A New Implementation of Dijkstra's Algorithm on Urban Rail Transit Network

Jimeng Tang [a], Quanxin Sun[b] and Zhijie Chen[c]
MOE Key Laboratory for Urban Transportation Complex Systems Theory and Technology, Beijing Jiaotong University, Beijing 100044
[a]13114232@bjtu.edu.cn, [b]qxsun@bjtu.edu.cn, [c]14114225@bjtu.edu.cn

Keywords: shortest paths; improved Dijkstra's algorithm; urban rail transit network; nodes with weights; labeling edge.

Abstract: Paths searching is a significant work for urban rail transit network. Because of various time weights included in nodes on this network, the traditional Dijkstra's algorithm fails to find the shortest paths. In order to still use this traditional algorithm, the general approach is to expand this network, but additional efforts are introduced. Another approach that will be proposed in this paper adopt the dual principle which changes the label-object from node to edge, and it doesn't need to expand the network, meanwhile, we take the time into/outside a station into account. The improved Dijkstra's algorithm has a time complexity of $O(m\log m)$. And the comparison of their computational efficiency on urban rail transit of Beijing shows that the new algorithm's computing time is a little longer than the traditional algorithm's. The time difference is about 0.016 seconds between a single OD pairs averagely. However, we think that the saving time with no need for additional efforts can balance the increased computing time.

Introduction

The shortest paths problem is one of the most fundamental network optimization problems, and is applied in the field of transportation and network communication widely. Many researchers have studied this problem, and have proposed a series of shortest paths algorithms. There are a lot of papers[1,2,3] that analyze various shortest paths algorithms from both aspects of theoretical analysis and practical application. The theoretical analysis is focused on algorithms' time complexity, while practical application is focused on the computing time of these algorithms used on different network structures. Lu have reviewed various shortest paths algorithms qualitatively[4]. The shortest path algorithm proposed by Dijkstra[5] in 1959 performs better than other algorithms on digraph without negative-value edge.

Paths searching is the precondition of the ticket income distribution and the passenger flow assignment of urban rail transit network. Xu et al[6] proposed a transit assignment model considering passengers' multi-path choice behavior. This model is based on paths searching, that is, they have to find K shortest paths before assigning traffic, and the shortest path is the fundamental of K shortest paths. Considering various weights within nodes on urban transit network, we have to expand these nodes before using the traditional Dijkstra's algorithm with labeling node. However, there is not only a larger scale network, but also additional efforts to map the shortest path on expanded network into original network, while expanding the original network. A good approach[7,8,9,10] is used to search shortest paths without expanding nodes on urban road network, which adopts dual principle to change the label-object from node to edge. However, there are some differences between urban rail transit network and road network, for example, the time into or outside a station appeared merely on urban rail transit network.

In order to search the shortest paths on unexpanded urban rail transit network, we reference the approach which labels edge instead of node on urban road network, and take the time into or outside a station into account meanwhile, to improve the traditional Dijkstra's algorithm.

The analysis of urban rail transit network structure

As soon as the OD(Origin and Destination) on urban rail transit network was determined, the ticket price of this trip is decided accordingly. Hence, passengers just care the travel time when they choose the path between this OD pairs. Moreover, we find the path with minimum travel time in this paper. The total time of a trip on urban rail transit network contains five parts: the time into a station, the train's running time at an interval, the train's dwell time at a station, the transfer time of passengers and the time outside a station. Notice that, the time into a station refers to the time from outside of a station to the station's platform, also includes the time for waiting the train's reach; moreover, the time into the same station and destined for different lines are different, as well the time outside a station. Likewise, the transfer time of passengers also contain the time for waiting the other train's reach. Considering the different influence extent of the five parts of the total time to passengers' path choice, we have to deal with them reasonably. For example, we can multiply an enlargement coefficient by the transfer time of passengers, because this part of time is more influential than other parts to passengers' path choices.

