \tag{2}$$

We need to find the orthogonal projection of center with respect to the edge, namely, $C'(c_1', c_2')$ can be found with the process which will be described below. Suppose that we took the orthogonal projection of the edge AB with the locations $A(x_1, y_1)$ and $B(x_2, y_2)$ and we found the $C'(c_1', c_2')$ point. If m being the slope of any line or line segment, we know that

$$m_{CC'} \cdot m_{AB} = -1 \tag{3}$$

We deducted above equation from the fact that CC' and AB lines are orthogonal to each other. After getting the result, we actually know the slope m of AB which is

$$m_{AB} = \frac{y_2 - y_1}{x_2 - x_1} = m \tag{4}$$

We can claim from the fact (3) that the slope of CC' is

$$m_{CC'} = \frac{x_1 - x_2}{y_2 - y_1} = \frac{-1}{m} \tag{5}$$

We can find the line equations of AB and CC'. The line equation of AB is

$$\frac{(y - y_1)}{(x - x_1)} = \frac{y_2 - y_1}{x_2 - x_1} = m \tag{6}$$

The bias n of the edge AB can be found using the formula

$$n = -\frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 + y_1 = -m \cdot x_1 + y_1 \tag{7}$$

Therefore, the line equation of CC' will be

$$y = \frac{x_1 - x_2}{y_2 - y_1} \cdot (x - c_1) + c_2 = \frac{-1}{m} \cdot (x - c_1) + c_2 \tag{8}$$

It is known that the lines who are orthogonal to each other can only overlap in one point which is $C'(c'_1, c'_2)$. That means $C'(c'_1, c'_2)$ can satisfy both equations mentioned above.

$$\frac{(c'_2 - y_1)}{(c'_1 - x_1)} = \frac{y_2 - y_1}{x_2 - x_1} = m \tag{9}$$

$$c'_2 = \frac{x_1 - x_2}{y_2 - y_1} \cdot (c'_1 - c_1) + c_2 = \frac{-1}{m} \cdot (c'_1 - c_1) + c_2 \tag{10}$$

We can arrange the equations (6) and (8) such that

$$-m \cdot x + y = n \tag{11}$$

$$\frac{1}{m} \cdot x + y = \frac{c_1}{m} + c_2 \tag{12}$$

We can use the linear algebra to solve both equations above and find the intersection point of both lines $C'(c'_1, c'_2)$. Let us create a matrix M which stores the coefficients of x and y and a vector b which stores the results. Our matrix M will be $M = \begin{bmatrix} -m & 1 \\ \frac{1}{m} & 1 \end{bmatrix}$ and our vector b $= \begin{bmatrix} n \\ \frac{c_1}{m} + c_2 \end{bmatrix}$. For$p = \begin{bmatrix} x \\ y \end{bmatrix}$, we can obtain the linear equation $M \cdot p = b$. The solution of the linear equation $p = M^{-1} \cdot b$. Therefore, we could obtain the solution through solving the equation

$$det|M| = -m - \frac{1}{m} \tag{13}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{-m - \frac{1}{m}} \cdot \begin{bmatrix} 1 & -1 \\ -\frac{1}{m} & -m \end{bmatrix} \cdot \begin{bmatrix} n \\ \frac{c_1}{m} + c_2 \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{-m}{m^2+1} \cdot \begin{bmatrix} n - c_2 - \frac{c_1}{m} \\ -\frac{n}{m} - c_1 - c_2 \cdot m \end{bmatrix} \tag{15}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{m^2+1} \cdot \begin{bmatrix} m \cdot (c_2 - n) + c_1 \\ n + m \cdot (c_1 + m \cdot c_2) \end{bmatrix} \tag{16}$$

After finding the coordinates $C'(c'_1, c'_2)$ from the linear equation, $(c'_1 = xandc'_2 = y)$, we can check if $C'(c'_1, c'_2)$ is between the locations of the particular edge. We can check it via **isBetween** method and the implementation of the **isBetween** method is mentioned above. If $C'(c'_1, c'_2)$ is between the locations of the edge, we need to check the second constraint which will be mentioned below. The edge and the circle can take two different positions.

