Shortest and Least Time-Spent Routing in Subway System

EL7363 Communications Networks II: 0486634   Yaolong Li

In this report, we introduce two algorithms to find shortest paths in a subway system: one is to find all the passible paths then pick the paths which use least nodes or least time; the other is based on Dijkstra's Algorithm to find the path which spend the least weight (in this report, weight is the time spent in the path).

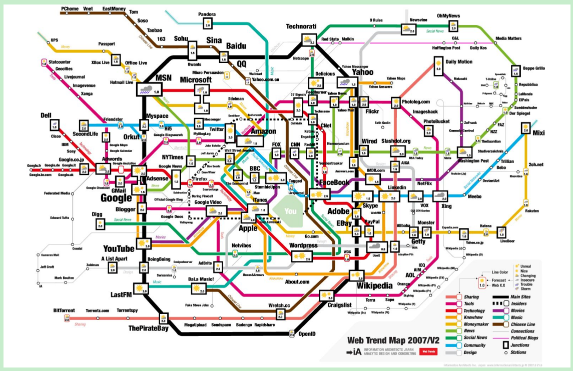
Since Dijkstra's Algorithm is only suitable in the model of only one link between two nodes, which is not the case in the subway system, here, in the second algorithm a modified version is introduced. A concept of label similar to Heuristic is used here. For each iteration, we update the list of label by adding one link to get the new shortest path to a new node.

In Java simulation, both algorithms show the same result, which improves the correctness of the modified Dijkstra's Algorithm.

Keywords: shortest-path, weight, Dijkstra, Heuristic, label, subway system

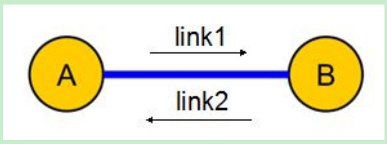
1. BACKGROUND

In big cities like New York and Chicago, a great amount of traffic is in divided into subway system, since the roads in these cities cannot hold so many vehicles and not every one can afford a private car. People have destinations and need a path to follow in a subway system. A single train may not take the whole trip and people need to change their trains in subway station. In the case of the unestimatable factor, such as rainstorm, train damages, railway blocking,the links can change greatly.Therefor, people should change their path suitably. Since the complexity of the subway system, additional help is needed to get the path.

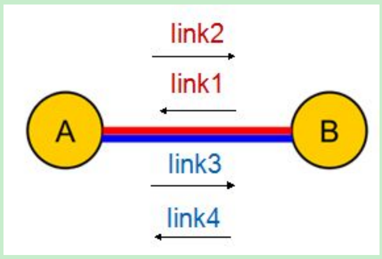


2 Model The System

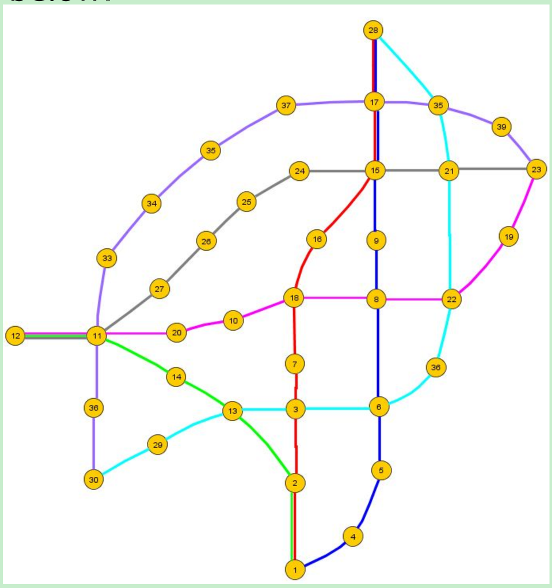
In this model, we have N Stations, linked by L links. Stations are considered as Nodes in this system. Links are directed, which means for the same line between two stations (nodes), there are two links. And these two links have opposite directions.



For example, Node A and Node B is linked by a railway, however, there are directed links on this road, they are link1 and link2.



For two station, there also may more than one subway lines. Thus, more links are put there. For each link, there only one subway line(e.g R,D,F train).
Following the above principles, we build a virtual subway system just as the photo below.



Each color line stands for one subway line

3 Algorithms to Find Path

3.1 Algorithms to Find Path all the paths

Outlist: For one node, the list of the links come from this node.

