

An Extended Shortest Path Problem with Switch Cost Between Arcs

Xiaolong Ma, Jie Zhou *

Abstract—Computing the shortest path in a graph is an important problem and it is very useful in various applications. The standard shortest path problem has been studied extensively and intensively, but it can't handle the situation when there is a switch cost between arcs. For example, in a train transportation network, the switch cost between arcs contains waiting time in stations, times of transfer and so on. Obviously, the switch cost is an important factor for users to make decisions. Taking into consideration of the switch cost between arcs, we extend the standard shortest path problem and propose an algorithm and its optimized version to solve the extended single source shortest path problem. Test results show that the proposed algorithms can give reasonable and acceptable results for users.

Keywords: *Switch Cost, Extended Shortest Path Problem, Extended Dijkstra Algorithm, Optimized Extended Dijkstra Algorithm*

1 Introduction

Computing the shortest path is a classic graph problem. It has many applications in real word such as internet routing, transportation, games and so on. Many researches relating to it have been done. Even recently, there are still a lot of work being carried out.

However, the standard shortest path problem doesn't take the switch cost between arcs into account. In some scenarios, the switch cost between arcs does exist and is very important. For example, in a train transportation network where the vertices represent stations and arcs represent the train routes from stations to stations, the waiting time in a station is a kind of switch cost between arcs and it depends on the arcs. Obviously, the waiting time in stations should be considered when computing the optimal travel route. To handle the situation with switch cost between arcs, we will propose an extended shortest path problem.

Some other works[1] [2][3][4] were done to resolve the Earliest Arrival Problem (*EAP*) and Minimum Number of Transfers Problem (*MNTP*). However, those problems

are different from the proposed one. The proposed problem focuses on integrating the switch cost between arcs into the path cost, which provides a more general framework. The *EAP* and *MNTP* problem can be regarded as special cases of the proposed problem.

We also propose an algorithm to solve the extended single source shortest path problem. The algorithm can easily combine the switch cost with the path cost. However, this algorithm tends to expand many arcs. To fix this problem, we give an optimized algorithm to reduce the number of expanded arcs.

The contributions can be summarized as follows:

- * propose an extended shortest path problem by integrating the switch cost between arcs into the path cost.
- * develop an algorithm and its optimized version to resolve the single source problem based on the extended shortest path problem.

The rest of this paper is organized as follows: In Section 2, the extended shortest path problem will be proposed. The extended Dijkstra algorithm and its optimized one will be given in Section 3. An application example is presented in Section 4. The experiment result will be shown in Section 5. And some conclusions will be drawn in the end.

2 Extended Shortest Path Problem

The standard shortest path problem only considers the cost on the arc, and it doesn't consider the cost of switching between arcs. But in some applications such as transportation networks, the switch cost between arcs does exist and is important. So it is necessary to solve the problem by considering switch cost between arcs. This paper will focus on this point. To explain the problem better, we define some notations first.

- * G : a weighted directed graph
- * V : the collection of vertices in G

*Research Center of Modern Service Science & Technology. Department of Automation, Tsinghua University, Beijing, 100084, China. maxl@mails.tsinghua.edu.cn, jzhou@tsinghua.edu.cn.

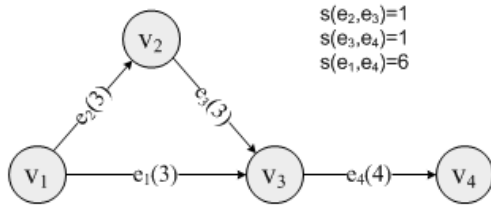


Figure 1: A simple demonstration for the extended shortest path problem. $e_2(3)$ means $w(e_2) = 3$. $s(e_2, e_3) = 1$ means the switch cost between e_2 and e_3 equals 1.

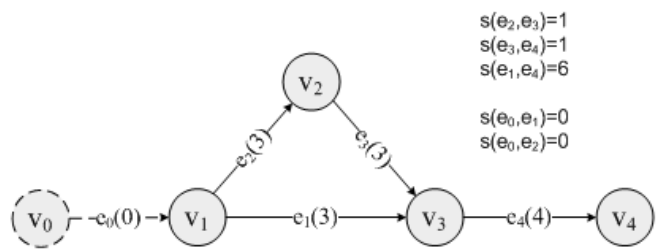


Figure 2: Add a virtual node v_0 and a virtual arc e_0 where $e_0.head$ equals to the src node.

