文章编号: 2096-1618(2021)02-0162-07

A Coherent Approach for Transportation Network Analysis

XIANG Wenli
(Hubei Airports Group Company Limited, Wuhan 430320, China)

Abstract: This paper presents a coherent approach to the analysis of transportation network based on the concept of Minimum Spanning Tree algorithm and Dijkstra's algorithm. Network connectivity is an important aspect of any transportation network, the role of connectivity is to provide a connection to possibly travel from point A to point B by using various modes. The importance of node is, for example, greatly contribute to short connections between nodes, handle a large amount of the traffic, generate relevant information. In order to quantify the relative importance of nodes, we use two algorithm. The first one is Minimum Spanning Tree algorithm, we use it to find the shortest route between two point. The second one is Dijkstra's algorithm we use it to find the route with lowest cost between two point.

Keywords: transportation network; nodes; connectivity; Minimum Spanning Tree algorithm; Dijkstra's algorithm
CLC number: TP393.02 **Document code**: A
doi: 10.16836/j.cnki.jcuit.2021.02.007

1 INTRODUCTION

Typically, transportation network means we make the spatial network as real to consider vehicular movement's structure. We first introduce network connectivity. The analysis of network connectivity can assist decision makers in identifying weak components, detecting and preventing failures, and improving connectivity in terms of decreased travel time, reduced costs, increased reliability, and accessibility. Then we discuss about Minimum Spanning Tree algorithm. A minimum spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. We introduce the possibility multiplicity, uniqueness, cut property of minimum spanning tree and describe two major algorithm, Kruskal's algorithm and Prim's algorithm to analysis the transportation network. Third, we discuss about Dijkstra's algorithm. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. We introduce the description, algorithm, running time of Dijkstra's algorithm. Finally, we describe the input data of two algorithm and get the program results, we analyze the results and get out conclusion.

2 DESCRIPTION OF CONNECTIVITY

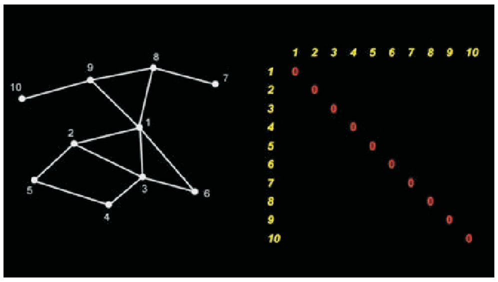
Connectivity typically means the transportation degree of the connection degree. Simply,

High connectivity = high accessibility and low isolation

Low connectivity = low accessibility and high isolation

It is important when we try to find the shortest route, since connectivity is a measures do not consider of distance.

If two locations are connected directly, we code with a1, else we code with a0. Then we need to reduce the transportation network to matrix format within a1 and a0. We assume all the distance as topology distance as 1. And number the vertices and create a matrix (Fig. 1).

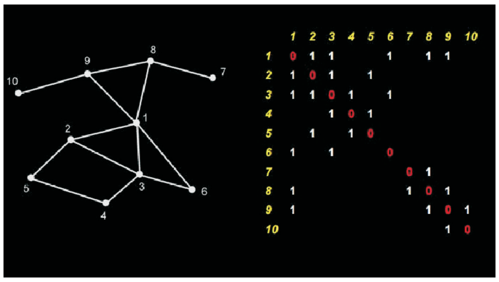


$$\begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 1 & 0 & & & & & & & & & \\ 2 & & 0 & & & & & & & & \\ 3 & & & 0 & & & & & & & \\ 4 & & & & 0 & & & & & & \\ 5 & & & & & 0 & & & & & \\ 6 & & & & & & 0 & & & & \\ 7 & & & & & & & 0 & & & \\ 8 & & & & & & & & 0 & & \\ 9 & & & & & & & & & 0 & \\ 10 & & & & & & & & & & 0 \end{array}$$

Fig. 1 Matrix create

Then we directly coding connection as 1 and using the matrix (the lower half of matrix). And we simply copy the lower half of the matrix to the rest part of matrix to finish (Fig. 2).

