ISSN (e): 2250 – 3005 || Volume, 16 ||Issue, 2||February-2026 ||
International Journal of Computational Engineering Research (IJCER)

AI Apps using the Dijkstra Algorithm. Transport optimization based on the Dijkstra Algorithm for a Romanian transport company

Catalin Silviu Nutu
Constanta Maritime University
Constanta, Romania

ABSTRACT
This paper refers to a Romanian transport company moving goods inland, between its stores located in six different locations in the country. The Dijkstra Algorithm is one of the most useful and one of the most used computer subroutines when it comes to software applications and AI tools used in the transport business, in order to make the best decisions and optimizations of the transport routes. The algorithm was initially used to determine the shortest route between two different nodes of a distance graph. This paper presents some of these optimizations which can be made using this algorithm, and it shows how this algorithm can be employed to achieve best results based on certain assumptions, presented further on in this paper. This paper may also serve as a basis for further considerations and developments, and is useful for those wanting to develop AI or other optimization applications, based on the Dijkstra Algorithm, software applications which can be successfully used in the transport business.

Keywords: *Dijkstra Algorithm, vertices, paths, graphs, nodes, routes, transport, cargo*

Date of Submission: 24-01-2026 Date of acceptance: 06-02-2026

I. Introduction

This paper refers to a Romanian transport company moving goods inland, between its stores located in six different locations in the country. The Dijkstra Algorithm [2], [3], finds the shortest path from one vertex to all other vertices. It does so by repeatedly selecting the nearest unvisited vertex, using the matrix of distances between vertices, which in this case are the stores' locations.

The Dijkstra Algorithm is one of the most useful and one of the most used computer subroutines when it comes to software applications and AI tools used in the transport business, in order to make the best decisions and optimizations of the transport routes. The algorithm was initially used to determine the shortest route between two different nodes of a distance graph.

The Dijkstra Algorithm is used in this paper to mirror two different kinds of route optimization, from two different standpoints: from the point of view of the fastest routes between two different stores, and from the standpoint of the most fuel consumption efficient routes.

The Python [4] script used in this paper is taken from [1], as being one of the most elegant and one of the best implementations of the Dijkstra Algorithm to be found online, and employed for the optimization decisions made under the assumptions in this paper. This paper presents some of these optimizations which can be made using this algorithm, and it shows how this algorithm can be employed to achieve best results based on certain assumptions, assumptions presented further on in this paper.

This paper also shows, how an algorithm initially developed to solve a certain task, can be hijacked and employed in different adjacent areas and with completely different purposes than initially intended for.

The results, findings and optimization decisions in this paper may very well serve as a foundation for present and future AI application developments, for those interested to compete with the present software developers today, or for those interested to generate their own AI apps intended for all sort of transport optimizations, possibly within other various and completely different areas, where optimizations based on the Dijkstra Algorithm can be made.

Based on the results and findings in this paper, management of the transport companies can also perform various kinds of economic analyses and they can also make comparisons between various transport routes by analyzing their corresponding costs, but also by analyzing the related profit rates by comparing them one to another, this enabling them to make informed decisions based on actual data.

II. Data and data related assumptions used

II.1. Depending on the orders received in each region of Romania, the transport activity takes place between the six stores of the transport company, each of the stores being situated in the county seat of each of the six regions of Romania: the region of Dobrogea with the store located in Constanta (1/CT), the region of Muntenia, with the store located in the capital city of Romania, Bucharest (2/B), Moldova with one store located in Iasi (3/IS), Transylvania with two different stores, in Brasov (4/BV) and Cluj-Napoca (6/CJ) and the region of Banat, with one store located in Timisoara (TM).