Suppose that the user is in station A and wants to go to station B.
2. Step 1: Firstly, get the node(station) A's outlist. In the list, get the first link: link1. Then, we can go to station X1 with

link1.(since for one link there is only one terminal and one original node)
3. Step 2: Check X1,
 (1)if X1 is node B, then stop , we get the terminal. Remember this path.
 (2)If X1 is A or other stations which we have gone through, stop using this link1, use the second link(link2) in the outlist of A, then went to step 2 and check the link2's terminal node X2.
(3)If it's the first time to go through X1, add X1 to a node list, called used-node list. Then, start from X1 to follow step 1 again.
4 .Try all the nodes and links, then we get all the paths available.

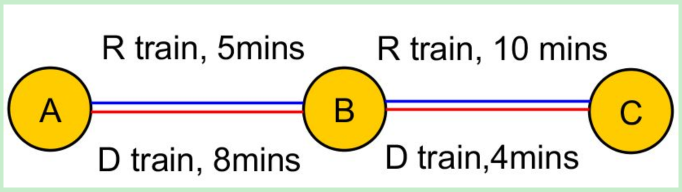
Thus, we can get all the possible paths to the users' terminal. Then, it is easy to get the number of nodes needed and spent time for each path. The Algorithm to find the need nodes and time is introduced in slides P15-P16.

3.2 A modified Dijkstra's Algorithm to find the shortest path

A. Why Dijkstra's Algorithm cannot used in the subway model ?
B.

In the original model of Dijkstra, there is only one link between nodes. In subway, there is more than one links between nodes. when try to find the time spent on the path, the station waiting-time should be take into consideration.The waiting time can be changed with time, while the on-train time is the same.

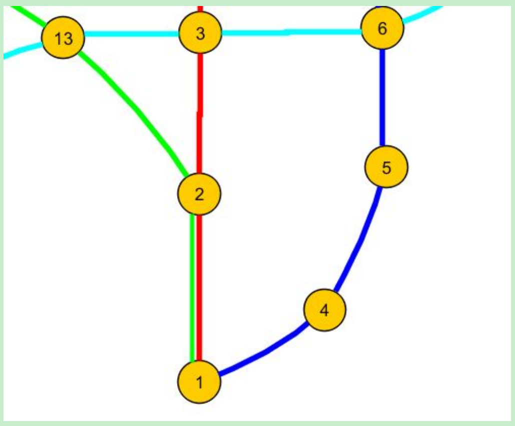
For example, one user wants to go to station C from station A through station B. We use Dijkstra's Algorithm to find the least time-spent path is R train from A to B. Then we consider going to B, R is needed. Same way, we find D train from B to c. In the end, the path is take R train from A to B, then change to D train to C, since in Dijkstra's Algorithm principle, no matter what is the terminal station, we must use R train from A to B if we go through B. However, It is possible a wrong result. Though it is true that from A to B, R train is best, changing from R train to D train, we need to wait at station B for a long time, so, the best path maybe taking D train from A to C without any changing without any wait at station B.



B. A new Algorithm in the subway system

Here, we consider a modified Dijkstra. Since to find the least-time spent path is more complex than to find the shortest node path, we will only take consider the least-time case here. (to convert a least-time problem into a shortest node problem,just need to change the time weight to 1)
In this example, user wants come from Node 1 to Node 6.
here, we introduce a concept of label
label : (node, time-weight,root link)
node: for a single node.
time-weight: the time when we reach. this node.
root link: in order to reach this node, the last one link we need.

Step 0: we get node 1, and take node label as **(node 1, 0, null)**. Since node 1 is the original node, we need not wait any time to get to it (weight-time=0), and it has no root link. Save this label in a list.



Step 1: we scan this list, we get label about node 1. With the links which come from node 1, we can reach node 2 and node 4. Then, we get labels as below:

(node 2, weight-time1, green link);
(node 2, weight-time2, red link);
(node 4, weight-time3, blue link).

Weight_time1=0+wait_time(green)+travel-time(green)
Weight_time2=0+wait_time(red)+travel_time(red)
Weight_time3=0+wait_time(blue)+travel_time(blue)

Find the label which owns the smallest weight-time, for example, if **weight-time1<weighttime2<weight-time3,** we will only keep first label about node2 by green link into the list, and delete the other labels. Thus, we can consider from the very begin time to weight-time1, we can only go to node 2 with green link using the shortest time.