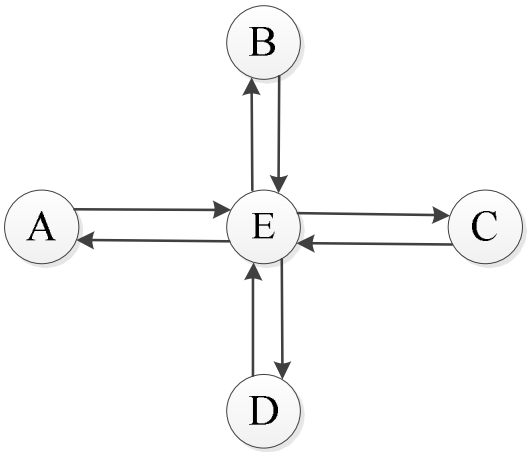

Fig. 1 A sketch map of urban rail transit network

In order to represent the urban rail transit network directly and obviously, we usually adopt the topological structure as shown in Fig. 1, in which a node or an edge refers to a station or a direction of train running interval. Nevertheless, this structure only describes the train's running time as the attribute of edge, and the other four parts of the total time are included in nodes. Because of nodes including various time weights, the traditional Dijkstra's algorithm with labeling node fails. For example, when the node E is scanning in the process of finding the shortest path between A and C, we can't decide the node E's label value(refers to the total time of the path from A to E). The cause is that

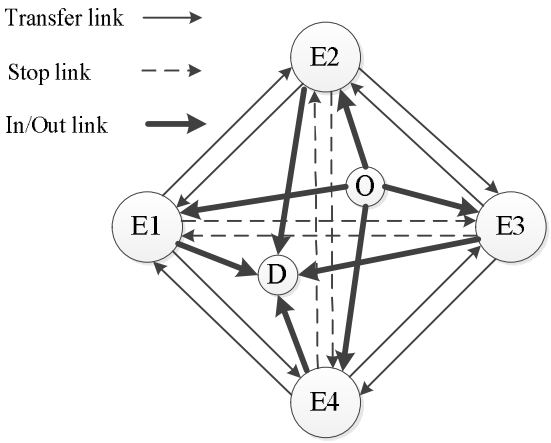

Fig. 2 The expanded graph of a transfer station

we can't determine which transfer time or dwell time is selected in node E before the next node is chosen(perhaps B or C or E). To make the traditional Dijkstra's algorithm available, the general

approach is to expand all the nodes on the network. The expansion of node E, as shown in Fig. 2, can represent the other four kinds of time of a trip as the attribute of different edge.

Although the traditional Dijkstra's algorithm is valid on the expanded network(graph), the more large-scale network needs either a larger computer storage space to save the graph, or more time to scan the nodes on this graph. In addition, the shortest path on the expanded network has to transform into the path represented by nodes(stations) on original network, which is another additional trouble work. In order not to expand the original network, some researchers adopt a new approach to deal with the problem that nodes include various time weights on urban road network[9], in which they give label-value to edge instead of node. They have to decide the first and the last edge firstly before finding the shortest path between an OD pairs. This is reasonable because a trip starts on an edge and ends on another edge on urban road network, however, a trip on urban rail transit network has to starts on a node(station) and ends on another node. In addition, the time of into or outside a station only appears on urban rail transit network. Hence, we adopt the approach with labeling edge on urban road network for reference in this paper, meanwhile, consider the time of into or outside a station, to improve Dijkstra's algorithm and make it available on unexpanded urban rail transit network.

A new implementation of Dijkstra's algorithm

The traditional Dijkstra's algorithm based on labeling node starts at origin node and search outward step by step until the destination node is found. Firstly, two sets are constructed: a node set *D* stores the nodes labeled permanently and the other node set *S* contains the nodes labeled tentatively. Then extract the node with minimum label-value from the tentative set *S* and put it into the permanent set *D*, and then update its successive nodes' label-value and put them into the tentative set *S*. the algorithm terminates , if the destination node is labeled permanently or the tentative set *S* is empty. The fundamental difference between the traditional Dijkstra's algorithm and the improved algorithm in this paper is that they have different label-object(the former is node and the latter is edge). First let we denote some variables that will be used in new algorithm.