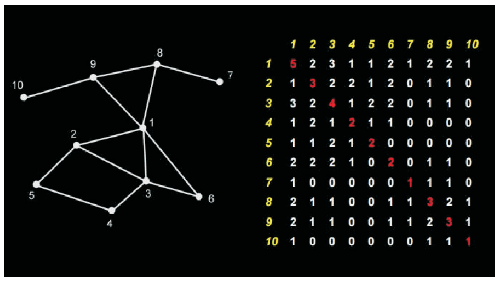
收稿日期: 2020-12-29



	1	2	3	4	5	6	7	8	9	10
1	0	1	1			1		1	1	
2	1	0	1		1					
3	1	1	0	1		1				
4			1	0	1					
5		1		1	0					
6	1		1			0				
7							0	1		
8	1						1	0	1	
9	1							1	0	1
10									1	0

Fig. 2　Matrix copy

We power the matrix to determine 2 steps linkage, we will do the process for all the cells in matrix, then we can get the results, the matrix now can be seen as all possible 2 step combination. And we added the two matrix, the result matrix can represent the whole step routes (one and two steps). We will continue to power the matrix again and repeat the whole process (Fig. 3).



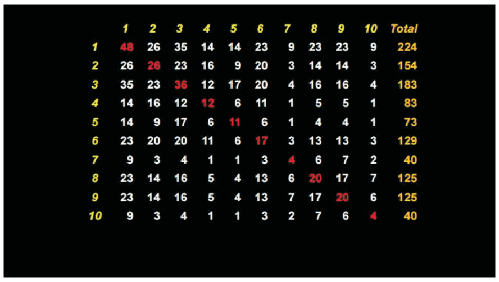
	1	2	3	4	5	6	7	8	9	10
1	5	2	3	1	1	2	1	2	2	1
2	1	3	2	2	1	2	0	1	1	0
3	3	2	4	1	2	2	0	1	1	0
4	1	2	1	2	1	1	0	0	0	0
5	1	1	2	1	2	0	0	0	0	0
6	2	2	2	1	0	2	0	1	1	0
7	1	0	0	0	0	0	1	1	1	0
8	2	1	1	0	0	1	1	3	2	1
9	2	1	1	0	0	1	1	2	3	1
10	1	0	0	0	0	0	0	1	1	1

Fig. 3　Process repeat

The diagonal represents all routes from a vertex back to itself. Finally, we will power the matrix then add the matrix again and again until we fill all the zero cells. The time we need to make all empty cells full, that is named Diameters.

Diameter = the fewest number of step needed to connect the vertices which are the farthest apart topologically.

We can see the matrix now is includeall the connectivity matrix, plus the all the total rows gives the connectivity for every vertex (Fig. 4).



	1	2	3	4	5	6	7	8	9	10	Total
1	48	26	35	14	14	23	9	23	23	9	224
2	26	26	23	16	9	20	3	14	14	3	154
3	35	23	36	12	17	20	4	16	16	4	183
4	14	16	12	12	6	11	1	5	5	1	83
5	14	9	17	6	11	6	1	4	4	1	73
6	23	20	20	11	6	17	3	13	13	3	129
7	9	3	4	1	1	3	4	6	7	2	40
8	23	14	16	5	4	13	6	20	17	7	125
9	23	14	16	5	4	13	7	17	20	6	125
10	9	3	4	1	1	3	2	7	6	4	40

Fig. 4　Matrix finished

Now we get the results for connectivity matrix results as:

(1) 7with 40 connections means the least accessible node.

(2) 1 with 224 connections means the most accessible node.

3　INTRODUCTION FOR MINIMUM SPANNING TREE ALGORITHM

Minimum Spanning Tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching. Other practical applications are: Cluster Analysis, Handwriting recognition, Image segmentation.

We know that a given graph will have some different spanning trees, each edge will have a weight, that is the number meaning the unfavorable. We familiar with getting the weight to a spanning trees by calculate the all the weights' sum. Minimum Spanning Tree means the weight of all the spanning trees is larger than a spanning tree's weight.

