A New Public Transportation Data Model And Shortest-Path Algorithms

Hongmei Wang, Ming Hu, Wei Xiao
School Of Computer Science and Engineering
Changchun University Of Technology
ChangChun, JiLin, China
Wanghm0326@gmail.com

Hongmei Wang
School Of Computer Science and Technology
JiLin University
ChangChun, JiLin, China
Wanghm0326@gmail.com

***Abstract*—By studying the best-path problem for public transportation systems, we found that the nature of transfer is that it requires extra costs from an edge to its adjacent edge. Therefore, we propose the notion of direct/indirect adjacent edges in weighted directed multigraphs and extend the notion of path to the line. We use the direct/indirect adjacent edges weighted directed multigraph as a public transportation data model and improve the storage of an adjacency matrix. We introduce the space storage structure, the matrix VL, in order to store the scattered information of transfer in indirect adjacent edges lists. Thus, we solve the problem of complex network graphs' storage and design a new shortest path algorithm to solve transit problem based on the data model we propose in this paper. Algorithm analysis exhibits that the data model and the algorithm we propose are prior to a simple graph based on the Dijkstra's algorithm in terms of time and space.**

***Index Terms*—direct adjacent edges, indirect adjacent edges, the vertex-line matrix, shortest-path algorithms, performance analysis**

I. INTRODUCTION

The public transportation data model can be used as a basis to solve the problem of computing public transit shortest paths. Passengers need to change lines if they can not arrive directly. Transfers are categorized as original place transfers and neighborhood transfers. For original place transfers, passengers may consider factors such as waiting time, ticket costs and other transfer costs. For the neighborhood transfer, passengers may consider walking time and so on. To accurately represent line transfer and transfer costs, then find the exactly shortest path is crucial for the public transportation data model [1].

Presently, there are several solutions to this problem. One method is to abstract nearest stop nodes into one node in a network graph. It considers standard transfer times [2][3].The result is an approximate shortest path. A second method is to transform a complex network graph into a simple network graph, and then use the Dijkstra's algorithm to solve the problem. After transformation, the scale of a simple network graph would be a hundred times larger than a normal network graph and the time complexity of using the Dijkstra's algorithm is unsatisfactory. The final method is to use the neighborhood transfer to represent different stop nodes and build a transfer junction [4][10]. Then convert network costs, such as walking distance, costs, the transfer times, waiting time and extra costs into time directly stored on the corresponding arc and regarded as a weight on the arc [4].The model is too large and too difficult to modify in this algorithm. And then there is a method to introduce an arrival matrix and a least transfer matrix to describe the relation of the transfer between nodes [6]. It increases the algorithm's space costs and time costs of maintaining auxiliary matrices.

In this paper, we propose the direct/indirect adjacent edges weighted directed multigraph used as public transportation data model and improve the storage of an adjacency matrix [8]. We use the vertex-line matrix(the matrix VL) to store direct/indirect adjacent edges weighted directed multigraph and design a shortest path algorithm based on the Dijkstra's algorithm, A^* algorithm, and depth first search strategy(DFS). Finally, we analyze the algorithm in time and space and compare it with the Dijkstra's algorithm.

II. DIRECT/INDIRECT ADJACENT EDGES WEIGHTED DIRECTED MULTIGRAPHS

A. Related Definition

Definition 1 Adjacent Edges In a weighted directed graph $G = (V, E)$, an edge $\langle v_j, v_k \rangle$ is adjacent to an edge $\langle v_i, v_j \rangle$ if and only if $v_i, v_j, v_k \in V$, $\langle v_i, v_j \rangle$ and $\langle v_j, v_k \rangle \in E$.

Definition 2 Direct Adjacent Edges/Indirect Adjacent Edges If there is no cost from an edge to its adjacent edge, the adjacent edge is called the edge's direct adjacent edge. Otherwise, it's called the edge's indirect adjacent edge. The weight of an indirect adjacent edge represents the cost from an edge to its adjacent edge.

Definition 3 Line In a weighted directed graph $G = (V, E)$, a line from v_p to v_q is the vertex sequence $v_p = v_{i0}, v_{i1}, \cdots, v_{im} = v_q$, where $\langle v_{ij-1}, v_{ij} \rangle$ and $\langle v_{ij}, v_{ij+1} \rangle \in E$ and an edge $\langle v_{ij}, v_{ij+1} \rangle$ is the direct adjacent edge of an edge $\langle v_{ij-1}, v_{ij} \rangle (1 \le j \le m\text{-}1)$.