$G(V,E)$ refers to original graph without expanding, where V refers to the set of n nodes(stations) and E refers to the set of m edges;

p_{st} refers to the shortest path between origin s and destination t, and we take the stack to store the sequence of these nodes in p_{st};

e_{ij} refers to the edge start from node i to node j, therein, $i, j \in V$;

I_j refers to the set of all edges that enter into node j, that is, $I_j = \{e_{ij} \,|\, e_{ij} \in E, i \in V\}$;

O_i refers to the set of all edges that exit from node i, that is, $O_i = \{e_{ij} \,|\, e_{ij} \in E, j \in V\}$;

c_{ij} refers to the train's running time at the interval(edge) e_{ij};

a_{ij} refers to the time into station i, and then choose the interval(edge) e_{ij} to travel;

b_{ij} refers to the time that passengers reach at station j from interval e_{ij} and go out of the station;

c_{ijk} refers to the transfer time or dwell time, if the station i and k belong to the same line, c_{ijk} refers to dwell time, otherwise the transfer time;

p_{ij} refers to the label-value of the edge e_{ij}, that is, the total time of the path from origin s to the edge e_{ij} on current iterative step;

F refers to a map that stores the predecessor edge of every edge labeled either permanently or temporarily. If the path cost(time) from origin to the edge e_{jk} is minimized by routing the edge e_{ij}, that is $p_{jk} > p_{ij} + c_{jk} + c_{ijk}$, we denoted e_{ij} as the predecessor edge of e_{jk}, expressed as $F[e_{jk}] = e_{ij}$;

S refers to the set of all edges labeled temporarily. The set S is a minimum priority queue such that the first element of S is the edge with minimum label-value in this set;

D refers to the set of all edges labeled permanently. These edges' label-value and predecessor edge don't change any more;

$L[e_{ij}]$ refers to the set of all successive edges of the edge e_{ij}. The successive edge of e_{ij} is the edge that exit from the node *j*, but the edge e_{ji}.For example, $L[e_{\mathrm{AE}}]=\{e_{EB},e_{\mathrm{EC}},e_{\mathrm{ED}}\}$ as shown in Fig. 1.

The traditional Dijkstra's algorithm is based on the *Sub-path Optimality Principle*, that is, any sub –path p_{ik} of the shortest path p_{st} is also the shortest path between *i* and *k*. However, weights within nodes on urban rail transit network result in that this principle isn't satisfied any more. There is another version of Sub-path Optimality Principle related with edge [9], that is, if the path $p_e=\{e_1,e_2,\mathbf{L},e_k,\mathbf{L},e\}$ is the shortest path from origin node *r* to the edge *e*, any sub-path $p_{e_k}=\{e_1,e_2,\mathbf{L},e_k\}$ of p_e is also the shortest path from node *r* to the edge e_k (the edge e_1 exit from the node *r*). This variant principle is the foundation of the new Dijkstra's algorithm with labeling edge in this paper. Because the label-object is edge instead of node, while using some edge to update one of its successive edges, the transfer time or dwell time between them is determined automatically. Now we present the flow chart of the new algorithm, as shown in Fig. 3.