In a given undirected graph $G=(V,H)$, the (u,v) represents the edge between vertex u and vertex v, $(u,v) \in E$, $\omega(u,v)$ means the weight of the edge. If we can find $T(T\subseteq E)$, then we can find the minimum value of $\omega(T)$, when:

$$\omega(T)=\sum_{(u,v)\in T}\omega(u,v)$$

This is a Minimum Spanning Tree. We can see that every edge is labeled and the weight is roughly proportional to length (Fig. 5).

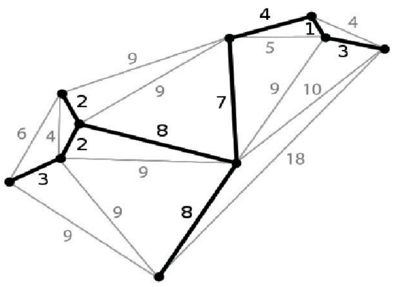


Fig. 5　Minimum Spanning Tree

Now we introduce some properties of Minimum Spanning Tree.

(1) Possible multiplicity

We will find some different Minimum Spanning Tree with the same weight having the same minimum number of edges. If we have the vertices in the graph, every spanning tree's with just one minus edges.

(2) Uniqueness

If we do not have the unique edge weights, then the set of the weight of minimum spanning is unique. It is all the same for all Minimum Spanning Trees.

We know that sometimes will get more than one Minimum Spanning Tree. From Fig. 6 the two trees below the graph show two different given graph's Minimum Spanning Tree's possibility.

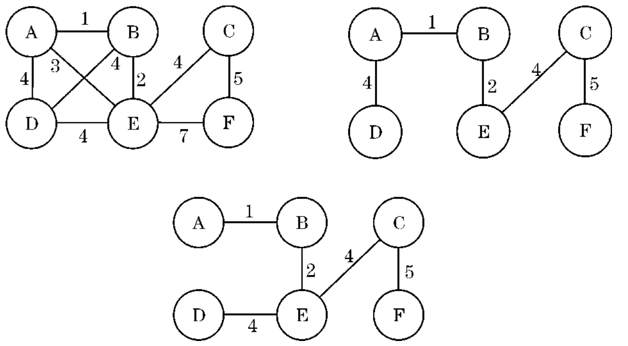


Fig. 6 Two different given Graph's Minimum Spanning Tree's possibility

(3) Cut property

In the graph, every cut of C, assume the weight of all of the edges of C is larger than the weight of an edges of C, we can conclude that this edge belongs to all minimum panning trees of the graph.

We can see that the Minimum Spanning Tree's cut property. T is the only Minimum Spanning Tree of the graph. Assume $S=\{A,B,C,D\}$, then $V\text{-}S=\{C,F\}$, then we can get three possibilities of the edge of the cut(S, V-S), it is edges BC, EC, EF of the original graph (Fig. 7).

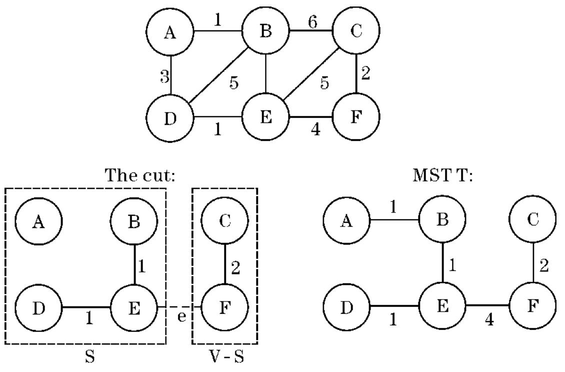


Fig. 7 Three possibilities of the edge

There are two famous algorithms for finding the Minimum Spanning Tree.

(1) Kruskal's Algorithm

KIruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps:

(i) Sort the graph edges with respect to their weights.

(ii) Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.

(iii) Only add edges which doesn't form acycle, edges which connect only disconnected components.

