The Traveling Salesman Problem Using Minimum Spanning Tree Approach

Zhao Zi Chao, Wen Jia Heng, Arun Kumar Sangaiah

Abstract— **The Traveling Salesman Problem (TSP) is a widely mentioned classic case in data structure and algorithm, which can be solved in many methods. The idea was first aroused in a brochure, mentioned a problem of a salesman travelling through Germany and Swiss. At that time, the problem remained unclear and came up with no mathematical methods. The task allows a list of cities and their pair-wise distances, aiming to find out a shortest route, which visited each city only once, in order to save time and money. In this paper we are planning to consider the solution of Minimum Spanning Tree (MST) on a complete graph G = (V, E) with n=10 vertices. We have listed 10 different cities in China and aimed to find out a best route for the customers based on their inputs about which cities are the final and shortest destinations. We shall consider the method for solving the problem: first, using Minimum Spanning Tree in Heuristic Searching Algorithm, to find the shortest path. Mark the coordinates of each city and turn the statistics into distances (kilometers). Add a group of input figures, such as the number of cities visited and which is the beginning location. Output the best route for the clients eventually.**

Index Terms—**Travelling Salesman Problem, Minimum Spanning Tree, Prim's algorithm**

I. INTRODUCTION

The TSP problem is a combinatorial optimization problem. This problem can be proved to have NP (nondeterministic polynomial time) computational complexity. Therefore, anything that make the problem's solving method simplified, are highly evaluated. Traveling salesman problem is one of the most prominent problems in graph theory, that is, " given 'n' points of complete graph, each edge has a length, and the total length of the shortest after each vertex is just a closed loop". Moreover the minimum cost spanning tree has been applied different fields of application. In this paper, minimum spanning tree has used to find the minimum distance for travelling all cities from source to destination at most once. In addition, minimum cost spanning tree has been widely implements via two methods: prim's algorithm and kruskal's algorithm. Thus, our objective is to find the shortest route among source and destination we have used the combination of minimum spanning tree and prim's algorithm for TSP problem.

Manuscript received Sep 02, 2014

Zhao Zi Chao, Wen Jia Heng School of Information & Software Engineering, University of Electronic Science and Technology of China

Arun Kumar Sangaiah, School of Computing Science and Engineering, VIT University, Vellore, India

II. Related Literature

Traveling salesman problem, as a representative of the typical NP problem, has been a hot topic in the theoretical research of computer algorithm since introduced, and all kinds of algorithms emerge in endlessly for this problem. It has always been a focus in the study of the industry; its application range is wide, with important guiding significance in many areas. Objectively speaking, it does not exist an optimal algorithm for TSP problem currently, each algorithm has its deficiencies, for classical algorithm pursues the accuracy of the answer and ignores the consumption of time and space, while modern popular ones seek for approximate solution, but are unacceptable on results to some extent. In the future, study on this algorithm should grasp the three aspects: continue to improve the existing algorithms, adopt the idea of artificial intelligence, create new TSP algorithm. In addition, the significance of TSP in optimization problems are greatly acknowledged in literatures [11,12,13]. Moreover, number of studies [5,6,7] has stated significance of minimum spanning tree for giving estimated solution to the hard problem such as TSP. In addition, the previous studies [9,10] have implemented minimum spanning tree via prim's algorithm. Based on this context, to solve the TSP through minimum spanning tree using prim's algorithm has addressed in this paper.

III. Objectives and contribution of proposed work

"Traveling salesman problem" applications include: how to plan the most reasonable and efficient road traffic, in order to reduce congestion, how to plan commodity circulation better, in order to reduce operating costs, how to set the node in the Internet environment, in order to let the information flow better.

1. To get an access to TSP problem and find out an algorithm for the best routes between China's cities.
2. Discuss the certain method (minimum spanning tree, prim's algorithm) we have used, combine the ideas and experiences during the program about the proposed methodologies.
3. Proposed approach finds shortest route from source to designation as well as to get positional co-ordinates.

IV. Prim's algorithm

In the area of computer and software engineering, Prim's algorithm, which is widely used in graph theory aims to find an MST for a connected weighted undirected graph. During this process, an assemble of edges is found and forms a tree that consist of each vertexes, the sum of cost of this tree should be minimum. The prim's algorithm is a greedy method

which finds a MST for a connected weighted undirected graph. It can finds the subset of edges and forms a tree that includes every vertex, where the total weights of all the edges in the tree is minimized.

In this algorithm, edges are added step by step, and continuously fills the assemble's size until all the vertices are in this assemble.

Input: A non-empty connected weighted graph with several vertexes V and edges E (the weights can be negative).
Initialize: V' = {x}, x, which is the start point of the newly initialized assemble V', can be any node in V, E' = {}
Repeat until V' = V: Choose an edge {u, v} with minimal weight so that u is in V' and v is not (pick any one among the edges with same weight). Thus, v is transferred to V', and {u, v} transferred to E'
Output: An MST combined by assembles V' and E'.
Psudocode:

MST-PRIM(G, w, r)
1 a point u∈ V[G]
2 do key[u] ← ∽
3 π[u] ← NIL
4 key[r] ← 0
5 Q ← V[G]
6 while Q≠Φ
7 do u ← EXTRACT-MIN(Q)
8 for each v ∈ Adj[u]
9 do if v ∈ Q and w(u, v)< key[v]
10 then π[v] ← u
11 key[v] ← w(u, v)

This paper is planning to solve TSP along with MST and Prim's algorithm is used to compute the shortest path between cities. The paper contributes the following steps to find an optimum solution.