Step 2: Scan the list again, this time, we get labels about node 1 and node 2. From these nodes, get their nearby modes (node 3, node 13, node 4) and calculate the new label just like in Step 1:
(node X, weight-time, a colored link)
$$\text{weight\_time} = \text{root node'sweight\_time} + \text{wait\_time} + \text{travel\_time}$$
Save the label which has the smallest weight-time to the list.

step 3: Get to step 2 again, until node X is the node the user wants. Then, the terminal node's weight-time is the time to reach the terminal.

Then, from the terminal node, we can get the root, and the root-node, and the root-node's root....Thus, We get all the root links, we get the path. This algorithm is similar to Heuristic, since we should compare each new link connected to existing nodes as each "expansion " of our label list.

C. Proof of this algorithm
If we find a path P from A to B by using this algorithm, and the weight-time of label B is T, and T is the time needed for the trip. Easy to see, by each search for a new node, the new label's weight time is increasing (the weight-time equation in step 2). If path P is not the shortest time-spent path, there must be a path X from A to B, and the time needed is less than T, which means this path has been searched before path P. However, if we did find B by using path X, we would have reach node B. In step 3 principle, if we had found B by path X, we should have stopped instead of search for B anymore, and we cannot get path P. Thus, it is impossible to find a path use less time than T.

3.3 The Comparisons of this two algorithms

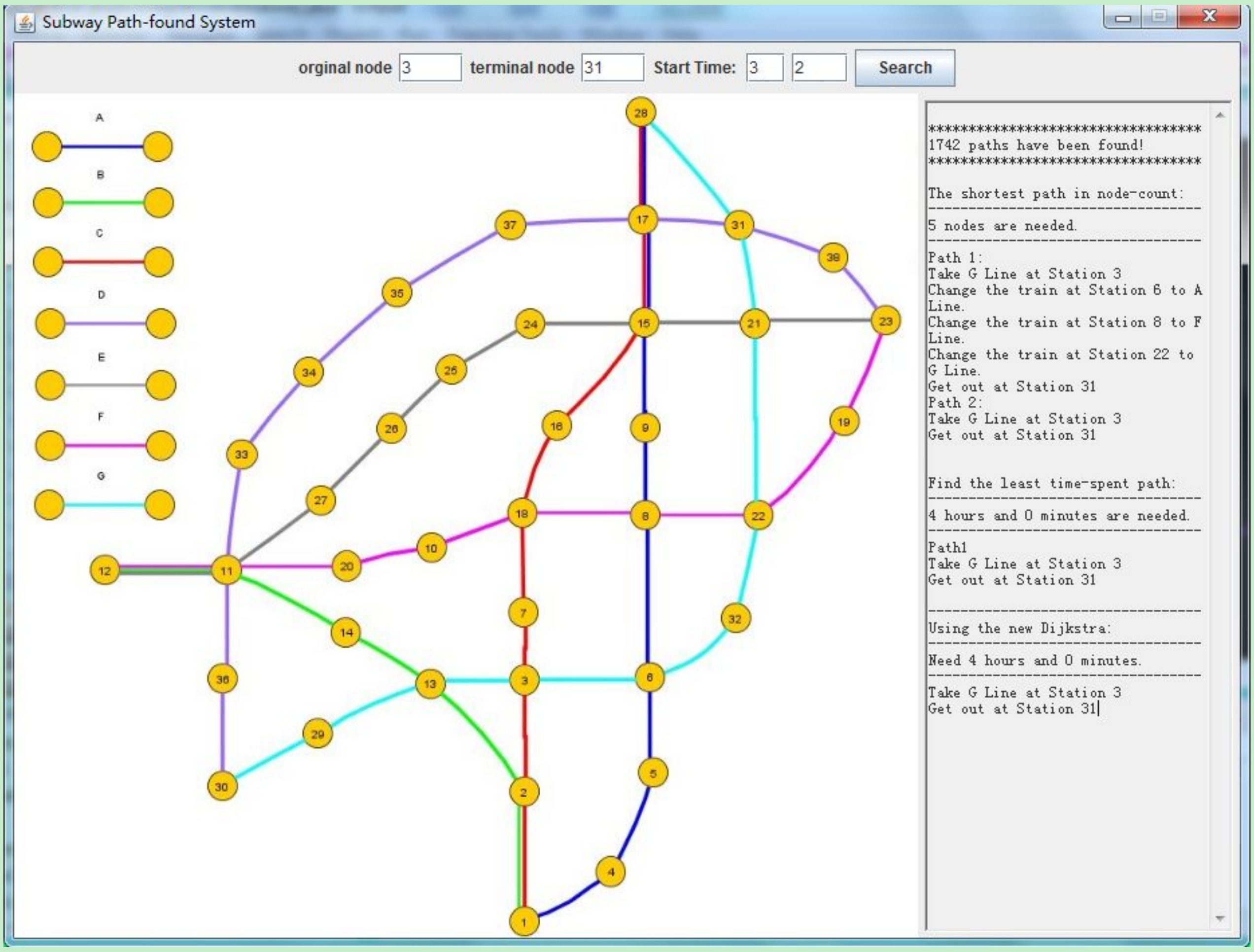
The modified Dijkstra Algorithm can save great time to find the shortest path since not all the path has been found. However, since for one node, there is only one label, one root link in the modified