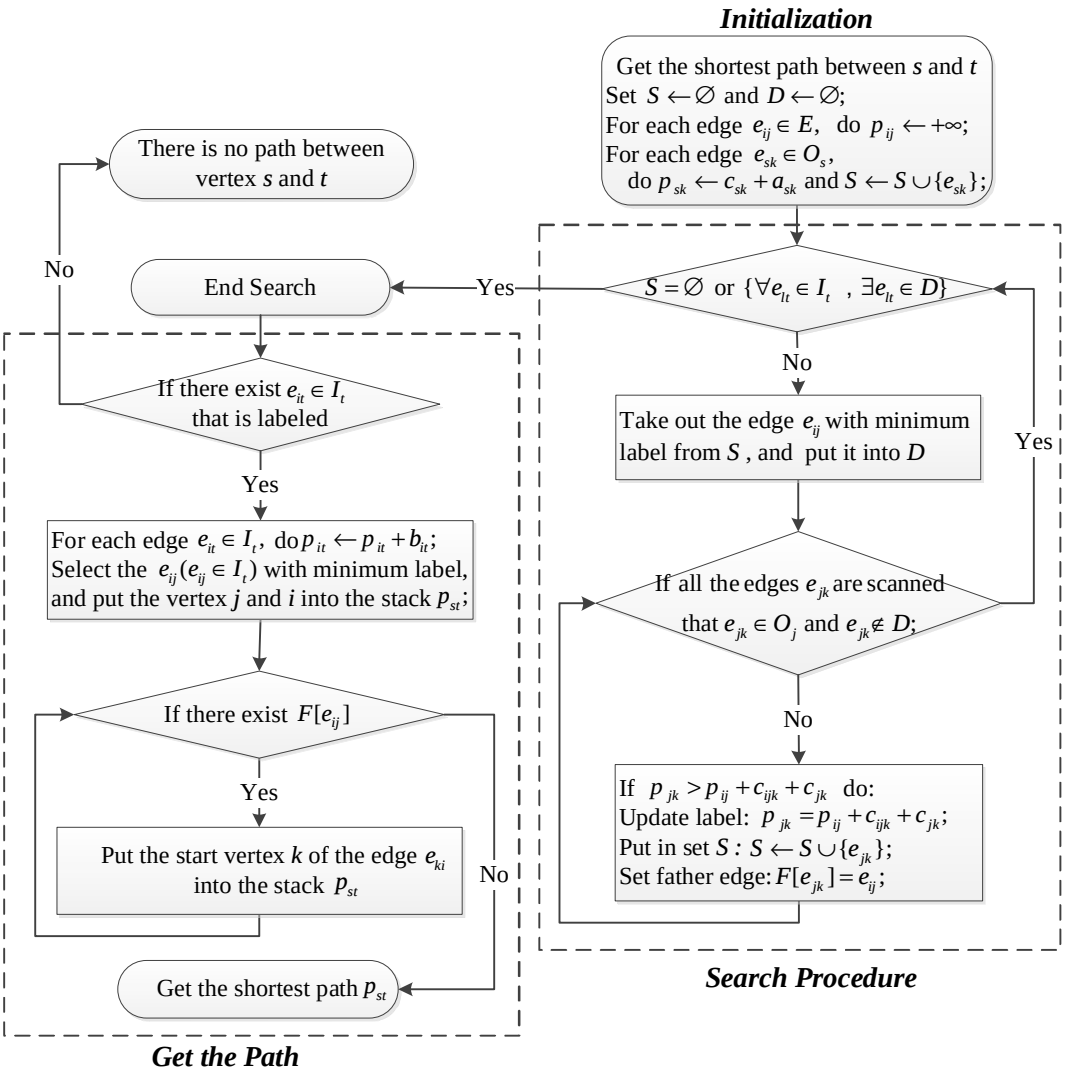

Fig. 3 The flow chart of new Dijkstra's algorithm with labeling edge

As we can see from Fig. 3, the new Dijkstra's algorithm is composed of *Initialization*, *Search Procedure* and *Get the Path*. We initialize the label-value of all the edges in the graph *G* in the *Initialization* phase. With regard to every edge exiting from the origin node *s*, its initial label-value is the sum of its train's running time c_{sk} and the corresponding time into station *s*, and the other edges'

label-value are set to infinity. In the second phase, similar to the traditional Dijkstra's algorithm in the iterative process, the new algorithm extracts an edge with minimum label-value in the temporary set S and put it into the permanent set D, and then update the label-value of all its successive edges, until that the set S is empty or all edges entering into the destination t are permanently labeled. In the third phase, because of considering the time outside a station, we can't determine which the shortest path is before adding the corresponding time outside station s. With regard to every edge entering into destination t, we add its corresponding time outside station s to its label-value, and select the edge with minimum value as the last edge of the shortest path. We can get the all nodes in the shortest path by means of going back from the last edge to origin s using the map F that stores the predecessor edge of every edge labeled either permanently or temporarily.

Different data structure used to store the edges(nodes) labeled temporarily corresponds to different time complexity in the Dijkstra's algorithm. There is the best computational complexity $O(m+n\log n)$ of the traditional Dijkstra's algorithm by using a Fibonacci-heap so far[11]. We use the dual principle that changes the label-object from node to edge in this paper, however, the search strategy is similar. While extracting the minimum element from the set S, we have to determine whether all the edges entering into destination t are labeled permanently or not, that is, to check the permanent set D with the time complexity $O(\log m)$ every loop. Therefore, the time complexity of the new Dijkstra's algorithm is $O(m+m\log m)+m\times O(\log m)$, that is $O(m\log m)$, so that there may be a good computational efficiency on a sparse graph.