- The edge and the circle can overlap at least one point.
- The edge and the circle never overlaps.

If the first condition occurs, that means the distance between the center $C(c_1, c_2)$ of the circular storm and the orthogonal projection of the center $C'(c'_1, c'_2)$ is less than or equal to the radius of the circular storm. We can find the distance of CC' line segment without explicitly finding $C'(c'_1, c'_2)$; since, CC' is the orthogonal projection of the particular edge. The formula for finding the orthogonal projection of the center on the line $y = m \cdot x + n$ is

$$|CC'| = \frac{|c_2 - m \cdot c_1 - n|}{\sqrt{m^2+1}} \leq r \tag{17}$$

If the second condition occurs, that means the edge and the circle above never overlap. Another claim can be proposed that the distance of CC' line segment is greater than the radius of the circular storm, namely

$$|CC'| = \frac{|c_2 - m \cdot c_1 - n|}{\sqrt{m^2+1}} > r \tag{18}$$

The process mentioned above is the logic behind the **isOverlapping** method. To implement **isOverlapping**, two helper functions are created. The first helper function **getSlope** finds the slope of the edge using the beginning node and the end node. The parameters of the **getSlope** method are the beginning and the end location of the edge, the function returns a floating point number. The slope m of the edge they are linked can be found using the formula

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{19}$$

The second helper function **getBias** finds the slope of the edge using the beginning node and the end node. The parameters of the **getBias** method are the beginning and the end location of the edge, the function returns a floating point number. The bias n of the edge they are linked can be found using the formula

$$n = -\frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 + y_1 \tag{20}$$

The circular storm with the center $C(c_1, c_2)$ with the radius r can be obtained from the indexes of **Storms_X_Coordinate**, **Storms_Y_Coordinate** and **Storms_Radius** arrays. To determine whether an edge and a circular storm overlap, we have all we need. First, we can find the orthogonal projection of the center of the circular storm on the edge. Using the process mentioned above, we can find the projection point $C'(c'_1, c'_2)$. To check whether the point C' is between the locations of the particular edge, we use **isBetween**. If the point C' is not between the locations, we can return false, we are done. Otherwise, we can find the distance between the center and the projection point via the inequality (17). If the inequality is satisfied, that means the circular storm is blocking the road. Otherwise, the road is not blocked. To conclude, the function returns if the distance between the projection point of the center and the center itself is less than the radius, namely, the circular storm is overlapped with the road. After the implementation of the **isOverlapping** function, we traverse the elements of **boolStorm** array, namely, for each edge, we look for any overlapping with given storms. If the overlapping is found between the edge and any circular storm, we set the edge to zero in the **boolStorm** array. It means that we cannot use the particular road to travel because it is overlapping with any circular storm.

3.4 Decision Variables

The descriptions about decision variables are provided in the table below.

Name	Type	Description
validPath	int array	Stores the optimal path
u	float array	The dummy positive decision variable

Table 4: Decision Variable Description

The number of parameters are delivered in the table below.

Name	Type	Number of Decision Variables
validPath	int array	870
u	float array	29

Table 5: The Number of Decision Variables

The first decision variable named validPath is used to represent that each location is arrived at from exactly one other location and each location there is a departure to exactly one other location namely, the x_{ij}'s in Miller-Tucker-Zemlin formulation. The second decision variable named u_i such that $2 \leq i \leq n$ is a dummy positive variable and it is used to model the constraint that there is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities.

3.5 Constraints

- The velocity of the cargo truck should not be less than zero.
- The beginning and end nodes should not be the same.
- The decision variable should only take the value which is either 0 or 1.
- The edge should be linear, namely, it should have only one beginning node and only one end node.
- The time spent on passing through any edge should not be less than zero.
- There is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities.
- If any circular storm and any edge overlaps, the edge is considered as blocked and the cargo truck cannot use the edge to travel.

The first and fourth constraint is handled in execute block when the elements of **averageTime** float array is set to the particular distance divided by the velocity. The velocities with respect to the road types are hard coded. The second constraint are handled when the set **edges** are created. The remaining constraints are handled in subject to block.