- * E : the collection of arcs in G . There may be multiple arcs with the same direction between two vertices in G .
- * n : the number of vertices in V , i.e. $n = |V|$.
- * m : the number of arcs in E , i.e. $m = |E|$.
- * I_i : the number of arcs that go into vertex v_i .
- * O_i : the number of arcs that go out from vertex v_i .
- * v : a vertex of G . $v \in V$
- * e : a arc of G . $e \in E$
- * $e.head$: a vertex in G which e goes into.
- * $e.tail$: a vertex in G which e goes out from.
- * $w(e)$: the nonnegative weight of arc e
- * $s(e_1, e_2)$: the nonnegative switch cost between arc e_1 and arc e_2 . It is valid only when $e_1.head$ equals to $e_2.tail$.
- * p : an arc sequence which is named *path*. For example, path $p = e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_k$ is a path from vertex $e_1.tail$ to $e_k.head$, where $e_i.head = e_{i+1}.tail$.
- * $c_s(p)$: the cost of path p in the standard shortest path problem
- * $c_e(p)$: the cost of path p in the extended shortest path problem

Given a path $p = e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_k$, the cost of the path p is defined as $c_s(p) = w(e_1) + w(e_2) + \dots + w(e_{k-1}) + w(e_k)$ in the standard shortest path problem, while it is defined as $c_e(p) = w(e_1) + s(e_1, e_2) + w(e_2) + \dots + w(e_{k-1}) + s(e_{k-1}, e_k) + w(e_k)$ in the extended shortest path problem. When $s(e_i, e_j)$ is a const c for all arc pairs, the extended shortest path problem is identically to the standard shortest path problem. Since let $w'(e_i) = w(e_i) + c$, then $c_e(p) + c = c'_s(p) = w'(e_1) + w'(e_2) + \dots + w'(e_{k-1}) + w'(e_k)$.

In the standard shortest path problem, if $p = e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_k$ is the shortest path between $e_1.tail$ and $e_k.head$,

then $p_{ij} = e_i \rightarrow e_{i+1} \rightarrow \dots \rightarrow e_j$ must be the shortest path between $e_i.tail$ and $e_j.head$. But it is no longer right when there exists non-zero switch cost between arcs. Fig 1 is a simple example. In the standard shortest path problem, the shortest path between vertex v_1 and v_4 is $e_1 \rightarrow e_4$, while the shortest path between vertex v_1 and v_3 is e_1 . But in the extended shortest path problem, the shortest path between vertex v_1 and v_4 is $e_2 \rightarrow e_3 \rightarrow e_4$, and the shortest path between vertex v_1 and v_3 is e_1 , but not $e_2 \rightarrow e_3$.

3 Extended Algorithms

Dijkstra algorithm[5] is a classic one to solve the single source shortest path problem in a directed graph with nonnegative arc weights. Many efforts [6] [7] [8] [9] [10] [11] have been done to speed up this algorithm in both theory and practice.

However, Dijkstra is based on the standard shortest path problem, so it can't take into account the switch cost between arcs. Inspired by the Dijkstra algorithm, we propose extended algorithms based on it to resolve the extended single source shortest path problem.

3.1 Overview of the algorithms

Before diving into the algorithms, we make some transformation in the graph which can help better understand the algorithm. We add a virtual vertex v_0 and a virtual arc e_0 to G , where $w(e_0) = 0$, $e_0.tail = v_0$, $e_0.head = v_1$ (suppose that v_1 is the src node here). Fig 2 is a simple demonstration.

In the standard Dijkstra algorithm, each vertex has two labels: *previous vertex* and *shortest value*. Since there may be multiple arcs with some direction between arcs, *previous arc* is used to replace *previous vertex*. $v.prevArc$ and $v.val$ are used to represent these two labels separately. In our algorithm, each arc has two labels: *previous arc* and *shortest value*, and each vertex also has two labels: *selected arc* and *shortest value*. $e.prevArc$, $e.val$, $v.selectedArc$ and $v.val$ are used to represent these labels separately.