II.2. The origin and the central store of the transport company is strategically located in Constanta because of its neighborhood to the Constanta Harbor

II.3. There are used two types of means of transportations: 20 tons heavy trucks to move the goods using low speed routes (corresponding to the bluelines graph scenario below) and 5 tons light trucks moving cargo at high speeds (corresponding to the redlines graph scenario below)

II.4. The medium fuel consumption for the light highspeed trucks is of 15 liters of diesel/100 km, and of 30 liters diesel/100 km for the heavy low speed trucks

II.5. The drivers' salaries and the overhead costs corresponding to one driver, amount to a value of 12.300 lei/month, which corresponds to a total distance per driver of about 12.300 km/driver/month, thus resulting an additional cost for the company of about 1 RON/km, additional to the fuel cost corresponding to the mileage

II.6. According to this paper's assumptions, the shortest distances correspond to the Bluelines scenario, and they are made using the heavy trucks. The bluelines routes correspond thus to the lowest transport costs, and are made at the medium speed of 50km/h. Hence, the most cost-effective scenario is the blueline scenario

II.7. The most time effective scenario corresponds to the Redlines scenario where the respective distances between stores are made at a medium speed of 100km/h using light trucks of 5 tons

II.8. In order to calculate the total cost C per transported ton of cargo, for a certain distance D, using a 20 tons heavy truck, which has a fuel consumption of 30 liters/ 100 km at a cost of 8 lei/diesel liter and with an additional cost of 1 leu/km corresponding to the driver's salary and the overhead costs, it results the value:

$$C = \frac{\frac{D}{100} * 30\ ltr * \frac{8lei}{ltr} + D * 1lei/km}{20\ tons} = D * 0.17 \qquad (1)$$

And thus, it results a cost/ton/km coefficient of 0.17, value which is used in the input data table for the Bluelines scenario.

III. Scenarios and scenario related assumptions used

There are two alternative scenarios and their related distances' graphs (the Bluelines one and the Redlines one) corresponding to two ways of moving goods between the stores (vertices or nodes of the graph):

- the Bluelines scenario corresponding to shorter distances between the regions' stores and to a medium speed of 50 km/h, using heavy trucks of 20 tons
- the Redlines scenario corresponding to longer distances between the regions' stores and to a medium speed of 100 km/h, using light trucks of 5 tons

These two different scenarios, can be visualized from the point of view of their related distances in the Figure 1 below.

The Bluelines scenario regards the cost optimization with respect to the routes and corresponds to the calculated cost price per km transported cargo ton of 0.17 lei as calculated before, the cargo being transported by road without any constraint regarding the transport time, as normal shipments, at a medium speed of 50 km/h. The Redlines scenario regards the time optimization with respect to the routes, and it assumes that the cargo is transported in fast road shipment, at a medium speed of 100 km/h.

