Public transport route planning: Modified dijkstra's algorithm

Conference Paper · October 2017

DOI: 10.1109/UBMK.2017.8093444

3 authors, including:

Alican Bozyigit
Dokuz Eylul University
9 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Gazihan Alankuş
Izmir University of Economics
25 PUBLICATIONS **643** CITATIONS
SEE PROFILE

Public Transport Route Planning: Modified Dijkstra's Algorithm

Alican Bozyiğit
Dept. of Computer Science
Dokuz Eylul University
Izmir, Turkey
alican.bozyigit@deu.edu.tr

Gazihan Alankuş
Dept. of Computer Engineering
Izmir University of Economics
Izmir, Turkey
gazihan.alankus@ieu.edu.tr

Efendi Nasiboğlu
Dept. of Computer Science
Dokuz Eylul University
Izmir, Turkey
efendi.nasibov@deu.edu.tr

***Abstract*— Public transport applications, which aim to propose the ideal route to end users, have commonly been used by passengers. However, the ideal route for public transport varies depending on the preferences of users. The shortest path is preferred by most users as a primary criterion for the ideal route. According to our research, Dijkstra's Algorithm is mostly used in order to find shortest path. However, Dijkstra's Algorithm is not efficient for public transport route planning, because it ignores number of transfers and walking distances. Thus, in order to minimize these shortcomings, Dijkstra's Algorithm is modified by implementing penalty system in our study. Additionally, our modified algorithm is tested on the real world transport network of Izmir and compared with the results of Dijkstra's Algorithm. It is observed that our modified algorithm is quite efficient for route planning in the public transport network in terms of the number of transfers, distance of proposed route and walking distance.**

Keywords—Dijkstra's algorithm; public transport; route planning; shortest path; number of transfer; penalty system

I. INTRODUCTION

Public transport is a popular means of transportation in cities. Numerous internet and mobile applications aim to help users of public transport. These developed applications generally try to propose an ideal route for the user, who wants to go from one location to another in urban areas.

Users have a variety of preferences when it comes to the definition of an ideal route, which lead to various criteria for planning and evaluation of routes. Nasibov et al. have presented a study that consisted in 81 local residents from various locations in Izmir [1]. According to this study, twenty people out of eighty-one preferred the shortest path as primary criteria for the route selection in the survey.

There are many algorithms to find shortest path in the literature. Dijkstra's Algorithm is the most commonly used one in order to find the shortest path between two vertices [2]. The algorithm finds the shortest paths from the source vertex to all other vertices (single-source shortest path problem) in a short time, assuming that the weights of all edges are non-negative.

Although Dijkstra's Algorithm is efficient in terms of running time to determine the shortest path optimally, the proposed path is far from being the ideal route for most users; because Dijkstra's Algorithm does not take the number of transfers or the walking distances into account. These two shortcomings are important disadvantages for end-users.

In our study, "Modified Dijkstra's Algorithm" is evaluated by modifying Dijkstra's Algorithm, in order to overcome the two shortcomings (ignoring number of transfers and walking distance). A "small penalty cost" is added to the cost of the path in cases of stated shortcomings. Thus, the number of repeated walking and the number of transfers are minimized by slightly increasing the distance of the proposed path in the new route selection method.

Furthermore, Modified Dijkstra's Algorithm is tested on a real world transport network (Izmir, Turkey) with an experimental study. It is observed that the modified method has significantly better results than Dijkstra's Algorithm regarding number of transfers and walking distances.