We will detail the explanation with Fig 1. Supposing we want to compute the shortest path from vertex v_1 to other vertices. Fig 3(a) is the label result for the standard Dijkstra algorithm. Fig 3(b) is the label result for the extended Dijkstra algorithm. To explain how to get the shortest path by using the label result, we'll take v_1 and v_4 for example. Suppose we want to get the shortest path from v_1 to v_4 , the steps in Fig 3(a) are shown as follows:

1. Check $v_4.prevArc$ and find that it is e_4 . And $e_4.tail = v_3$.
2. Check $v_3.prevArc$ and find that it is e_1 .
3. Since $e_1.tail = v_1$, then we get the standard shortest path from v_1 to v_4 is $e_1 \rightarrow e_4$.

In Fig 3(b), the steps are given as follows:

1. Check $v_4.selectedArc$ and find that it is e_4 .
2. Check $e_4.prevArc$ and find that it is e_3 .
3. Check $e_3.prevArc$ and find that it is e_2 . $e_2.prevArc = 0$ means we reach the virtual arc.
4. Check $e_2.tail$ and find that it is v_1 , which means that the extended shortest path from v_1 to v_4 is $e_2 \rightarrow e_3 \rightarrow e_4$.

In the following algorithms, the first parameter of the corresponding function means the graph, and the second means the source vertex. The *ExtractMin* function extracts the element with minimum value from the given set O .

3.2 Extended Dijkstra Algorithm

The extended Dijkstra algorithm is used to solve the extended single source problem. Since it is similar to the standard Dijkstra algorithm, we omit the proof of correctness here. This algorithm will be called E-Dijkstra algorithm in the following.

```

program ExtendedDijkstra(G,u)
1 for each vertex v in V[G]
2   v.val=infinity; v.selectedArc=undefined;
3 for each arc e in E[G]
4   e.val=infinity; e.prevArc=undefined;
5 e0.val=0; e0.prevArc=null;
6 e0.tail=null; e0.head=u;
7 O=E[G] union e0; C={};
8 while O is not empty
9   e=ExtractMin(O);
10  C=C union e;
11  if e.val<e.head.val

```

```

12    e.head.val=e.val;
13    e.head.selectedArc=e;
14  for each arc a outgoing from e.head
15    if e.val+s(e,a)+w(a)<a.val
16      a.val=e.val+s(e,a)+w(a);
17    a.prevArc=e;

```

The space complexity of this algorithm is $O(n+m)$. The computation complexity of this algorithm is $O(N \log N + M)$, where $N = m + 1$ is the number of arcs and $M = \sum_{i=0}^n I_i O_i + O(u)$ is the number of expanded arcs. m, n, I_i, O_i have been defined before and $O(u)$ is the number of u 's in-arcs.

We can find that the number of expanded arcs is too huge in the proposed algorithm. In order to reduce the number of expanded arcs, we propose an optimized version of this extended algorithm in the following.

3.3 Optimized Extended Algorithm

We need to compute $e.minCost$ for each arc in advance. $e.minCost$ is the minimum switch cost between $e.tail.inArcs$ and e , where $e.tail.inArcs$ is the collection of arcs which goes into $e.tail$. Similarly, in the following codes, $e.head.outArcs$ means the collection of arcs that goes out from $e.head$.

Round i represents the i th time during which codes 11-23 in the following program are executed. Supposing at *Round i*, e_{r_i} is the arc with minimum value i.e. $e_{r_i} = ExtractMin(O)$, and at *Round j* $e_{r_j} = ExtractMin(O)$, where $j > i$. According to Dijkstra algorithm, we know that $e_{r_i}.val \leq e_{r_j}.val$. This property is also exist in our algorithm. Supposing arc a is in $e_{r_i}.head.outArcs$ and also in $e_{r_j}.head.outArcs$. If $s(e_{r_i}, a) = a.minCost$, then we know that $a.val \leq e_{r_i}.val + s(e_{r_i}, a) + w(a)$ at *Round j*. Since $e_{r_i}.val \leq e_{r_j}.val$ and $s(e_{r_i}, a) \leq s(e_{r_j}, a)$, we can get $a.val \leq e_{r_j}.val + s(e_{r_j}, a) + w(a)$, which means a won't be updated at *Round j*. So if at *Round i* $s(e_{r_i}, a) = a.minCost$, we can delete a from $e_{r_i}.head.outArcs$ which will never be updated after *Round i*. According to this reason, we obtain the optimized algorithm as following:

```

program OptimizedExtendedDijkstra(G,u)
1 for each vertex v in V[G]
2   v.val=infinity; v.selectedArc=undefined;
3 for each arc e in E[G]
4   e.val=infinity; e.prevArc=undefined;
5 e0.val=0; e0.prevArc=null;
6 e0.tail=null; e0.head=u;
7 for each arc e outgoing from u
8   e.minCost=0;
9 O=E[G] union e0; C={};
10 while O is not empty
11  e=ExtractMin(O);