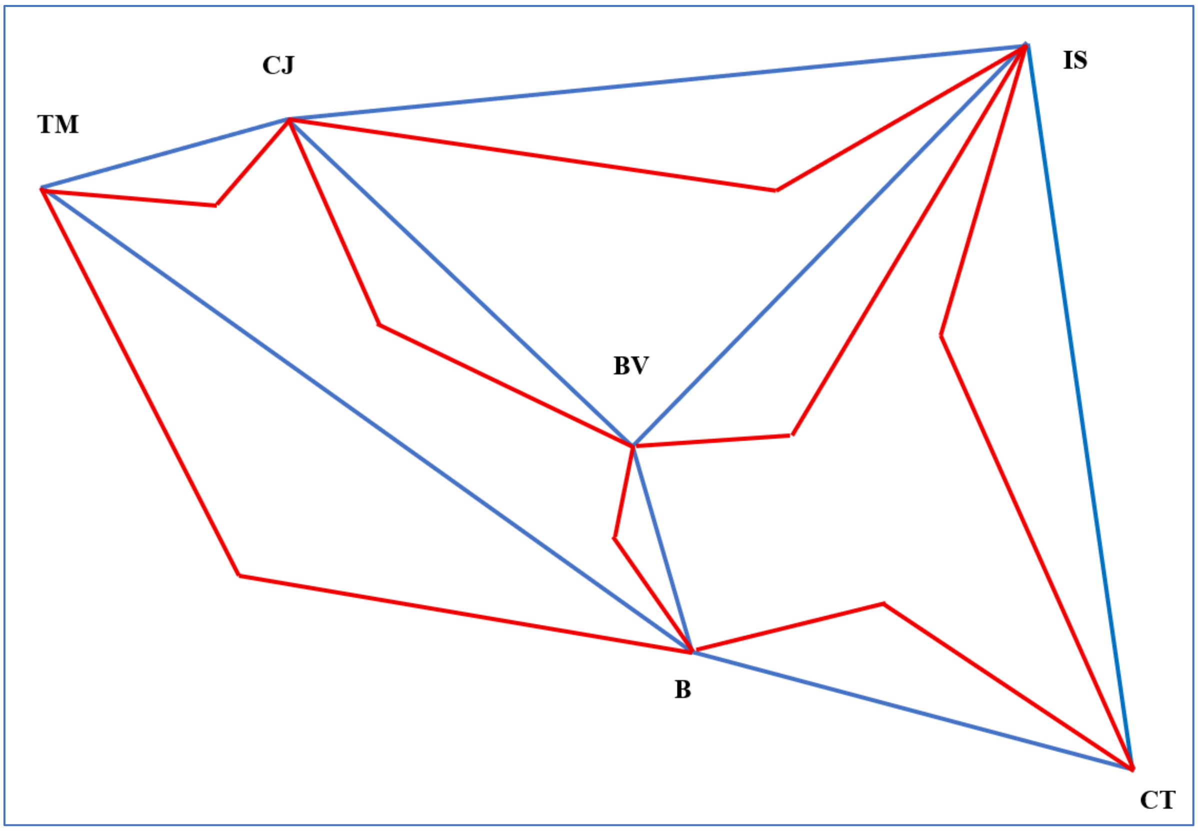

Figure 1 The Bluelines and Redlines scenarios

III.1. Distances between stores in the Bluelines graph scenario:

- Distance Constanta/ Bucharest: 225 km
- Distance Bucharest/ Brasov: 184 km
- Distance Brasov/ Cluj-Napoca: 270 km
- Distance Brasov/ Iasi: 308 km
- Distance Cluj/ Iasi: 392 km
- Distance Constanta/ Iasi: 402 km
- Distance Cluj-Napoca/ Timisoara: 312 km
- Distance Timisoara/ Bucharest: 549 km

III.2. Distances between stores in the Redlines graph scenario:

- Distance Constanta/ Bucharest: 240 km
- Distance Bucharest/ Brasov: 220 km
- Distance Brasov/ Cluj-Napoca: 310 km
- Distance Brasov/ Iasi: 340 km
- Distance Cluj/ Iasi: 430 km
- Distance Constanta/ Iasi: 440 km
- Distance Cluj-Napoca/ Timisoara: 350 km
- Distance Timisoara/ Bucharest: 570 km

III.3. The fuel consumption per 100km of the same type of trucks is considered to be the same, namely of 15 liters diesel/100 km for the light trucks of 5 tons, and of 30 liters/100 km for the heavy trucks of 20 tons

III.4. The optimization using the Dijkstra Algorithm in this paper is made from two different standpoints. On one hand, from the standpoint of cost, taking into account the costs incurred in the transport activity in the most cost-effective case of the Bluelines scenario, and on the other hand from the standpoint of the transport time, based on the Redlines scenario distances and on the respective average speed of 100km/h.

III.5. In the case of emergency transport of the cargo, the price the company receives for shipments exceeds by far the employed cost of the transport, and hence, the optimization is made only based on the shortest possible time

IV. Dijkstra Algorithm and input data used

As already stated before, the Dijkstra Algorithm will be used in the paper with the purpose to do two different optimizations, one cost optimization and one time optimization, each of the two being made separately for each of the both scenarios used in this paper. The first optimization refers to the most cost-effective routes between the stores' locations, and the second one refers to the most time effective routes between the stores. Based on the assumptions made in this paper, since each of the two scenarios comprises all the data necessary for the optimizations considered, there is no need to compare otherwise the data between these two scenarios used.