The vertices are connected or not can be checked by using DFS which starts from the first vertex is visited or not. But DFS will make time complexity large as it has an order of $O(V+E)$ where O is the number of vertices, E is the number of edges. So the best solution if "Disjoint Sets":

Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.

Consider following example:

In Kruskal's algorithm, at each iteration we will select the edge with the lowest weight. So, we will start with the lowest weighted edge first i.e., the edges with weight 1. After that we will select the second lowest weighted edge i.e., edge with weight 2. Notice these two edges are totally disjoint. Now, the next edge will be the third lowest weighted edge i.e., edge with weight 3, which connects the two disjoint pieces of the graph. Now, we are not allowed to pick the edge with weight 4, that will create a cycle and we can't have any cycles. So we will select the fifth lowest weighted edge i.e., edge with weight 5. Now the other two edges will create cycles so we will ignore them. In the end, we end up with a Minimum Spanning Tree with total cost 11 (= 1 + 2 + 3 + 5) (Fig. 8, 9).

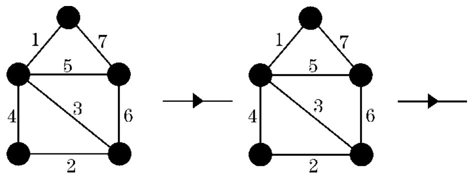


Fig. 8 Start with the lowest weighted edge

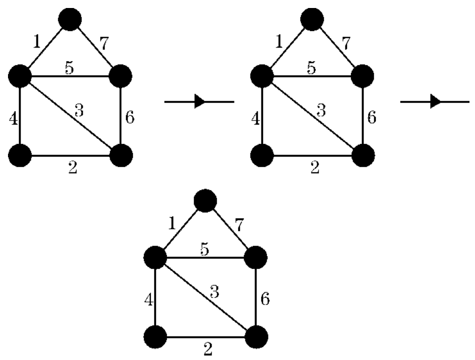


Fig. 9 Minimum Spanning Tree with total cost

Time Complexity:

In Kruskal's algorithm, most time consuming operation is sorting because the total complexity of the Disjoint-Set operations will be $O(E\lg V)$, which is the overall Time Complexity of the algorithm.

(2) Prim's Algorithm

Prim's Algorithm alsouse Greedy approach to find the Minimum Spanning Tree. In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an edge in Kruskal's, we add vertex to the growing spanning tree in Prim's.

Algorithm Steps:

(i) Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.

(ii) Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert thevertices, that are connected to growing spanning tree, into the Priority Queue.

(iii) Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

Consider the example below:

In Prim's Algorithm, we will start with an arbitrary node (it doesn't matter which one) and mark it. In each iteration we will mark a new vertex that is adjacent to the one that we have already marked. As a greedy algorithm, Prim's algorithm will select the cheapest edge and mark the vertex. So we will simply choose the edge with weight 1. In the next iteration we have three options, edges with weight 2, 3 and 4. So, we will select the edge with weight 2 and mark the vertex. Now again we have three options, edges with weight 3, 4 and 5. But we can't choose edge with weight 3 as it is creating a cycle. So we will select the edge with weight 4 and we end up with the Minimum Spanning Tree of total cost 7 (= 1 + 2 +4) (Fig. 10).

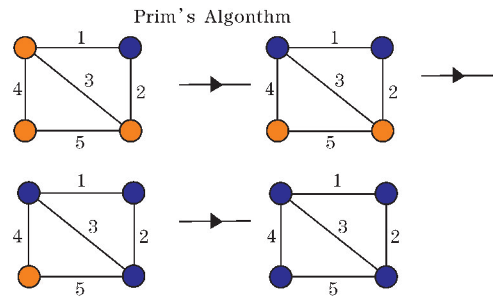


Fig. 10　Select the edge with weight 4, get total cost

Time Complexity:

The time complexity of the Prim's Algorithmis $O((V+E)\lg V)$, because each vertex is inserted in the priority queue only once and insertion in priority queue take logarithmic time.

4　INTRODUCTION FOR DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. The key purpose we use Dijkstra's algorithm is to find the shortest paths between nodes in graph. Especially, Dijkstra's original variant can be used to find the shortest path between two nodes. Besides, there still has a single node with common variant call the source node, finding the shortest paths from source to all other nodes can make the shortest-path tree.