II. RELATED WORK

Dijkstra's Algorithm is a popular method to find the shortest path in public transport planning [1]. In this study, a penalty system is implemented to Dijkstra's Algorithm in order to minimize number of transfers and walking distances. There are also other studies that modified the Dijkstra's Algorithm for public transport route planning. Wang et al. implemented an extra cost for each transfer in a public route selection algorithm [3]. They have designed public transport system as a new data model, and proposed a new shortest path algorithm in their study. Weaknesses of the shortest path algorithms were analyzed and Improved Dijkstra's Algorithm was proposed by Xu et al. [4]. Xiaoyong and Xueqin presented a heuristic algorithm that considered both transfer criterion and distance criterion for route selection [5]. Jian-lin explained that Dijkstra's Algorithm was not appropriate for route selection and proposed a new algorithm that was based on least number of transfers [6]. Wu and Hartley presented a study that used K-Shortest Paths Algorithm and proposed the optimal path among the K-Shortest Paths by taking the user preferences into consideration [7]. Ferreira et al. proposed a new advisor system based on the integration of various data sources and they used a Dijkstra's Algorithm implementation in this system [8]. Goel et al. presented a variant of Dijkstra's Algorithm by precomputing transfer patterns between hub nodes in order to be used for a multi-mode (i.e., bus, train and walk) transport network of Mumbai city, India [9]. Nguyen and MacDonald presented a new data model "Exploded

Graph" and they used Dijkstra's Algorithm in their new model to propose path with the least number of transfers [10]. Biswas introduced a new fuzzy condition factor in their network graph and modifies Dijkstra's Algorithm in this way [11]. In contrast, our approach is based on minimizing number of transfers and walking distance by slightly increasing distance of the path proposed by Dijkstra Algorithm. Thus, our modified algorithm strives to find an ideal path while keeping the path length short.

III. THE SHORTCOMINGS OF DIJKSTRA'S ALGORITHM

First; route selection with respect to the shortest path may include more transfers than other possible paths whose costs are nearly equal to the distance of the shortest path. For example, there is a directed weighted graph $G = (V, E)$ in Fig. 1. The distances between adjacent vertices are labeled on edges. Assume that from v_1 to v_3 there is the transit line 1; from v_3 to v_4 there is the transit line 2; from v_4 to v_5 there is the transit line 3; and lastly the transit line 4 traverses v_1, v_2 and v_5.

As clearly seen in Fig. 1, the shortest path from the source vertex v_1 to the target vertex v_5 is $P_1(v_1, v_5) = (v_1, v_3, v_4, v_5)$. The cost of P_1 is 6.

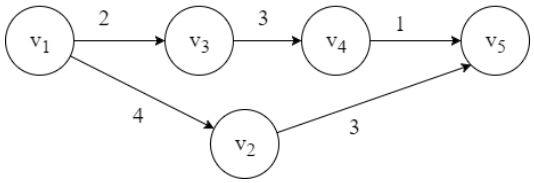

Fig. 1. A public transport network instance

By using Dijkstra's Algorithm, P_1 is proposed to the users as the ideal route from the v_1 to v_5; since P_1 is the shortest path. However, there is another path $P_2(v_1, v_5) = (v_1, v_2, v_5)$ from the source vertex v_1 to the target vertex v_5 where cost of this path is 7. Although the cost of P_2 is only one unit higher than the cost of the proposed path P_1, there is no need for any transfers on the path P_2. In order to go one less unit of distance, Dijkstra's Algorithm proposes the shortest path P_1 that includes two transfers. However, most users would prefer P_2 rather than P_1.

In addition, optimal route selection with respect to the shortest path may include long walking distances. One way to incorporate walking in Dijkstra's Algorithm is to add walking edges between nodes that are below a certain distance apart. Dijkstra's Algorithm tends to chain these edges. For instance, there is a directed weighted graph $G = (V, E)$ in Fig. 2.

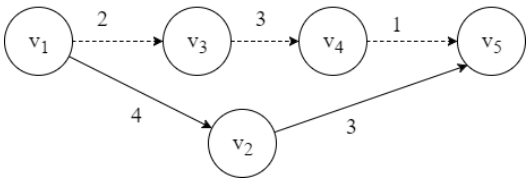

Fig. 2. A public transport network instance with dashed walking edges

Dashed straight lines illustrate walking paths between vertices. Solid straight lines illustrate transit lines between vertices in the figure. The shortest path from v_1 to v_5 is $P_1(v_1, v_5) = (v_1, v_3, v_4, v_5)$; which is a walking path. The cost of this path is 6. However, there is another path P_2=(v_1, v_2, v_5) using transit lines and its cost is 7. In order to go one less unit of distance, Dijkstra's Algorithm proposes the shortest path P_1 which would be a long walking path. However, most users would prefer P_2 rather than P_1.