In the third constraint, we need to ensure that each location is arrived at from exactly one other location, namely, the edge has to have only one beginning location. Suppose that n being the number of locations and x_{ij} being the boolean variable which represents whether the path from the location i to the location j is included in the optimal path, the model of the constraint can be expressed that

$$\sum_{i=1,i\neq j}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{21}$$

We need to ensure that from each location there is a departure to exactly one other location, namely, the edge has to have only one end location as well. Suppose that n being the number of locations and x_{ij} being the boolean variable which represents whether the path from the location i to the location j is included in the optimal path, the model of the constraint can be expressed that

$$\sum_{j=1,i\neq j}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{22}$$

In the fifth constraint, the constraint enforces that there is only a single tour covering all locations and not two or more disjointed tours that only collectively cover all locations, namely, the optimal path should include every location only once. Suppose that n being the number of locations, u being the dummy positive variable and x_{ij} being the boolean variable which represents whether the path from the location i to the location j is included in the optimal path, the model of the constraint can be expressed that

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{23}$$

$$0 \leq u_i \leq n - 1 \qquad 2 \leq i \leq n; \tag{24}$$

In the last constraint, the constraint enforces that the edge should not overlap with any of the given circular storm, namely, the optimal path should not include any blocked edge. We can model the last constraint by modifying the third constraint. Suppose that S being the set of the edges which is blocked by the circular storm, n being the number of locations and x_{ij} being the boolean variable which represents whether the path from the location i to the location j is included in the optimal path, the model of the constraint can be expressed that

$$\sum_{i=1, i \neq j, x_{ij} \notin S}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{25}$$

$$\sum_{j=1, i \neq j, x_{ij} \notin S}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{26}$$

3.6 Objective Function

The objective function is total time spent at visiting each location in the island. The parameter for objective function is **TotalTime** and it can be calculated by taking the summation of element-wise multiplication between **averageTime** matrix and **validPath** matrix. Suppose that S being the set of the edges which is blocked by the circular storm, n being the number of locations, x_{ij} being the boolean variable which represents whether the path from the location i to the location j is included in the optimal path, matrix a is the **averageTime** matrix and x is the **validPath** matrix, the model of objective function can be expressed that

$$x_{ij} \in 0, 1 \qquad i, j = 1, ..., n \tag{27}$$

$$min \sum_{i=1}^{n} \sum_{j=1, i \neq j, x_{ij} \notin S}^{n} a_{ij} \cdot x_{ij} = 1 \qquad i, j = 1, ..., n \tag{28}$$

3.7 Model

In the model, we are minimizing the total time spent subject to the constraints mentioned above and some of the constraints are handled before we construct our model. It is time to put them together.

minimize {

$$x_{ij} \in 0, 1 \qquad i, j = 1, ..., n \tag{29}$$

$$min \sum_{i=1}^{n} \sum_{j=1, i \neq j, x_{ij} \notin S}^{n} a_{ij} \cdot x_{ij} = 1 \qquad i, j = 1, ..., n \tag{30}$$

}

subject to {

$$\sum_{i=1, i \neq j, x_{ij} \notin S}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{31}$$

$$\sum_{j=1, i \neq j, x_{ij} \notin S}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{32}$$