```

```

12  C=C union e;
13  if e.val<e.head.val
14      e.head.val=e.val;
15      e.head.selectedArc=e;
16  outArcs={};
17  for each arc a in e.head.outArcs
18      if e.val+s(e,a)+w(a)<a.val
19          a.val=e.val+s(e,a)+w(a);
20          a.prevArc=e;
21      if s(e,a)>a.minCost
22          outArcs=outArcs union a;
23  e.head.outArcs=outArcs;
    
```

The time bound of this algorithm is same as that of the extended Dijkstra algorithm. But in many situations this optimized algorithm can significantly reduce the number of expanded arcs in the extended algorithm. This can be shown in the experiment part. The algorithm will be named as O-E-Dijkstra algorithm in the following.

$v_1(0,0)$	$v_1(0,0)$	$e_1(0,3)$
$v_2(e_2,3)$	$v_2(e_2,3)$	$e_2(0,3)$
$v_3(e_1,3)$	$v_3(e_1,3)$	$e_3(e_2,7)$
$v_4(e_4,7)$	$v_4(e_4,12)$	$e_4(e_3,12)$
(a) Standard	(b) Extended	

Figure 3: Label results for the two algorithms. In (a), $v_4(e_4, 7)$ means $v_4.prevArc = e_4, v_4.val = 7$. In (b), $v_4(e_4, 12)$ means $v_4.selectedArc = e_4, v_4.val = 12$. $e_4(e_3, 12)$ means $e_4.prevArc = e_3, e_4.val = 12$.

4 Application

We'll take transportation network as example. In transportation network, each node represents a train station or airdrome, and each arc has an ID to represent its train/plane number. Each arc has a departure time and an arrival time. There may be multiple arcs between two nodes.

In the transportation network, the transferring times and the waiting time in stations are two key factors which influence people's decisions when they select travel pathes. These two factors can be summarized as switch cost between arcs in transportation network.

The standard shortest path problem is unable to handle the switch cost, so the result given by classic shortest path algorithm may be unacceptable. In the following section, some examples will be shown to demonstrate this. While our algorithms are used to solve the extended shortest path problem which can integrate switch cost, they can solve this problem easily.

A *struct SVal* is used to represent both weight cost and

switch cost. In this scenario, *SVal* contains two fields: *transfers* and *time*. *SVal* can be written as a pair (*transfers, time*).

To compute the extended shortest path, we need provide a comparison method between two *struct SVals*. Different methods represent different need. Two methods are provided as follows:

Method 1:

```

int Compare(SVal val1, SVal val2)
    if (val1.transfers != val2.transfers)
        return sign(val1.transfers - val2.transfers);
    else return sign(val1.time - val2.time);
    
```

Method 2:

```

int Compare(SVal val1, SVal val2)
    v1 = val1.transfers * w + val1.time;
    v2 = val2.transfers * w + val2.time;
    return sign(v1 - v2);
    
```

Method 1 means that people care more about number of transfers. Method 2 is more general. Here w is a factor, which reflects the importance of transfers. For example, $w = 2 \text{ hours/per transfer}$ means one transfer equals to 2 hours. $w = 0$ means people doesn't care about transfers. When $w \rightarrow \infty$, method 2 is equivalent to method 1. Since method 1 is more intuitional, we'll use this in the following parts.

For each arc e , $w(e) = (0, e.arrivalT - e.departureT)$. For each arc pair $e_i \rightarrow e_j$, the switch cost $s(e_i, e_j)$ is defined as follows:

$$\begin{cases} (0, e_2.departureT - e_1.arrivalT) & \text{if } e_i.id = e_j.id \\ (1, e_2.departureT - e_1.arrivalT) & \text{if } e_i.id \neq e_j.id \end{cases}$$

Then we can use our algorithms to compute the extended shortest path on this model. The experiment bellow will show that our algorithms can give acceptable results which the classic ones solve.

The *MNTP* problem can be solved by using switch cost mentioned above. The *EAP* problem can be solved when the transferring times is ignored. Since these problems are not the foci of this paper, we won't discuss them in detail.