IV.1 Input data matrix for Bluelines scenario

According to the assumptions and the calculations made before, the input data matrix corresponding to the bluelines scenario, which is the most cost-effective scenario, corresponds to the right hand of the explanatory Table 1, below. In the cells of this input matrix for the Dijkstra Algorithm, one can find the calculated cost-effective values of the transported ton between the stores' locations, obtained by multiplying the distances with the calculated coefficient of 0.17.

Km	1 CT	2 B	3 IS	4 BV	5 TM	6 CJ	**Cost Coeff.**	**Cost/ ton**	1 CT	2 B	3 IS	4 BV	5 TM	6 CJ
1 CT	0	225	402	0	0	0	0.17	1 CT	0	38.25	68.34	0	0	0
2 B	225	0	0	184	549	0	0.17	2 B	38.25	0	0	31.28	93.33	0
3 IS	402	0	0	308	0	392	0.17	3 IS	68.34	0	0	52.36	0	66.64
4 BV	0	184	308	0	0	270	0.17	4 BV	0	31.28	52.36	0	0	45.9
5 TM	0	549	0	0	0	312	0.17	5 TM	0	93.33	0	0	0	53.04
6 CJ	0	0	392	270	312	0	0.17	6 CJ	0	0	66.64	45.9	53.04	0

Table 1 Input data for Dijkstra Algorithm in the Bluelines scenario

IV.2 Input data matrix for Redlines scenario

The input data matrix corresponding to the redlines scenario, which is the most time-effective scenario, corresponds to the right hand of the explanatory Table 2, below. In the cells of this input matrix for the Dijkstra Algorithm, one can find the values of shortest times corresponding to the transportation durations between the stores' locations, obtained by dividing the distances to the medium speed of 100 km/h.

Km	1 CT	2 B	3 IS	4 BV	5 TM	6 CJ	**Med. Speed**	**Hours**	1 CT	2 B	3 IS	4 BV	5 TM	6 CJ
1 CT	0	240	440	0	0	0	100	1 CT	0	2.4	4.4	0	0	0
2 B	240	0	0	220	570	0	100	2 B	2.4	0	0	2.2	5.7	0
3 IS	440	0	0	340	0	430	100	3 IS	4.4	0	0	3.4	0	4.3
4 BV	0	220	340	0	0	310	100	4 BV	0	2.2	3.4	0	0	3.1
5 TM	0	570	0	0	0	350	100	5 TM	0	5.7	0	0	0	3.5
6 CJ	0	0	430	310	350	0	100	6 CJ	0	0	4.3	3.1	3.5	0

Table 2 Input data for Dijkstra Algorithm in the Redlines scenario

V. Dijkstra Python scripts used

The Python scripts used for each of the both optimizations are based on [1], and they correspond to the implementation of the Dijkstra Algorithm, using the paper's assumptions and calculations previously made for the Bluelines and for the Redlines scenarios, as follows:

Python script corresponding to the Bluelines scenario	**Python script corresponding to the Redlines scenario**

import numpy as np
class Graph():
 	
    def __init__(self, vertices):
	import numpy as np
	class Graph():
	