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{33}$$

$$0 \leq u_i \leq n - 1 \qquad 2 \leq i \leq n; \tag{34}$$

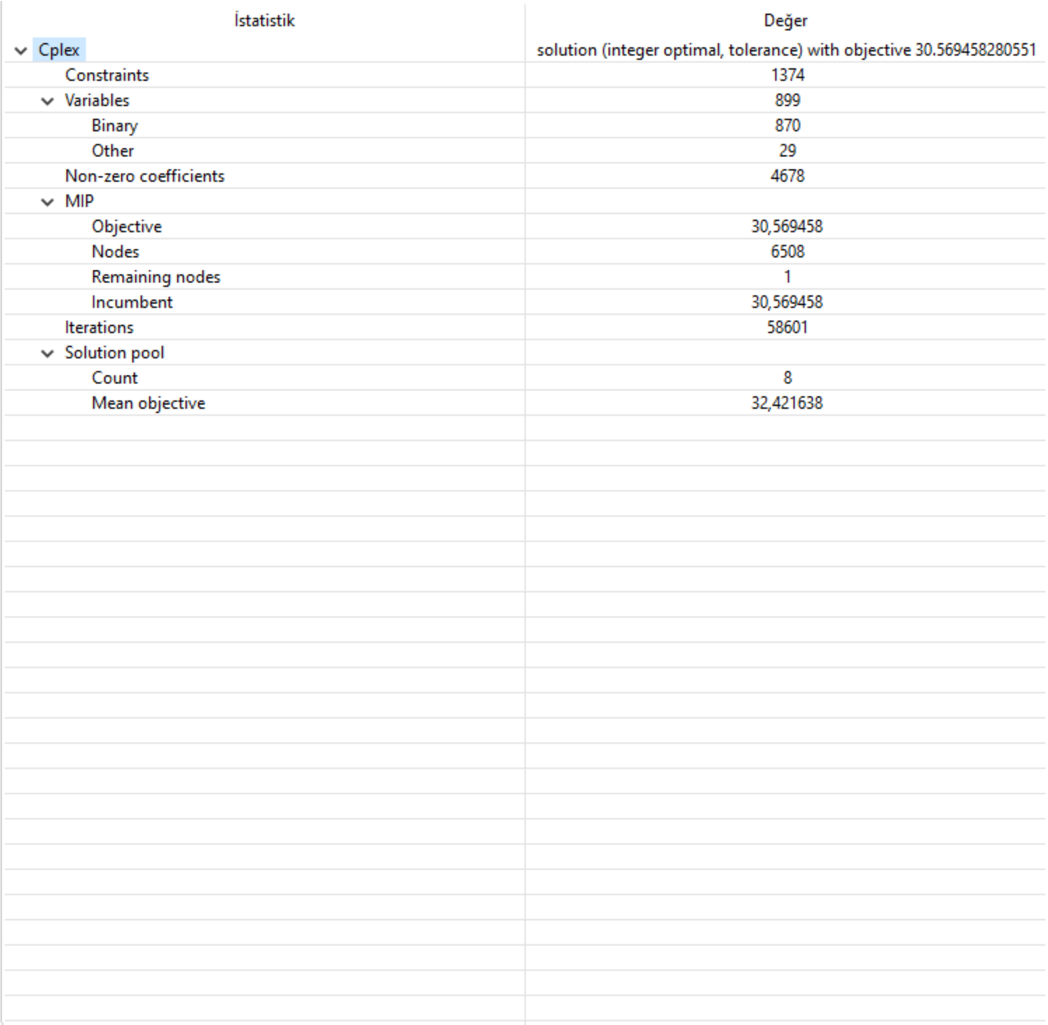
} [2]

We generated the model and we solved the problem using this model we mentioned.

4 Conclusion

4.1 Optimal Solution and Optimal Value

We solved the problem using our model and the data which is provided by the data.xlsx file. The model has found the optimal solution which is 30.569458280551 hours.



İstatistik	Değer
Cplex	solution (integer optimal, tolerance) with objective 30.569458280551
Constraints	1374
Variables	899
Binary	870
Other	29
Non-zero coefficients	4678
MIP	
Objective	30,569458
Nodes	6508
Remaining nodes	1
Incumbent	30,569458
Iterations	58601
Solution pool	
Count	8
Mean objective	32,421638

Figure 2: Optimal Solution

The CPLEX solution graph is provided below.