5 Experiment

5.1 Part One

In this section, we'll compare the performance of our algorithms with the standard Dijkstra algorithm with different types of switch cost.

5.1.1 Experiment data

The experiment data used is part of the train transportation graph of China. This graph contains 3067 nodes and 31368 arcs.

5.1.2 Experiment results


For each arc pair $e_i \rightarrow e_j$, we define three different types of switch cost $s(e_i, e_j)$ as follows:

- * Type 0: $s(e_i, e_j) = (0, 0)$
- * Type 1: $s(e_i, e_j) = (0, e_j.departureT - e_i.arrivalT)$
- * Type 2: switch cost mentioned in Section 4.

The experiment is shown in table 1. We run the single source problem for each vertex and compute the average run time and number of expanded arcs for each algorithms. From the experiment result we can find that:

- * Our algorithms need more run time than Dijkstra algorithm. But the time cost is worth handling switch cost which the Dijkstra algorithm can't finish.
- * Our O-E-Dijkstra algorithm can significantly reduce the number of expanded arcs in E-Dijkstra algorithm.
- * When the switch cost is zero, the number of expanded arcs in O-E-Dijkstra algorithm is equal to that in Dijkstra algorithm.

5.2 Part Two

We use the standard Dijkstra algorithm and the E-Dijkstra algorithm to compute the shortest path between two given stations or airdromes of China. Fig 4 shows the results on the train transportation network and Fig 5 shows the results on the air transportation network. In these pictures,  represents a transfer station/airdrome. Since the standard Dijkstra algorithm can't integrate the switch cost, the results it gives need a lot of transfers. Another more serious problem is that the time the standard Dijkstra algorithm gives is not the real time, because it doesn't take into account the waiting time in stations/airdromes. So according to the path computed by the standard Dijkstra, people might wait for more than one day to take another train or plane. Obviously, it is unacceptable.

6 Conclusion

We proposed a general framework by integrating switch cost between arcs. The proposed algorithms can handle many real problems with switch cost, such as train transportation network, bus transportation network and so on. The E-Dijkstra algorithm and the O-E-Dijkstra algorithm can solve the single source extended shortest path problem while the standard Dijkstra algorithm can't. The E-Dijkstra algorithm may expand huge number of arcs while the O-E-Dijkstra algorithm can significantly reduce the number of expanded arcs. Our algorithms can give acceptable result in most of applications where switch cost exists.

References

- [1] Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. ACM Journal of Experimental Algorithmics, Vol. 5, No. 12(2000)
- [2] Brodal, G.S., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. Proc. Algorithmic Methods and Models for Optimization of Railways. Electronic Notes in Theoretical Computer Science, Vol. 92. Elsevier(2003)
- [3] Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. DELIS Technical Report(2004)
- [4] Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. Algorithmic Methods for Railway Optimization, LNCS. Springer(2005)
- [5] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1959) 269–271
- [6] Cherkassky, B.V., Goldberg, A.V., Radzik, T.: Shortest Paths Algorithms: Theory and Experimental Evaluation. Proc. 5th ACM-SIAM Symposium on Discrete Algorithms(1994) 516–525
- [7] Cherkassky, B.V., Goldberg, A.V., Silverstein, C.: Buckets, heaps, lists, and monotone priority queues. Proc. 8th annual ACM-SIAM symposium on Discrete algorithms(1997) 83–92
- [8] Goldberg, A.V.: Shortest path algorithms: Engineering aspects. Proc. International Symposium on Algorithms and Computation, Vol. 2223. Springer(2001) 502–513
- [9] Goldberg, A.V., Harrelson, C.: Computing the shortest path: A* meets graph theory. Proc. 16th ACM-SIAM Symposium on Discrete Algorithms. ACM Press, New York (2005) 156–165

transportation network nodes/arcs	type of switch cost	Dijkstra average run time (in second) average number of expanded arcs	E-Dijkstra	O-E-Dijkstra
3067/31368	type 0	0.0054 31327	0.0847 1549020	0.0333 31327
	type 1	0.0057 31327	0.2156 1549020	0.1647 784289
	type 2	0.0060 31327	0.2999 1549020	0.2154 786930

Table 1: Experiment results on dense graph and sparse graph. The program is written in C++ and ran on a server with 3.2GHZ processor running Windows Server 2003.