	    def __init__(self, vertices):

```
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(node, "\t\t", dist[node])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initialize minimum distance for next node
        min = 1e7
        # Search not nearest vertex not in the
        # shortest path tree
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    # Function that implements Dijkstra's single source
    # shortest path algorithm for a graph represented
    # using adjacency matrix representation
    def dijkstra(self, src):

        dist = [1e7] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the
            # shortest path tree
            sptSet[u] = True
            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shortest path tree
            for v in range(self.V):
                if (self.graph[u][v] > 0 and
                   sptSet[v] == False and
                   dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

g = Graph(6)
g.graph = [[0, 38.25, 68.34, 0, 0, 0],
           [38.25, 0, 0, 31.28, 93.33, 0],
           [68.34, 0, 0, 52.36, 0, 66.94],
```

```
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(node, "\t\t", dist[node])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initialize minimum distance for next node
        min = 1e7
        # Search not nearest vertex not in the
        # shortest path tree
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    # Function that implements Dijkstra's single
source
    # shortest path algorithm for a graph represented
    # using adjacency matrix representation
    def dijkstra(self, src):

        dist = [1e7] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the
            # shortest path tree
            sptSet[u] = True
            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shortest path tree
            for v in range(self.V):
                if (self.graph[u][v] > 0 and
                   sptSet[v] == False and
                   dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

g = Graph(6)
g.graph = [[0, 2.4, 4.4, 0, 0, 0],
           [2.4, 0, 0, 2.2, 5.7, 0],
          [0, 31.28, 52.36, 0, 0, 45.9],
           [0, 93.33, 0, 0, 0, 53.04],
           [0, 0, 66.64, 45.9, 53.04, 0],
         ]
print(np.array(g.graph))
g.dijkstra(0)
g.dijkstra(1)
g.dijkstra(2)
g.dijkstra(3)
g.dijkstra(4)
g.dijkstra(5)

input()
```

```
          [4.4, 0, 0, 3.4, 0, 4.3],
          [0, 2.2, 3.4, 0, 0, 3.1],
           [0, 5.7, 0, 0, 0, 3.5],
           [0, 0, 4.3, 3.1, 3.5, 0],
         ]
print(np.array(g.graph))
g.dijkstra(0)
g.dijkstra(1)
g.dijkstra(2)
g.dijkstra(3)
g.dijkstra(4)
g.dijkstra(5)

input()
```

V. Results. Dijkstra Algorithm outputs

The Python scripts used, output in their first part the input data matrices used for each of the both optimizations, having as origin/source, the central store in Constanta City. Afterwards, the scripts output, on the left hand, the smallest costs, and on the right hand, the shortest times from each of the stores, whereas each of the stores is successively considered as being the source, for each of the both scenarios. In this way, one can have the optimizations of the routes depending on the store location where the respective truck starts the transportation of the cargo.

These outputs of the scripts above, used for each one of the two scenarios, are as follows:

Cost optimizations. The outputs of the Dijkstra Algorithm for Bluelines scenario

```
[[ 0.   38.25 68.34  0.    0.    0.  ]
 [38.25  0.    0.   31.28 93.33  0.  ]
 [68.34  0.    0.   52.36  0.   66.94]
 [ 0.   31.28 52.36  0.    0.   45.9 ]
 [ 0.   93.33  0.    0.    0.   53.04]
 [ 0.    0.   66.64 45.9  53.04  0.  ]]
Vertex   Distance from Source
0          0
1          38.25
2          68.34
3          69.53
4          131.57999999999998
5          115.43
Vertex   Distance from Source
0          38.25
1          0
2          83.64
3          31.28
4          93.33
5          77.18
Vertex   Distance from Source
0          68.34
1          83.64
2          0
3          52.36
4          119.97999999999999
5          66.94
Vertex   Distance from Source
0          69.53
1          31.28
2          52.36
3          0
```

Time optimizations. The outputs of the Dijkstra Algorithm for Redlines scenario

```
[[0.  2.4 4.4 0.  0.  0. ]
 [2.4 0.  0.  2.2 5.7 0. ]
 [4.4 0.  0.  3.4 0.  4.3]
 [0.  2.2 3.4 0.  0.  3.1]
 [0.  5.7 0.  0.  0.  3.5]
 [0.  0.  4.3 3.1 3.5 0. ]]
Vertex   Distance from Source
0          0
1          2.4
2          4.4
3          4.6
4          8.1
5          7.699999999999999
Vertex   Distance from Source
0          2.4
1          0
2          5.6
3          2.2
4          5.7
5          5.300000000000001
Vertex   Distance from Source
0          4.4
1          5.6
2          0
3          3.4
4          7.8
5          4.3
Vertex   Distance from Source
0          4.6
1          2.2
2          3.4
3          0
```

4	98.94	4	6.6
5	45.9	5	3.1
Vertex	Distance from Source	Vertex	Distance from Source
0	131.57999999999998	0	8.1
1	93.33	1	5.7
2	119.68	2	7.8
3	98.94	3	6.6
4	0	4	0
5	53.04	5	3.5
Vertex	Distance from Source	Vertex	Distance from Source
0	115.43	0	7.700000000000001
1	77.18	1	5.300000000000001
2	66.64	2	4.3
3	45.9	3	3.1
4	53.04	4	3.5
5	0	5	0

VI. Conclusions

Using the results corresponding to the outputs of the Python scripts above, one can conclude about the most cost effective or about the most time effective routes to be chosen in the activity of this Romanian transport company.