B. Public Transportation Data Model

Public transportation systems consist of different transit lines. There are several docking sites on every transit line. Every docking site may belong to multiple transit lines. Passengers can transfer to other lines on some docking sites, but require transfer costs. Public transportation systems are abstract as direct/indirect adjacent edges weighted directed multigraphs, where nodes in a graph represent docking sites, paths in a graph represent public transit lines, direct adjacent edges represent no transferring and its weight is expressed as line(0), and indirect adjacent edges represent transferring and its weight is expressed as line j(line i, node k, transfer costs).

978-1-4244-5194-4/10/$26.00 ©2010 IEEE

For instance, $l_1(0)$ implies that a user goes to a node along the transit line l_1 without transferring. $l_1(l_2, v_3, 1)$ implies that a user transfers from l_2 to l_1 at node v_3 and it has a cost of 1. Figure 1 illustrates a sample of public transportation systems corresponding to direct/indirect adjacent edges weighted directed multigraphs.

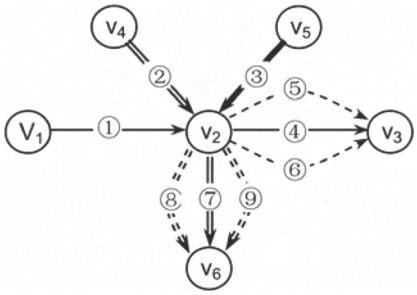

Figure 1. Public transportation systems abstracted as direct/indirect adjacent edges weighted directed mutigraphs.

①. $l_1(0)$ ②. $l_2(0)$ ③. $l_3(0)$ ④. $l_1(0)$ ⑤. $l_1(l_2, v_3, 1)$

⑥. $l_1(l_3, v_3, 2)$ ⑦. $l_2(0)$ ⑧. $l_2(l_1, v_6, 3)$ ⑨. $l_2(l_3, v_6, 1)$

There are three lines l_1, l_2 and l_3. l_1 passes v_1, v_2 and v_3. l_2 passes v_4, v_2 and v_6. l_3 passes v_5 and v_2. l_1 passes v_2 toward v_3 without any costs, which means edge ④ is a direct adjacent edge of edge①. At v_2, we can transfer from l_2 or l_3 to l_1, which means edge ⑤ and edge ⑥ are an indirect adjacent edge of edge ①.

Obviously, the number of indirect adjacent edges shows geometrical progression to grow, along with lines which pass a node increasing in the direct/ indirect adjacent edges weighted directed multigraphs. We propose the storage of matrix VL to solve the problem efficiency.

III. MATRIX VL

Definition 4 Matrix VL There are n nodes and m lines in the direct/indirect adjacent edges weighted directed multigraphs. The vertex-line matrix (matrix VL) of a graph is a n × m matrix, where the i^{th} row of the matrix represents the node v_i, the j^{th} column of the matrix represents the line l_j, the VL_{ij} of the matrix VL is defined as:

(1) VL_{ij} =0 if line l_j doesn't pass node v_i.

(2) The node structure of VL_{ij} is shown in Figure 2 if line l_j passes node v_i.

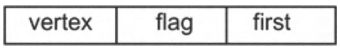

Figure 2. The node structure of the matrix VL's elements.

where **vertex**: stores node v_i's next node if line l_j passes node v_i. If node v_i is the end of line l_j, vertex = -1.

flag: flag=1 if there is indirect adjacent edges when the line l_j passes node v_i, otherwise flag=0.

first: the indirect adjacent edges make up a simply-linked linear list if there are indirect adjacent edges when line l_j passes node v_i. The simply-linked linear list is called the indirect adjacent edges list. The first is the header pointer of the indirect adjacent edges list. The first is the null pointer if flag=0.

The node structure of indirect adjacent edges list is shown in Figure 3.

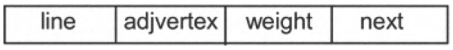

Figure 3. The node structure of the indirect adjacent edges list.

where **line** : stores the line which passes the adjacent edge.

adjvertex: stores the next node if the line passes the adjacent edge.

weight: stores the transfer costs of the adjacent edge.

next: points to the next node of the adjacent edge.

A graph with the storage of the matrix VL is shown in Figure 4.