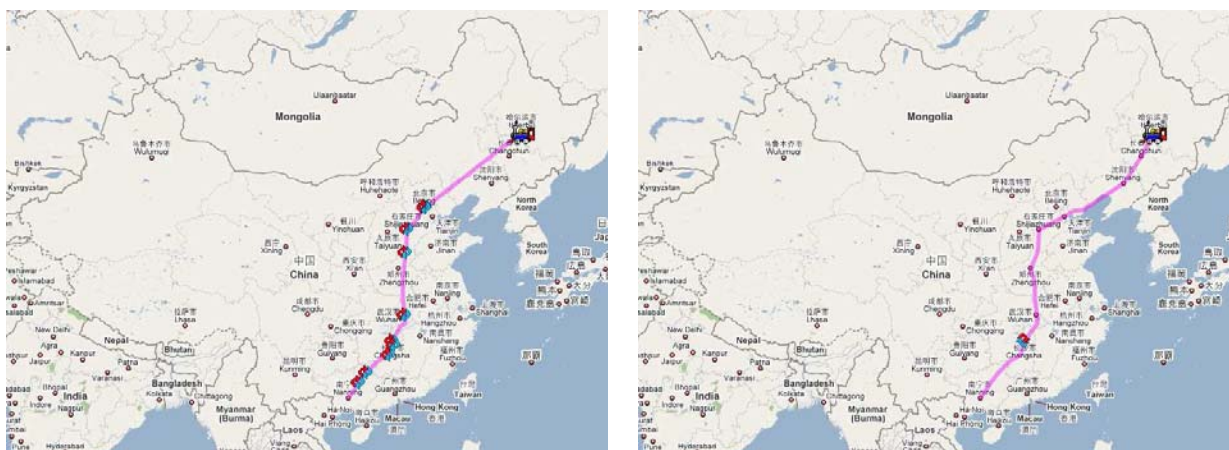


Figure 4: Left: shortest path computed by the standard Dijkstra algorithm on train transportation network. Right: shortest path computed by the extended algorithm on the same network. Obviously, the result on the left has too many transfers to be accepted.

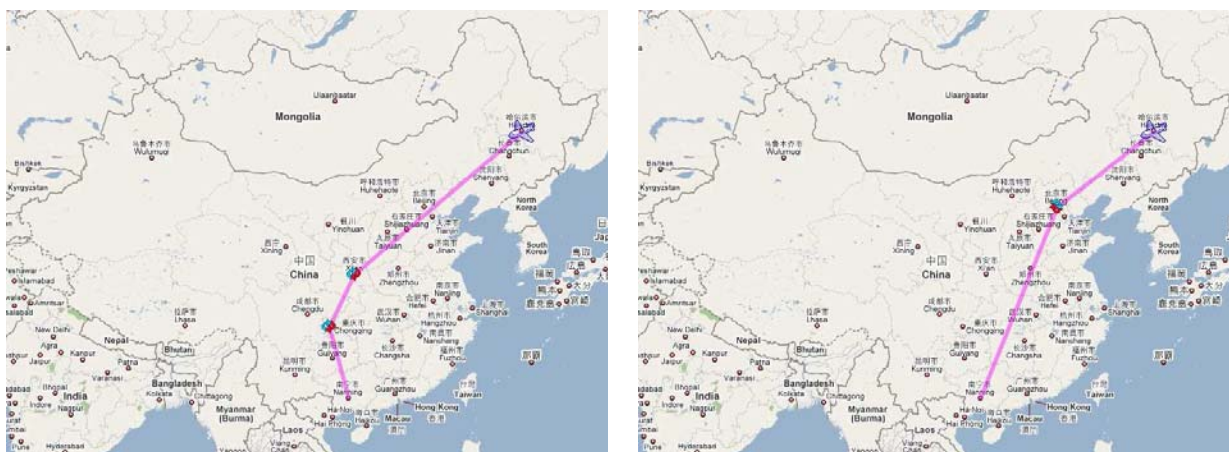


Figure 5: Left: shortest path computed by the standard Dijkstra algorithm on air transportation network. Right: shortest path computed by the extended algorithm on the same network. Obviously, the result on the left has too many transfers to be accepted.

[10] Sanders, P., Schultes, D.: Engineering highway hierarchies. ESA 2006. LNCS, Vol. 4168. Springer, Heidelberg(2006) 804–816

[11] Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science(2007)