IV. MODIFIED DIJKSTRA'S ALGORITHM

In order to avoid long walking distances and multiple transfers on the proposed path, a rule set is defined on Dijkstra's Algorithm in our study. The rule set of the penalty system is as follows;

- Assume that the current vertex is reached by walking. If the adjacent vertex of the current vertex is only reachable by walking again, the repeated walking will occur. In this case, the walking penalty is added to the alternative cost of the adjacent vertex to penalize the repeated walking.
- Assume that the current vertex is reached by walking. If the adjacent vertex of the current vertex is reachable by a transit line, the number of transfers and walking distance will be increased on the path. In this case, the transfer penalty is added to the alternative cost of the adjacent vertex to overcome numerous transfers and long walking distance on the proposed path.
- Assume that the current vertex is reached by a transit line. If its adjacent vertex is only reachable by a different transit line, the number of transfers will increase on the path. In this case, the transfer penalty is added to the alternative cost of the adjacent vertex.

By following the given rule set, the penalty cost from the current vertex to its adjacent vertex is calculated by using the Penalty Function. Pseudo code of the function is given in Algorithm 1.

Input: List of lines reached to the current vertex *currentLines*, list of lines reaches to the adjacent vertex from the current vertex *adjacentLines*
Output: Numeric value *penalty*, list of intersected lines *intersectedLines*
Function PenaltyFunction (*currentLines*, *adjacentLines*)

```
1.  Initialize a new list intersectedLines
2.  penalty ← 0
3.  if currentLines is null
4.      if adjacentLines is null
5.          penalty ← walkingPenalty
6.      else if adjacentLines is not null
7.          penalty ← transferPenalty
8.          intersectedLines ← adjacentLines
9.      end if
10. else if currentLines is not null and adjacentLines is not null
11.     for each line l in currentLines
12.         if adjacentLines contains l
13.         add l to intersectedLines
14.         end if
15.     end for
16.     if intersectedLines is null
17.         penalty ← transferPenalty
18.         intersectedLines ← adjacentLines
19.     end if
20. end if
21. return penalty, intersectedLines
```

Algorithm 1. Penalty Function

Additionally, the transit line matrix is used for storing transit lines that connect adjacent vertices. The transit line matrix L is a two-dimensional array which is $|V| \times |V|$ and each cell of the matrix contains a set of transit lines between the related adjacent vertices. The transit line set of the edge (v_i, v_j) is stated by $l_{i,j}$, where $v_i, v_j \in V$, $(v_i, v_j) \in E$ and $l_{i,j}$ is element of L. The transit line matrix L is stated as follows;

$$L_{nxn} = \begin{bmatrix} \emptyset & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots \\ l_{n,1} & \cdots & \emptyset \end{bmatrix}$$

Thus, the new penalty function is implemented to Dijkstra's Algorithm. This function is used for adding a penalty to alternative cost if it is needed. The transit line matrix L is taken as an input parameter in this new method in order to obtain the lines between the adjacent vertices. Lastly, an array list is used to store the data of the lines that reach to the vertices. The pseudo code of the new method is given in Algorithm 2.

Input: The weighted adjacency matrix of the graph $W[n,n]$, the transit line matrix of the graph $L[n,n]$ and the source vertex s
Output: An array that stores cost of the vertices $cost[n]$, N sized array list that stores the list of vertices which is path from the source to the related vertex $path[n]$
Function ModifiedDijkstra'sAlgorithm(W, L, s)
1. Initialize N sized arrays $cost\ [n], path\ [n], lines[n]$
2. Initialize vertex list $unvisited$
3. **for each** vertex v in the vertex set
4. $cost[v] \leftarrow \infty$
5. add v to $unvisited$
6. **end for**
7. $cost[s] \leftarrow 0$
8. **while** unvisited is not empty
9. $current \leftarrow v$ where v has min cost in $unvisited$
10. remove $current$ from $unvisited$
11. **for each** vertex $adjacent$ to $current$
12. initialize new line list $insersectedLines$
13. $penalty, insersectedLines \leftarrow$ PenaltyFunction$(lines[current], L[current, adjacent])$
14. $alternativeCost \leftarrow cost[current] + W[current, adjacent] + penalty$
15. **if** $alternativecost < cost[adjacent]$
16. $cost[adjacent] \leftarrow alternativeCost$
17. $path[adjacent] \leftarrow path[current] + current$
18. $lines[adjacent] \leftarrow intersectedLines$
19. **end if**
20. **end for**
21. **end while**
22. **return** $path, cost$