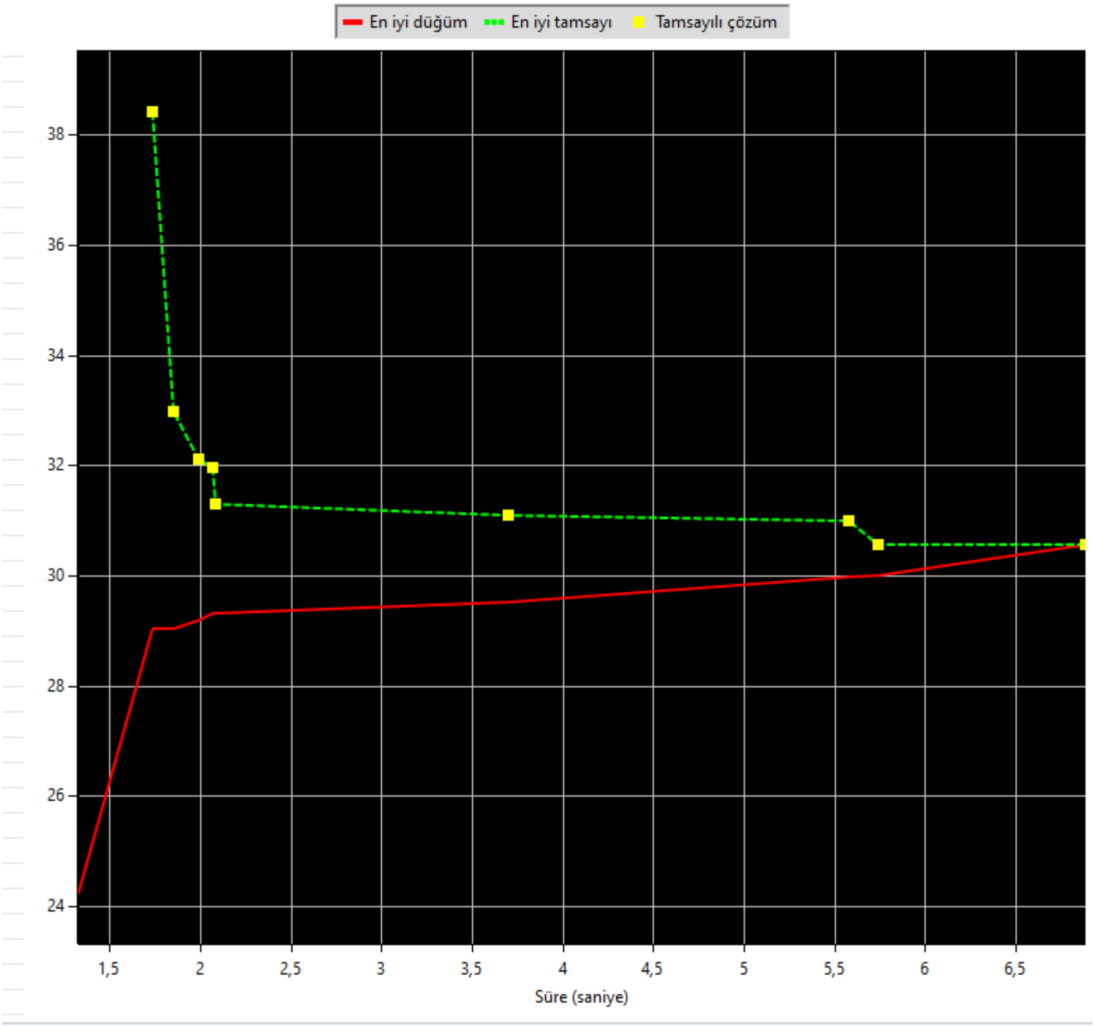


Figure 3: CPLEX Solution Graph

4.2 Simulated Path

According to the optimal solution mentioned above, the path of the cargo truck can be extracted using the **validPath** decision variable. It is shown in the table below.

Beginning Location	End Location	The Path Validity
L1	L23	1
L2	L22	1
L3	L5	1
L4	L6	1
L5	L4	1
L6	L28	1
L7	L1	1
L8	L19	1
L9	L29	1
L10	L9	1
L11	L20	1
L12	L27	1
L13	L2	1
L14	L7	1
L15	L30	1
L16	L11	1
L17	L14	1
L18	L24	1
L19	L12	1
L20	L10	1
L21	L26	1
L22	L3	1
L23	L13	1
L24	L16	1
L25	L17	1
L26	L25	1
L27	L15	1
L28	L18	1
L29	L8	1
L30	L21	1

Table 6: Unordered Optimal Path

The ordered path is arranged using the unordered path provided the table above and it is simulated using Desmos Graphing Tool. The simulated path which is obtained by the CIPLEX Optimization Studio is provided below.