Dijkstra Algorithm, which means only one path can be found. In the subway system, there may be more than one paths have the least spent time. In the case of the first algorithm, every path can be found.

4 Java Programming

In this java program, a GUI window is introduce, user can select the start node and the end node. Also, user should put into the time he wants to begin his journey, since this model just like a real world each station has a timetable for each train.

Yaolong Li 0486634



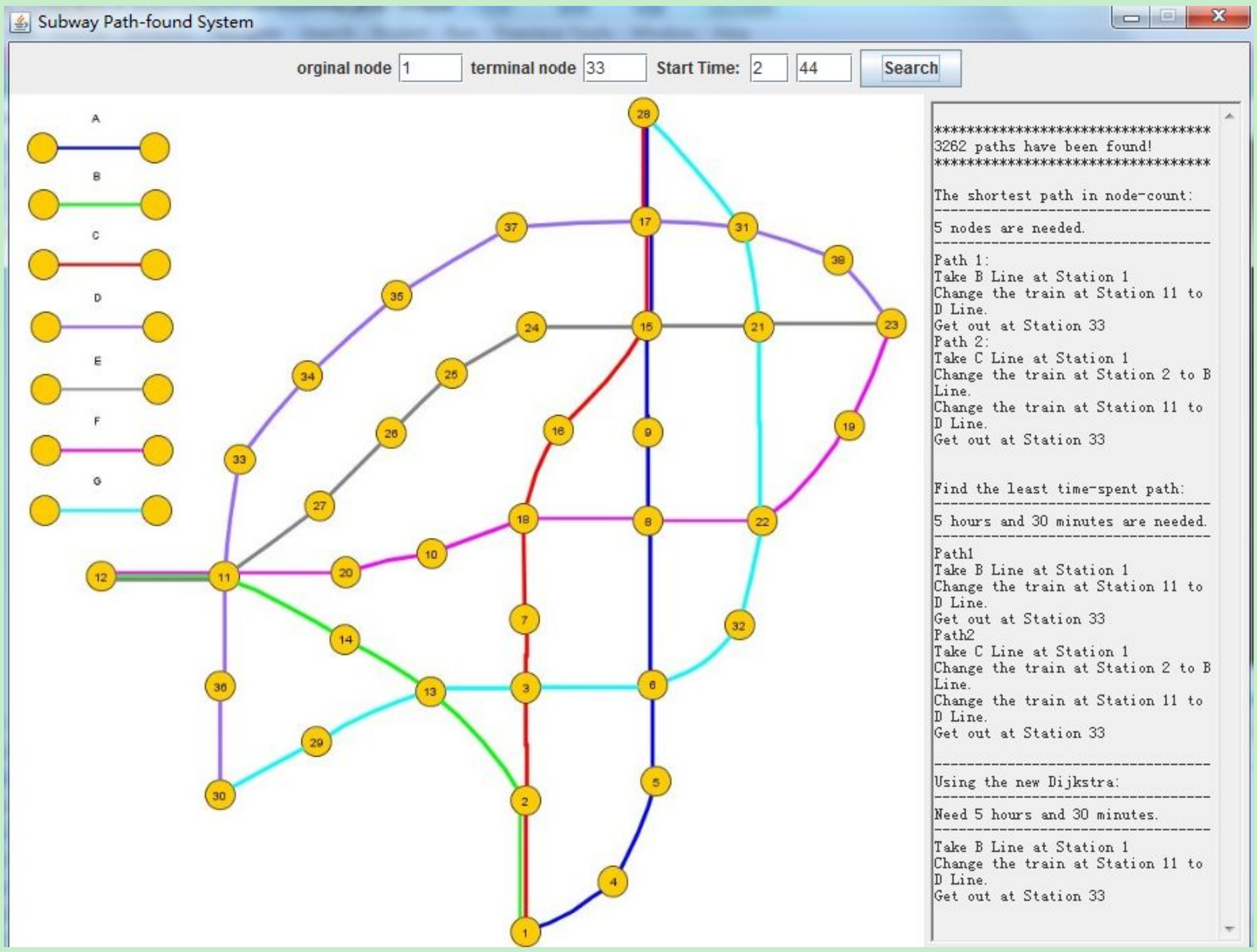
orginal node 3 terminal node 31 Start Time: 3 2

```
**********************************
1742 paths have been found!
**********************************

The shortest path in node-count:
----------------------------------
5 nodes are needed.
----------------------------------
Path 1:
Take G Line at Station 3
Change the train at Station 6 to A
Line.
Change the train at Station 8 to F
Line.
Change the train at Station 22 to
G Line.
Get out at Station 31
Path 2:
Take G Line at Station 3
Get out at Station 31


Find the least time-spent path:
----------------------------------
4 hours and 0 minutes are needed.
----------------------------------
Path1
Take G Line at Station 3
Get out at Station 31

----------------------------------
Using the new Dijkstra:
----------------------------------
Need 4 hours and 0 minutes.
----------------------------------
Take G Line at Station 3
Get out at Station 31
```