Computational experiments

In order to compare the traditional Dijkstra's algorithm(Primary Algorithm) and the improved Dijkstra's algorithm(New Algorithm), both of them are applied to find shortest paths on the urban rail transit network of Beijing in 2014(as shown in Fig. 4). There are 227 stations(nodes) and 508 directed edges, and will be increased to 964 nodes and 2292 edges if the network is expanded. The timetable comes from Beijing Mass Transit Railway Operation Corporation, while the transfer time and the time into/outside a station is estimated according to the real situation. Notice that the network doesn't include the Line 7 and Line 14 because they are operated by another corporation. We use C++ programming language to implement both of algorithms on a computer installed a Windows 7 operating system , and its CPU is Intel Core2 2.53 GHz, 2GB RAM.

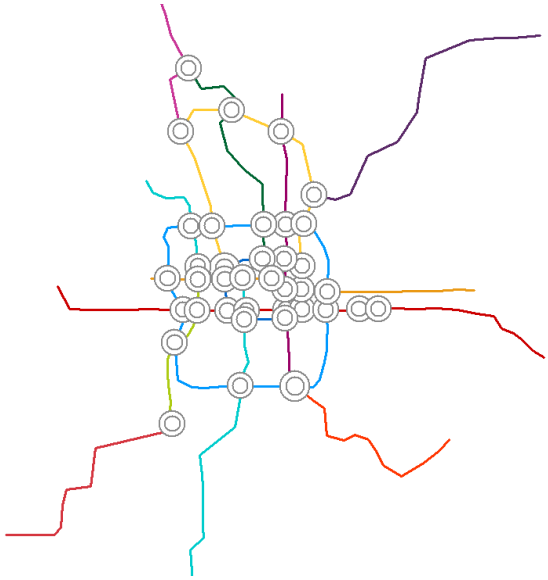

Fig. 4 The topological structure of urban rail transit network of Beijing in 2014

We select 10 groups of OD pairs randomly, such as 100 OD pairs, 200 OD pairs,…, 1000 OD pairs, and then use both of algorithms to find shortest paths of each group. Notice that before using the

Primary Algorithm, we have expanded the Beijing subway network, while the shortest paths are composed of expanded nodes. Hence the computing time of the Primary Algorithm doesn't include the time to expand network and map shortest paths from the expanded graph to the original graph. Although Dijkstra's algorithm could find all the shortest paths from origin node to the others on the graph, the OD pairs in each group are selected randomly, so that there is different origin node in the same group. Without loss of comparability of the two algorithms, hence, we make sure that any shortest path between each OD pairs is searched by using Primary or New Algorithm once, instead of using either of algorithm once to find multi-OD pairs' shortest paths.