4.1　Description

Dijkstra's algorithm uses a data structure for storing and querying partial solutions sorted by distance from the start. While the original algorithm uses a min-priority queue and runs in time $\Theta(|V|+|E|\lg|V|)$, (where $|V|$ is the number of nodes and $|E|$ is the number of edges), it can also be implemented in $\Theta(|V|^2)$ using an array. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

Djikstra's algorithm can find the shortest path between a and b. We choose the vertex with lowest distance, compute the distance from it to other neighbor. If it is similar, we update its distance (Fig. 11).

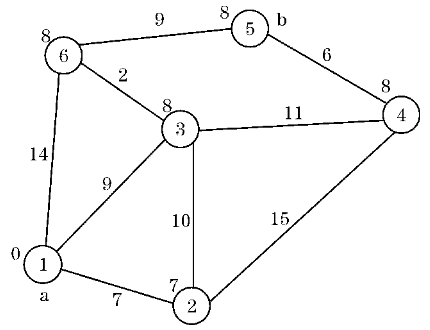


Fig. 11　The shortest path between a and b

4.2　Algorithm

In robot motion planning problem, we describe Dijstra's algorithm search for finding path from the start node to goal node. From the figure 12, the open node means that the tentative set and the filled node means the visit one. And the color represent the distance between node for filled node. More green means more distances. Different direction nodes are uniformly, as a circular wavefront in Dijstra's algorithm by using heuristic identically equal to 0.

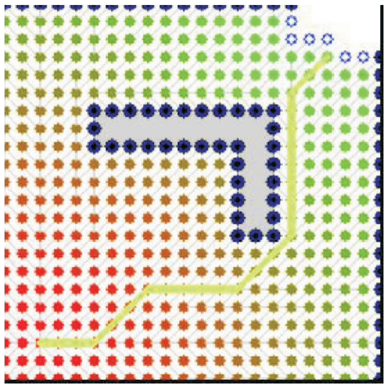


Fig. 12 The shortest path

4.3 Running time

Bounds of the running time Dijkstra's algorithm on a graph with edges E and vertice V can be expressed as a function of the number of edges, denoted $|E|$, and the number of vertices denoted $|V|$, using big-O notation. The complexity depends mainly on the data structure used to represent the set Q. In the following, upper bounds can be simplified because $|E|$ is $\Theta(|V|^2)$ for any graph, but that simplification disregards the fact that in some problems other upper bounds on $|E|$ may hold.

For any data structure for the vertexset Q, the running time is in: $\Theta(|E| \cdot T_{\mathrm{dk}} + |V| \cdot T_{\mathrm{em}})$, where T_{dk} and T_{dm} are the complexities of the decrease-key and extract-minimum operation in Q, respectively. The simplest version of Dijkstra's algorithm stores the vertex set Q as an ordinary linked or array, and extract-minimum is simply a linear search through all vertices in Q as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in Q.

In this case, the running time is: $\Theta(|E|+|V|^2) = \Theta(|V|^2)$

If the graph is stored as an adjacency list, the running time for a dense graph is: $\Theta(|V|^2 \lg |V|)$

For sparse graphs, that is, graphs with far fewer than $|V|^2$ edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a self-balancing binary search tree, binary heap, pairing heap, or Fibonacci heap as a priority queue to implement extracting minimum efficiently. To perform decrease-key steps in a binary heap efficiently, it is necessary to use an auxiliary data structure that maps each vertex to its position in the heap, and to keep this structure up to date as the priority queue Q changes. With a self-balancing binary search tree or binary heap, the algorithm requires: $\Theta(|E|+|V| \lg |V|)$

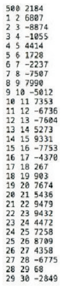
When using binary heaps, the average case time complexity is lower than the worst-case. Assuming edge costs are drawn independently from a common probability distribution, the expected umber of decrease-key operation is bounded by $\Theta(|V| \lg (|E|/|V|))$, giving a total running time of: $O(|E|+|V| \lg (|E|/|V|) \lg |V|)$.