orginal node 1 terminal node 33 Start Time: 2 44

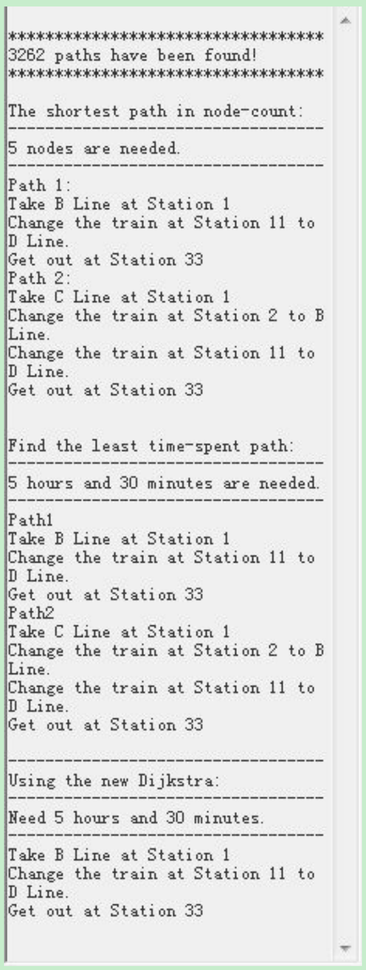
```
**********************************
3262 paths have been found!
**********************************

The shortest path in node-count:
----------------------------------
5 nodes are needed.
----------------------------------
Path 1:
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
Path 2:
Take C Line at Station 1
Change the train at Station 2 to B
Line.
Change the train at Station 11 to
D Line.
Get out at Station 33


Find the least time-spent path:
----------------------------------
5 hours and 30 minutes are needed.
----------------------------------
Path1
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
Path2
Take C Line at Station 1
Change the train at Station 2 to B
Line.
Change the train at Station 11 to
D Line.
Get out at Station 33

----------------------------------
Using the new Dijkstra:
----------------------------------
Need 5 hours and 30 minutes.
----------------------------------
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
```

Yaolong Li 0486634



```
**********************************
3262 paths have been found!
**********************************

The shortest path in node-count:
----------------------------------
5 nodes are needed.
----------------------------------
Path 1:
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
Path 2:
Take C Line at Station 1
Change the train at Station 2 to B
Line.
Change the train at Station 11 to
D Line.
Get out at Station 33


Find the least time-spent path:
----------------------------------
5 hours and 30 minutes are needed.
----------------------------------
Path1
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
Path2
Take C Line at Station 1
Change the train at Station 2 to B
Line.
Change the train at Station 11 to
D Line.
Get out at Station 33

----------------------------------
Using the new Dijkstra:
----------------------------------
Need 5 hours and 30 minutes.
----------------------------------
Take B Line at Station 1
Change the train at Station 11 to
D Line.
Get out at Station 33
```

Number of all the possible path paths

The shortest path found by the first algorithm

The shortest-time path found by the first algorithm

The shortest-time path found by the new algorithm