Algorithm 2. Modified Dijkstra's Algorithm

In this new method, walking distance and the numbers of transit lines used on the path are minimized. However, in proportion to this, the distance of the path is slightly increased. Thus, it can be stated that, an ideal route selection with respect to the shortest path is calculated by this proposed method.

V. APPLICATION

To visually compare the results of Modified Dijkstra's Algorithm and Dijkstra's Algorithm, a visual form application is developed in this study. Additionally, the application includes the GMap.NET (map component) [12]. The application is illustrated in Fig. 3.

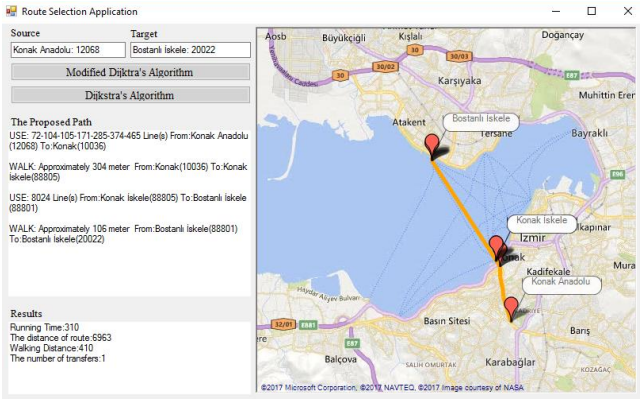

Fig. 3. The route selection application

The application uses the real-world transport network of Izmir (Turkey) as the dataset and this dataset is obtained from the web page of ESHOT [13]. Public transport network of İzmir has four transit modes as; bus, metro, ferry and light railway. There are 7,704 stations (vertices) in the public transport network of İzmir that includes ferry stations, metro stations, light railway stops, and mostly the bus stops. There are 8,837 transit line connections (edges) between all of these stations. Additionally, there are 34,630 walking connections between these stations. The walking distance is taken as 300 meters as a constant in this study.

Modified Dijkstra's Algorithm and Dijkstra's Algorithm tested on 2000 source-target pairs that are stations of the public transport network of İzmir. For each source-target pair, two routes (one for each method) are calculated. Additionally, costs (the distance of route, the number of transfers, walking distance) of these routes are calculated. The average results of both algorithms are presented in Table I.

TABLE I. THE AVERAGE RESULTS OF THE ALGORITHMS

Algorithm	***Average Running Time***	***Average Distance of Routes***	***Average Number of Transfers***	***Average Walking Distance***
Dijkstra's Algorithm	242 ms	30.974 km	4.72	0.482 km
Modified Dijkstra's Algorithm	309 ms	33.598 km	2.48	0.396 km

The average running times of both algorithms are remarkably fast enough to be used in the public transport applications. Although Modified Dijkstra's Algorithm has the worse running time, its average running time is only 0.3 second.

On the other hand, it is observed that Modified Dijkstra's Algorithm is significantly better in terms of average number of transfers. Its average number of transfers is %47 less than the result of Dijkstra's Algorithm. Additionally, it is slightly better in terms of average walking distance. Its average walking distance is 17% less than the result of Dijkstra's Algorithm. Although, this algorithm is a modified in order to minimize

number of transfers and long distance walks, it is close to Dijkstra's Algorithm in terms of average distance. Its average distance is 8% greater than the result of Dijkstra's Algorithm.