5 RESULT BASED ON MINIMUM SPANNING TREE ALGORITHM

We use Minimum Spanning Tree algorithm to solve the problem. We will code up Prim's Minimum Spanning Tree algorithm.

5.1 Introduction input data

We use the text file describe an undirected graph with integer edge (Fig. 13). For example, the first line of the file is (1, 2, 6807), indicates that the route between point 1 and point 2 is cost 6807. And there is 500 points in total.



```
500 2184
1 2 6807
2 3 -8874
3 4 -1055
4 5 4414
5 6 1728
6 7 -2237
7 8 -7507
8 9 7990
9 10 -5012
10 11 7353
11 12 -6736
12 13 -7604
13 14 5273
14 15 9331
15 16 -7753
16 17 -4370
17 18 267
18 19 903
19 20 7674
20 21 5436
21 22 9479
22 23 9432
23 24 4472
24 25 7258
25 26 8709
26 27 4358
27 28 -6775
28 29 68
29 30 -2849
```

Fig. 13 Undirected graph

We do not assume that edge costs are positive, nor should we assume that they are distinct. we task is to run Prim's Minimum Spanning Tree algorithm on this graph. We will report the overall cost of a Minimum Spanning Tree—an integer, which may or may not be negative—in the box window.

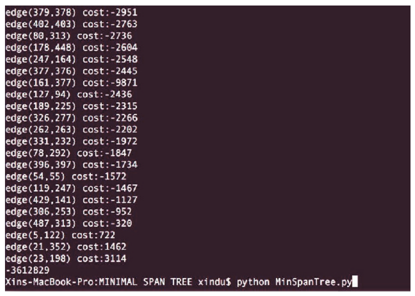
The graph is small enough that the straight forward

O time implementation of Prim's algorithm should work fine. For those of our seeking an additional challenge, we try to implement a heap-based version. The simpler approach, which should already give us a healthy speed up, to maintain relevant edges in a heap. The superior approach stores the unprocessed vertices in the heap.

The purpose of this program is to choose some parts of the routes to lower the cost.

5.2 Program results

The result shows the whole routs we need to choose (Fig. 14).



```
edge(379,378) cost:-2951
edge(402,403) cost:-2763
edge(80,313) cost:-2736
edge(178,448) cost:-2604
edge(247,164) cost:-2548
edge(377,376) cost:-2445
edge(161,377) cost:-9871
edge(127,94) cost:-2436
edge(189,225) cost:-2315
edge(326,277) cost:-2266
edge(262,263) cost:-2202
edge(331,232) cost:-1972
edge(78,292) cost:-1847
edge(396,397) cost:-1734
edge(54,55) cost:-1572
edge(119,247) cost:-1467
edge(429,141) cost:-1127
edge(306,253) cost:-952
edge(487,313) cost:-320
edge(5,122) cost:722
edge(21,352) cost:1462
edge(23,198) cost:3114
-3612829
Xins-MacBook-Pro:MINIMAL SPAN TREE xindu$ python MinSpanTree.py
```

Fig. 14　The whole routs we need to choose

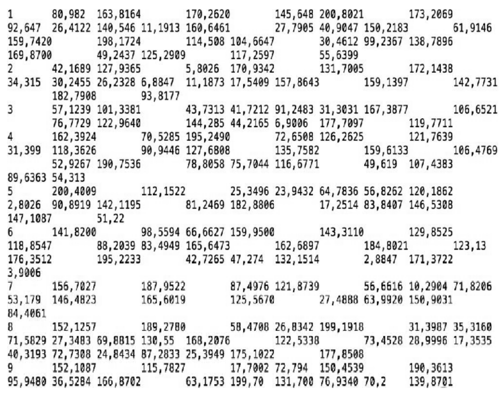
Forexample, edge (379, 378) cost: −2951 means that the route between point 379 and point 378 is necessary and the cost for this route is −2951. We also get the total minimum cost is −3612829.