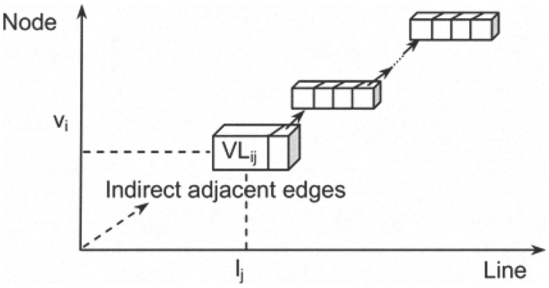

Figure 4. A graph with the storage of the matrix VL.

The matrix VL is shown in Figure 5 corresponding to the direct/indirect adjacent edges weighted directed multigraph shown in Figure 1.

	l_1	l_2	l_3
v_1	(v_2, 0, ∧)	0	0
v_2	(v_3, 1, ptr1)	(v_6, 1, ptr2)	(-1, 1, ptr3)
v_3	(-1, 0, ∧)	0	0
v_4	0	(v_2, 0, ∧)	0
v_5	0	0	(v_2, 0, ∧)
v_6	0	(-1, 0, ∧)	0

Figure 5. The matrix VL corresponding to Figure 1.

In Figure 5, the ptr1 pointes to the indirect adjacent edges list which line l_1 passing node v_2 is stored in. The ptr2 pointes to the indirect adjacent edges list which line l_2 passing node v_2 is stored in. The ptr3 pointes to the indirect adjacent edges list which line l_3 passing node v_2 is stored in. The indirect adjacent edges list of the matrix VL's elements are shown in Figure 6.

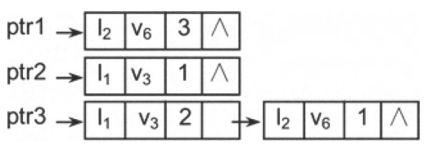

Figure 6. The indirect adjacent edges list of the matrix VL's elements.

IV. SHORTEST-PATH ALGORITHMS

It is not important to compute literally the shortest path from the site A to the site B. Passengers' main concern is whether to arrive directly or not. With the continuous improvement of public transport systems, usually, passengers only need to transfer once per trip.

In this paper, we propose a new transit line query algorithm based on the Dijkstra's algorithm and DFS. The algorithm outputs all direct lines from site A to site B and sorts according to the number of passed nodes. If there is no directly line, the algorithm outputs paths that require a single transfer, and sorts according to transfer costs and the number of passed nodes. The algorithm description is shown below:

Step 1 initialize the set SL with the lines which pass the node A in the matrix VL.

Step 2 initialize the set TL with the lines which pass the node B in the matrix VL.

Step 3 there are lines from the node A to the node B if $SL \cap TL \neq \Phi$,and then perform the following operation:

Step 3.1 get nodes which every line passes from the node A to the node B in the set $SL \cap TL$ and the number of nodes with DFS in the matrix VL.

Step 3.2 sort the lines according to the number of nodes in the set $SL \cap TL$.

Step 3.3 output arrival information and exit.

Step 4 initialize the set SV with the nodes which every line passes from the node A in the matrix VL in the set SL.

Step 5 initialize the set TV with the nodes which every line passes from the node A in the matrix VL in the set TL.

Step 6 if $SV \cap TV \neq \Phi$, which means transfer one time to arrive from the node A to the node B, perform the following operation:

Step 6.1 get (line i, node j, line k, weight) if line $i \in SL$, line $k \in TL$, node $j \in SV \cap TV$ and the line i is in the node j' s indirect adjacent edges list.

Step 6.2 get countA with the number of passed nodes from the node A to the node j

Step 6.3 get countB with the number of passed nodes from the node j to the node B

Step 6.4 insert (line i, node j, line k, weight, countA, countB) into the Fibonacci heap [9] with the transfer costs as the main key and the number of passed nodes as the second key.

Step 7 output every node in turn in the priority queue.

V. PERFORMANCE ANALYSIS

A. Related Definition

The matrix VL requires $O(n \times m \times k)$ space if the scale of the direct/indirect adjacent edges weighted directed mutigraph's nodes is n, the scale of the lines is m and the average number of nodes is k in the indirect adjacent edges lists. For the actual public transportation system, k can be regarded as a constant coefficient because $n>m$ and $m>>k$. So the matrix VL requires $O(n \times m)$ space. Compared with the Dijkstra's algorithm which requires $O(n^2)$ space, it reduces the space complexity.

The matrix VL will be a sparse matrix when the scale of nodes n and the scale of lines m are very large. In order to solve the problem of the sparse matrix VL, it is efficient to transform the matrix VL into an orthogonal list.