Based on paper's assumptions, the Bluelines scenario is also the most cost-effective scenario, and hence there is no need to compare results of the Dijkstra Algorithm between the Bluelines and Redlines scenarios when it is about costs, but one has only to use the results of the Dijkstra algorithm for the Bluelines scenario. In the very same way, because of the assumption of the paid market price seriously exceeds the costs incurred in the Redlines scenario, there is also no need for additional thoughts related to the choice of the Redlines scenario, when it is about time optimizations.

Hence, the analysis and the results in this paper are conditioned by the simplifying set of assumptions used, on which this paper is based upon. This has been made out of simplicity and clarity reasons for the presentation purposes of this paper.

One of the main goals of this paper is to show how a certain algorithm or other optimization tool, initially intended for a certain purpose, can be hijacked and can be used to do all sort of other adjacent different optimizations, than the optimizations and purposes initially intended for.

However, in the case of different assumptions, this is to say, in the case of different more complicated alternative scenarios, where the conclusions regarding the choice of a route/path over another are not so straightforward, there is need to compare results obtained from blue type and red type scenarios. Such much more complicated and exhaustive assumptions, disregarded in this paper, out of simplicity and clarity reasons, can be approached, using the recorded data and different computing techniques, able to compare them in real time, in order to do the best optimizations.

Moreover, the cost/ton/km coefficient has been calculated using the paper's assumptions, and thus resulted its constant value of 0.17 for each and every route possible. In the very same way, because of the medium speed of 100 km/h, there is a full proportionality between the data pertaining to both scenarios in the paper. This also explains why the ranking of the routes corresponding to both scenarios is the same. However, renouncing the simplifying assumptions of this paper and using actual data, it may be very possible that the cost/ton/km coefficients are all different, in the same way in which the medium speed may not always be of 100 km/h.

Based on the nowadays' huge computing power and based on the recording technology available today, able to record and store huge amounts of data, the research proposed in this paper can be substantially completed and can be made more elaborated, comprehensive and exhaustive and much more complex using such very large sets of recorded data. Hence, further developments of the research in this paper can be made by using and employing real-time data analysis, but also recording data during the transportation activity, and consequently analyzing this recorded data.

The main objective of this paper was, as already stated, to show how the Dijkstra Algorithm can be employed in order to make different choices when one has to make optimization decisions in the transport businesses.

At the same time, the results, findings and optimization decisions in this paper may very well serve as a foundation for present and further AI application developments, for those ones interested to compete with the present software developers today, and interested to generate their own AI apps. These apps may be intended for all sort of route optimizations, and possibly within other various and completely different areas than those where optimizations based on the Dijkstra Algorithm can be made.

Based on the results and findings in this paper, management of the transport companies can also perform various kinds of economic analyses and they can also make comparisons between various transport routes by analyzing their corresponding costs, but also by analyzing their related profit rates by comparing them one to another, and they can thus make informed decisions based on actual recorded or real time data.

Moreover, this recorded and real time data can be further on analyzed and used by employing various other AI tools and techniques, big data analysis and data mining techniques using all sort of other various algorithms available, and they can be thus integrated in a far more complex AI solution, useful for the respective transport businesses.

References

[1]. https://www.geeksforgeeks.org/python/python-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/

[2]. https://www.cse.ust.hk/~dekai/271/notes/L10/L10.pdf, Lecture 10 Dijkstra’s shortest path algorithm

[3]. Cormen Thomas H., Leiserson Charles E., “Introduction to Algorithms”, second edition, The MIT Press, Cambridge Massachusetts, London England https://www.cs.cmu.edu/afs/cs/academic/class/15451-s04/www/Lectures/shortestPaths.pdf

[4]. Halvorsen H.P.: “Python Programming”, ISBN:978-82-691106-4-7, 2020