6 RESULT BASED ON DIJKSTRA'S ALGORITHM

We use Dijkstra's shortest-path algorithm to find the shortest path.

6.1 Introduction input data

We use the undirected weighted graph file with 200 vertices labeled 1 to 200 with an adjacency list representation. Each node in the row are adjacent to that particular vertex along with the length of that edge. For example, we look at the 6th row. It shows that it has "6" as the first entry means that this row corresponds to the vertex labeled 6. The other part of this row shows (141, 8200) means that there is an edge between vertex 6 and vertex 141 and with length 8200. The rest parts of the row shows that the vertex which adjacent to vertex 6, and indicates the length of the corresponding edges (Fig. 15).



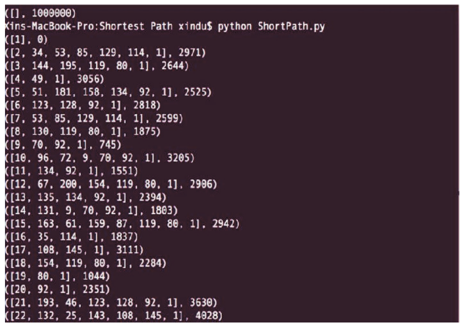
```
1        80,982   163,8164          170,2620          145,648  200,8021          173,2069
92,647   26,4122  140,546  11,1913  160,6461          27,7905  40,9047  150,2183          61,9146
159,7420          198,1724          114,508  104,6647          30,4612  99,2367  138,7896
169,8700          49,2437  125,2909          117,2597          55,6399
2        42,1689  127,9365          5,8026   170,9342          131,7005          172,1438
34,315   30,2455  26,2328  6,8847   11,1873  17,5409  157,8643          159,1397          142,7731
         182,7908          93,8177
3        57,1239  101,3381          43,7313  41,7212  91,2483  31,3031  167,3877          106,6521
         76,7729  122,9640          144,285  44,2165  6,9006   177,7097          119,7711
4        162,3924          70,5285  195,2490          72,6508  126,2625          121,7639
31,399   118,3626          90,9446  127,6808          135,7582          159,6133          106,4769
         52,9267  190,7536          78,8058  75,7044  116,6771          49,619   107,4383
89,6363  54,313
5        200,4009          112,1522          25,3496  23,9432  64,7836  56,8262  120,1862
2,8026   90,8919  142,1195          81,2469  182,8806          17,2514  83,8407  146,5308
147,1087          51,22
6        141,8200          98,5594  66,6627  159,9500          143,3110          129,8525
118,8547          88,2039  83,4949  165,6473          162,6897          184,8021          123,13
176,3512          195,2233          42,7265  47,274   132,1514          2,8847   171,3722
3,9006
7        156,7027          187,9522          87,4976  121,8739          56,6616  10,2904  71,8206
53,179   146,4823          165,6019          125,5670          27,4888  63,9920  150,9031
84,4061
8        152,1257          189,2780          58,4708  26,8342  199,1918          31,3987  35,3160
71,5829  27,3483  69,8815  130,55   168,2076          122,5338          73,4528  28,9996  17,3535
40,3193  72,7308  24,8434  87,2833  25,3949  175,1022          177,8508
9        152,1087          115,7827          17,7002  72,794   150,4539          190,3613
95,9480  36,5284  166,8702          63,1753  199,70   131,700  76,9340  70,2     139,8701
```

Fig. 15　The length of the corresponding edges

We plan to compute the shortest-path distance between 1 and all the other vertex and treated it as the source vertex. If we can get no path between a vertex 1 and vertex *v*, we will set the distance between the shortest-path to be very large.