For instance, from station 40015 (Buca Belediye Sarayi) to station 10255 (Ucyol Metro);

- Dijkstra's Algorithm proposes the route that is 40015→12068→10612→10255 (Buca Belediye Sarayi→Konak Anadolu→Osman Oksuz Parki→Ucyol Metro) in our study. This route includes two transfers and the distance of this route is 4.249 km.
- Modified Dijkstra's Algorithm proposes the route that is 40015→10255 (Buca Belediye Sarayi→Ucyol Metro) in our study. There is no need to make transfer on this route and the distance of this route is 4.432 km.

Dijkstra's Algorithm proposes the shortest path that includes two transfers. However, most users do not prefer making two transfers in order to travel less 0.283 km.

VI. CONCLUSION

Dijkstra's Algorithm is most commonly used algorithm to find shortest path. However, it is far from being ideal algorithm in public transport, because this algorithm does not take the number of transfers or the walking distances into account. Firstly, the shortcomings of Dijkstra's Algorithm are determined. In order to minimize these shortcomings, the rule set of the penalty system is defined. By following the given rule set, penalty function is developed. This penalty function is implemented to Dijkstra's Algorithm. Thus, a public transport route planning method is proposed by modifying Dijkstra's Algorithm. Modified Dijkstra's Algorithm is tested on the real world transport network of Izmir and it is compared with the results of Dijkstra's Algorithm. It is observed that Modified Dijkstra's Algorithm is significantly better than Dijkstra's Algorithm regarding the number of transfers and the walking distance. It can be stated that, an ideal route planning with respect to the shortest path is proposed by using our Modified Dijkstra's Algorithm.

ACKNOWLEDGMENT

This work is part of a thesis submitted by the first author to Izmir University of Economics in partial fulfillment of the requirements for the degree of Master of Science. This study was also partially supported under the project number 113E535 by The Scientific and Technological Research Council of Turkey (TUBITAK).

REFERENCES

[1] E. Nasiboğlu, A. Bozyiğit and Y. Diker, "Analysis and evaluation methodology for route planning applications in public transportation", *in Application of Information and Communication Technologies (AICT)*, Russia, 2015, pp. 477-481.

[2] E. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269-271, 1959.

[3] H. Wang, M. Hu and W. Xiao, "A new public transportation data model and shortest-Path algorithms", in *Informatics in Control, Automation and Robotics*, 2010, pp. 456-459.

[4] X. Xu, Y. Yin, and L. Liu, "Improved Dijkstra's algorithm and its application in intelligent transportation system", *Journal of Residuals Science and Technology*, vol. 13, no. 7, 2016.

[5] X. Yan and X. Niu, "Study on heuristic algorithm for public transport network multi-path selection", *Urban Transport of China*, vol. 3, 2005.

[6] W. A. N. G. Jian-lin, "The public transportation optimum route algorithm based on the least transfer [J]", *Economic Geography*, vol. 5, pp. 673-676, 2005.

[7] Q. Wu and J. Hartley, "Using k-shortest paths algorithms to accommodate user preferences in the optimization of public transport travel*"* in *Applications of Advanced Technologies in Transportation Engineering*, 2004, pp. 181-186.

[8] J. C. Ferreira, P. Filipe and A. Silva, "Multi-modal transportation advisor system", in *Integrated and Sustainable Transportation System (FISTS)*, 2011, pp. 388-393.

[9] N. Goel, N. R. Velaga, P. Vedagiri and T. V. Mathew, "Optimal routing algorithm for large scale multi-modal transit networks", 2017.

[10] D, Nguyen and T. MacDonald, "TDplanner: Public transport planning system for mobile devices.", Department of Mathematics and Computer Science, Suffolk University, Boston, Tech. Rep., 2010.

[11] S. S. Biswas, "Fuzzy Real Time Dijkstra's Algorithm", *International Journal of Computational Intelligence Research*, vol.13, no. 4, pp. 631-640, 2017.

[12] "GMap.NET - Great Maps for Windows Forms Presentation*", CodePlex*, 2017. [Online]. Available: https://greatmaps.codeplex.com. [Accessed: 02- Jul- 2017].

[13] "Eshot Genel Müdürlüğü Resmi Web Sitesi*", Eshot.gov.tr*, 2017. [online] Available at: http://www.eshot.gov.tr [Accessed: 02-Jul-2017].