B. Time Complexity Analysis

In Step 1, the algorithm needs traverse the row where the node A is in the matrix VL. It requires $O(m)$ time.

In Step 2, the algorithm needs traverse the row where the node B is in the matrix VL. It requires $O(m)$ time.

In Step 3, the algorithm needs traverse m_1 columns in the matrix VL if $|SL \cap TL| = m_1$. It requires $O(m_1 \times n')$ time because the nodes that lines pass are stored in the matrix VL.

The algorithm needs search m_2 lines in Step 4 and m_3 lines in Step 5 if $|SL| = m_2$ and $|TL| = m_3$. It requires $O(m' \times n')$ time if the average number of lines which pass a node and there are n' nodes on average per line at most.

In Step 6.1, the algorithm needs search the transfer costs in n_1 indirect adjacent edges lists if $|SV \cap TV| = n_1$.It requires $O(n_1 \times k)$ time. In Step 6.2 and Step 6.3, the algorithm computes the number of nodes in the matrix VL. It requires $O(n_1 \times n')$ time. In Step 6.4, the algorithm inserts the output into the Fibonacci heap. It requires $O(n_1 \times \log_2 n_1)$ time. So the algorithm requires $O(n_1 \times (k + n' + \log_2 n_1))$ time in Step 6.

In Step 7, the algorithm outputs every node in the priority queue. It requires $O(n_1)$ time.

To sum up, the algorithm requires $O(m + m_1 \times n')$ time to arrive directly from node A to node B. The algorithm requires $O(m + n_1 \times (k + n' + \log_2 n_1) + n_1)$ time if the trip requires one transfer. For the actual public transportation systems, m', n', k, m_1 and n_1 can be regarded as a constant coefficient because they are few in number and on average of no more than one number. So the algorithm requires $O(m)$ time. Compared with the Dijkstra's algorithm which requires $O(n^2)$ time, it improves the time performance. In this paper, the algorithm we propose can find all direct lines and lines requiring one transfer.

VI. CONCLUSIONS

The Dijkstra's algorithm is a search algorithm to compute the shortest path between two given nodes on a network topology, but it is unfit for the real time computation on a large transit network. The A^* algorithm is the classical heuristic search algorithm about the problem of the shortest path [2][5][10]. Unfortunately, the public transportation data model needs the geometry information of every vertex and every arc, thus the particularity of the transit problem minimizes the usage value of the heuristic function. In this paper we propose a direct/ indirect adjacent edges weighted directed multigraph and use the matrix VL to solve the problem of complex network graphs' storage. The algorithm has amazing practical value.

REFERENCES

[1] Changli LIU: Best-path planning for public transportation systems[C]. Proc of the 5th International IEEE Conference on Intelligent Transportation Systems. Singapore, 2002. 834-839J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Hong Liang, Xiaoqun Yuan, Rui Liu: Novel model and realization of public transport route inquiring system[J]. Computer Engineering and Applications,2007,43(3):234-238K. Elissa, "Title of paper if known," unpublished.

[3] Xiaoping Yu, Guodong Yang, Fengyan Wang, Huiping Xu,: Design and Implementation of Urban Public Transport Inquiry System[J]. Journal of Changchun Post and Telecommunication Institute,2005,23(6):675-678.Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[4] Hui Wen, Yuefeng Liu, Jianghua Zheng, Lie Yan: Study on Data Model for Public Transport Networks Based on Time Chain[J]. Geography and Geo-Information Science,2005,21(3):35-38

[5] Junjie Li, Hanli Zhang: A new solution to the k-shortest paths problem and its application in wavelength routed optical networks[J]. Photonic Network Communications, 2006,12(3):269-284

[6] Li Wang, Wenquan Li: Best-routing algorithm for public transportation systems[J]. Journal of Southeast University (Natural Science Edition),2004,34(2):264-267

[7] Qing Chang: Vehicle navigation and positioning method and its application[M]. Beijing:China Machine Press,2005.

[8] Hongmei Wang, Ming Hu, Tao Wang: Data Structure(C++)[M]. Tsingua University Press.2005

[9] Fredman, M.L., Tarjan, R.E: Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms[J]. Journal of the ACM, 1987, 34 (3) :596-615

[10] Jia Liu, Shaofang Xia, Yanan Lv,Lichao Chen: Algorithm for solving K-shortest paths problem in complicated network[J]. Journal of Computer Applications, 2008,28(4):951-953