6.2 Program results

The result shows the shortest path for every 200 points to point 1. For example, ((22, 132, 25, 143, 188, 145, 1) 4828) means that if point 1 need to be connected to point 22, it should be from poin (145, 188, 143, 25, 132, 22) one by one based on our algorithm to make sure the shortest path and the required distance is 4828 (Fig. 16).



```
([], 1000000)
Xins-MacBook-Pro:Shortest Path xindu$ python ShortPath.py
([1], 0)
([2, 34, 53, 85, 129, 114, 1], 2971)
([3, 144, 195, 119, 80, 1], 2644)
([4, 49, 1], 3056)
([5, 51, 181, 158, 134, 92, 1], 2525)
([6, 123, 128, 92, 1], 2818)
([7, 53, 85, 129, 114, 1], 2599)
([8, 130, 119, 80, 1], 1875)
([9, 70, 92, 1], 745)
([10, 96, 72, 9, 70, 92, 1], 3205)
([11, 134, 92, 1], 1551)
([12, 67, 200, 154, 119, 80, 1], 2906)
([13, 135, 134, 92, 1], 2394)
([14, 131, 9, 70, 92, 1], 1803)
([15, 163, 61, 159, 87, 119, 80, 1], 2942)
([16, 35, 114, 1], 1837)
([17, 108, 145, 1], 3111)
([18, 154, 119, 80, 1], 2284)
([19, 80, 1], 1044)
([20, 92, 1], 2351)
([21, 193, 46, 123, 128, 92, 1], 3630)
([22, 132, 25, 143, 108, 145, 1], 4028)
```

Fig. 16　The shortest path and the required distance

7 SUMMARY AND CONCLUSION

In order to analysis the transportation network, we use two algorithms to help us fulfill two major purpose. To decrease the cost as most as possible, we use Minimum Spanning Tree algorithm. To find the shortest path from point to point, we use Dijkstra's algorithm. Connect to transportation network analysis, we believe that these tows algorithms helps. Simply, we use Minimum Spanning Tree algorithm when we need to consider the cost to connect two cities. And we use Dijkstra's algorithm when we care about the length of path mostly.

REFERENCE:

[1] 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.

[2] Gabow H N, Galil Z, Spencer T, et al. Efficient algorithms for finding Minimum Spanning Trees in undirected and directed graphs[J]. Combinatorica,1986,6(2):109.

[3] Chong Kawong, Han Yijie, et al. Concurrent threads and optimal parallel Minimum Spanning Trees algorithm[J]. Journal of the Association for Computing Machinery,2001,48(2).

[4] Bader David A, Cong Guojing. Fast shared-memory algorithms for computing the minimum spanning forest of sparsegraphs[J]. Journal of Parallel and Distributed Computing, 2006, 66 (11): 1366 - 1378.

[5] Dahlhaus E, Johnson D S, Papadimitriou C H, et al. The complexity of multiterminal cuts[J]. SIAM Journal on Computing,1994,23(4):864-894.

[6] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem[C]. Graduate School of Industrial Administration, CMU,1976.

[7] Fredman Michael Lawrence, Tarjan Robert E. Fibonacci heaps and their uses in improved network optimization algorithms[J]. 25th Annual Symposium on Foundations of Computer Science. IEEE. 1984.

[8] Fredman Michael Lawrence, Tarjan Robert E. Fibonacci heaps and their uses in improved network optimization algorithms[J]. Journal of the Association for Computing Machinery,1987,34(3):596-615.

[9] Zhan F Benjamin, Noon Charles E. Shortest Path Algorithms: An Evaluation Using Real Road Networks[J]. Transportation Science,1998,32(1): 65-73.

传输网络分析

向文力
(湖北机场集团有限公司,湖北 武汉 430320)

摘要:分别介绍了基于最小生成树算法和 Dijkstra 算法的传输网络分析方法。通过引入传输网络的概念,分析了网络连通性。网络连通性是传输网络中的重要一环,其作用在于提供 A 点到 B 点间的多种模式连接。通过对网络连通性的分析,可极大地缩短节点间的连接路径,加大流量处理并生成相关信息。通过使用最小生成树算法和 Dijkstra 算法,将传输网络中的节点信息进行了量化处理。其中,通过最小生成树算法,分析、筛选出了连接首末两点间的最短路径;通过 Dijkstra 算法,分析、筛选出了连接首末两点间成本最低的路径。

关　键　词:传输网络;节点;连接性;最小生成树算法;Dijkstra 算法