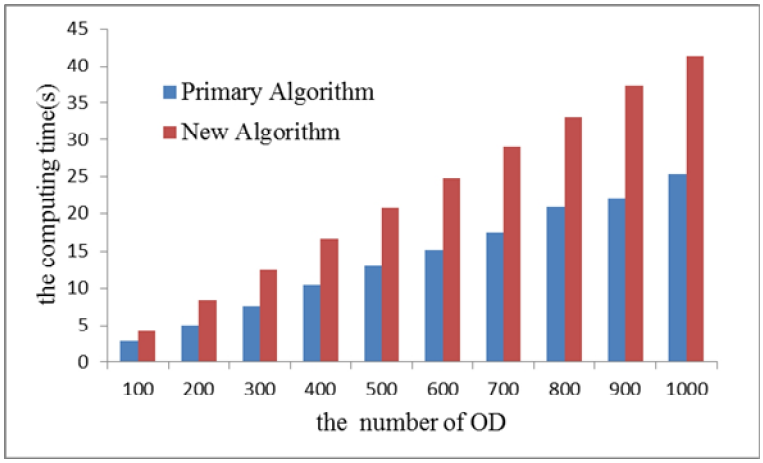

Fig. 5 The comparison of the two algorithms

As shown in Fig. 5, the New Algorithm takes more computing time than the Primary Algorithm in each group, and the gap widens between them with the number of OD pairs increasing. However, the time difference between these algorithms remains stable(about 0.016 seconds), while finding the shortest path of a single OD pairs, as shown in Table 1, the New Algorithm's average computing time of any group is about 0.042 seconds, and the Primary Algorithm's is about 0.026 seconds. It shows that these algorithms have stable performance with random OD pairs on the given network. The reasons why the New Algorithm is slightly slower than the other, is that the extra works to search the permanent set D has to be done during each iteration, and the time to find the transfer time or dwell time in a set which stores these time is added during the updating label process.

Table 1 The average time of different OD groups [s]

the number of OD	100	200	300	400	500	600	700	800	900	1000	mean
Primary Algorithm	0.028	0.025	0.025	0.026	0.026	0.025	0.025	0.026	0.025	0.025	0.026
New Algorithm	0.042	0.042	0.042	0.042	0.041	0.041	0.041	0.041	0.041	0.041	0.042
Differences	0.014	0.017	0.016	0.016	0.015	0.016	0.016	0.015	0.017	0.016	0.016

Although the New Algorithm's computing time is a little longer than the Primary Algorithm, it doesn't need to expand the original network and map the paths on the expanded network to the original network. Therefore, the saving time with no need for these extra works may compensate the New Algorithm's excess computing time on some types of network.

Conclusions

Paths searching is precondition for the passenger flow assignment and the ticket income distribution on urban rail transit network. Because of various time weights within nodes on this network, the traditional Dijkstra's algorithm with labeling node fails to find the shortest paths. We adopt the dual principle that changes the label-object from node to edge, and consider the time into/outside a station meanwhile, to improve Dijkstra's algorithm so that it is available on the original

network without expanding. And the improved algorithm has a time complexity of $O(m\log m)$. The application of both algorithms on Beijing Subway network indicates that the New Algorithm's computing time is a little longer than the Primary Algorithm's. The average difference remains 0.016 seconds steadily between a single OD pairs. The extra works to search the set D and the set which stores all the transfer time and the dwell time result in more computing time in the New Algorithm. However, the saving time with no need for expanding network and mapping the paths on the expanded network into the original network may compensate the New Algorithm's excess computing time on some types of network. And this will be studied in the future.

Acknowledgements

The authors are grateful to the National Basic Research Program (2012CB725406), the National Natural Science Foundation (71131001, 71390332) of China for their supports on this work. The authors also thank the anonymous reviewers and the editor for their suggestions to improve this paper.

References

[1] G. Gallo and S. Pallottino: Ann. Oper. Res Vol. 13 (1988), p. 3-79.

[2] B.V. Cherkassky, A.V. Goldberg and T. Radzik: Math. Program Vol. 73(1996), p. 129-174.

[3] M. Glabowski, B. Musznicki, P. Nowak and P. Zwierzykowski: International Journal on Advances in Software Vol. 7-1(2014), p. 20-30.

[4] F. Lu: Acta Geodaetica et Cartographica Sinica Vol. 30-3(2001), p. 269-275.

[5] E.W. Dijkstra: Numer. Math Vol. 1(1959), p. 269-271.

[6] R.H. Xu, Q. Luo and P. Gao: Journal of the China Railway Society Vol. 31-2(2009), p. 110-114.

[7] Y.L. Chen and H.H. Yang: Transp. Res. Part B Methodol Vol. 34(2000), p. 241-253.

[8] G. Han, J. Jiang, J. Chen and Y.D. Cao: Acta Geodaetica et Cartographica Sinica Vol. 31-4(2002), p. 366-368.

[9] G. Ren, W. Wang and W. Deng: Journal of Southeast University Vol. 34-1(2004), p. 104-108.

[10] L. Yongtaek and K. Hyunmyung: Journal of the Eastern Asia Society for Transportation Studies Vol. 6(2005), p. 1426-1438.

[11] M.L. Fredman and R.E. Tarjan: Journal of the ACM Vol. 34-3(1987), p. 596-615.