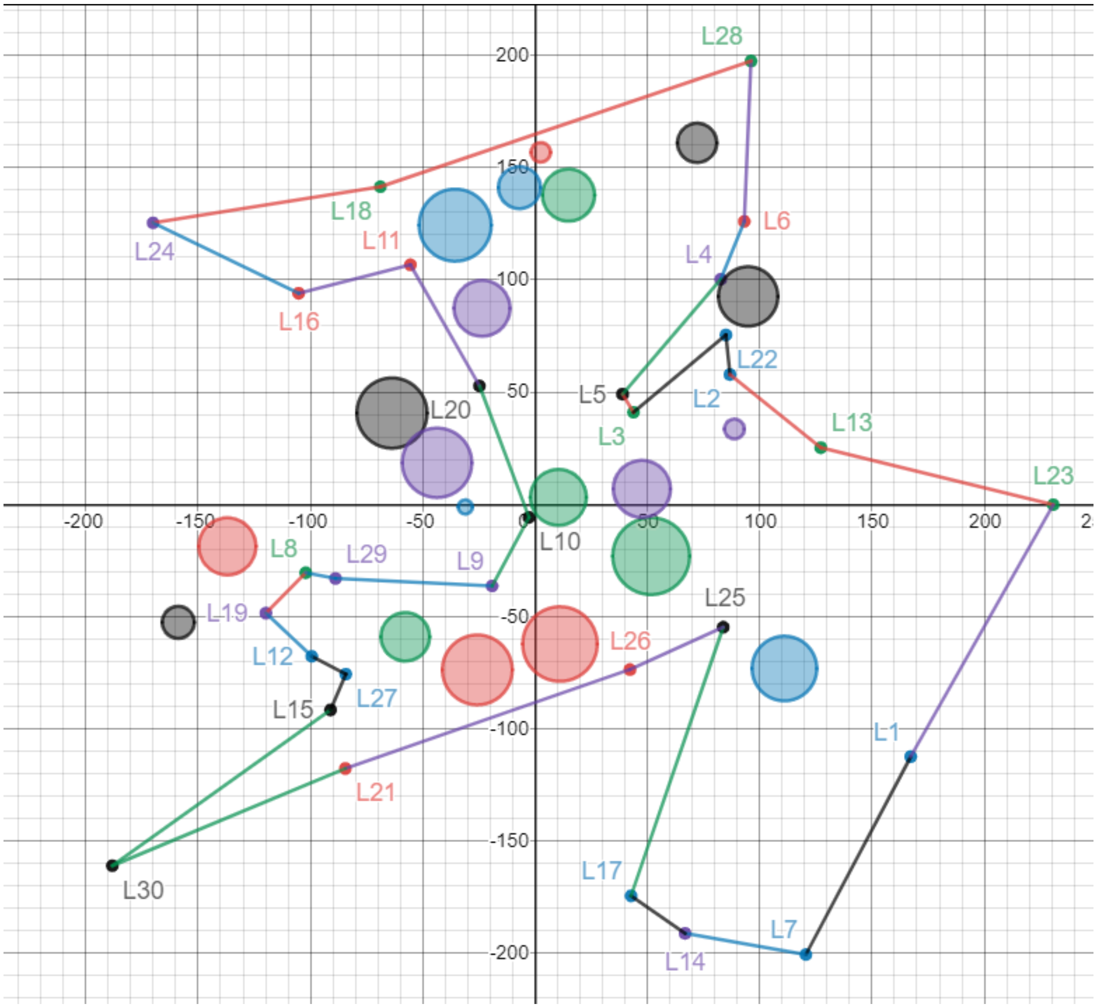


Figure 4: Optimal Path

As it can be seen that the cargo truck can complete its mission within 30.569458280551 hours and without encountering any regional storm following the path above.

References

[1] Claudemir Woche. *Modeling and solving the Traveling Salesman Problem with Python and Pyomo*. en-us. Educational. publisher: OPL. Feb. 2020. URL: `http://www.opl.ufc.br/post/tsp/` (visited on 08/02/2020).

[2] Hernan Caceres. *CPLEX Seminar - Getting started with CPLEX Studio (part 2) - YouTube*. English. Video. publisher: IBM. Feb. 2014. URL: `https://www.youtube.com/watch?v=URHDTzzDJak&t=1657s` (visited on 08/02/2020).