Step1: Select the map and compute the distance (kilometers) among source and destination.
Step2: Choose to particular city to travel and find the shortest route.
Step3: In a graph choose vertices which represent the cities.
Step4: By using MST find the minimum cost to travel from source to destination and covers cities at most once.
Step5: The MST is implemented through prim's algorithm.
Step6: The shortest path among the cities will find the optimum result.

V. Implementation of TSP

The following data shows the indispensable figures that we are going to use along the overall documents, such as the cities and corresponding numbers, longitude and latitude of each cities as shown in Table 1 and Table 2.

City[1]=Beijing	City[2]=Shanghai
City[3]=Guangzhou	City[4]=Tianjin
City[5]=Wuhan	City[6]=Jinan
City[7]=Xian	City[8]=Chongqing
City[9]=Nanchang	City[10]=Chengdu

Table 1: China cities

Beijing(116.46, 39.92)	Shanghai(121.48, 31.22)
Guangzhou(113.14, 23.08)	Tianjin(117.20, 39.13)
Wuhan(114.17, 30.35)	Jinan(117.00, 36.40)
Xian(108.57, 34.17)	Chongqing(106.33, 29.35)
Nanchang(115.55, 28.40)	Chengdu(104.04, 30.40)

Table 2: longitude and latitude of each city

Set up vertexes with these 10 cities. Based on that, calculate the straight-line distances between any two cities (in degree), than change the unit to kilometers (Explain: 1 longitude=85.39km, 1 latitude=111km), distances is calculated by MST and prim's algorithm (difference of longitude) and d-Lat (difference of latitude), with the formula of the distance of two points, distances between cities are listed as follow (unit: kilometers):

The distances between Beijing and other cities.

cites	2	3	4	5	6	7	8	9	10
1	1056.56	1890.62	108.08	1080.12	393.43	928.05	1457.67	1281.08	1497.13

The distances between Shanghai and other cities.

cites	1	3	4	5	6	7	8	9	10
2	1056.56	1150.45	951.04	631.63	690.61	1149.99	1310.21	595.30	1491.98

The distances between Guangzhou and other cities.

cites	1	2	4	5	6	7	8	9	10
3	1890.62	1150.45	1814.97	811.75	1514.81	1291.36	906.93	625.35	1124.27

The distances between Tianjin and other cities.

cites	1	2	3	5	6	7	8	9	10
4	108.08	951.04	1814.97	1008.34	303.51	919.87	1428.29	1199.33	1483.84

The distances between Wuhan and other cities.

cites	1	2	3	4	6	7	8	9	10

5	1080.12	631.63	811.75	1008.34	713.71	639.10	678.60	246.45	865.02

The distances between Jinan and other cities.

cites	1	2	3	4	5	7	8	9	10
6	393.43	690.61	1514.81	303.51	713.71	761.21	1201.04	896.59	1291.60

The distances between Xian and other cities.

Cites	1	2	3	4	5	6	8	9	10
7	928.05	1149.99	1291.36	919.87	639.10	761.21	568.18	874.90	569.86

The distances between Chongqing and other cities.

Cites	1	2	3	4	5	6	7	9	10
8	1457.67	1310.21	906.93	1428.29	678.60	1201.04	568.18	794.33	227.64

The distances between Nanchang and other cities.

Cites	1	2	3	4	5	6	7	8	10
9	1281.08	595.30	625.35	1199.33	246.45	896.59	874.90	794.33	1007.60

The distances between Chengdu and other cities.

Cites	1	2	3	4	5	6	7	8	9
10	1497.13	1491.98	1124.27	1483.84	865.02	1291.60	569.86	227.64	1007.60

Example graph (in order to show these 10 cities in Chinese map): Some screenshot of our software is by taking the example of china map. Distance between the cities is shown in Fig.1.

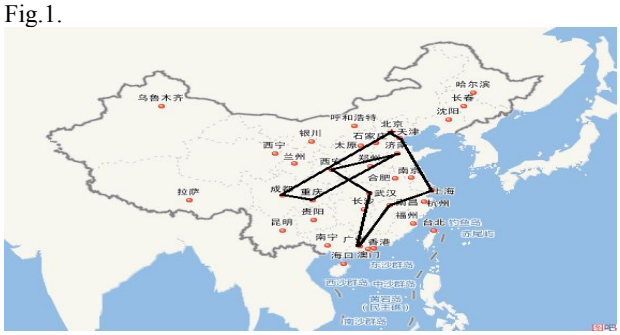

Figure 1. Distance between Chinese Cities

If you want to visit all the ten cities, and start with Beijing, the best route is:
Beijing -> Tianjin -> Shanghai -> Nanchang -> Guangzhou -> Wuhan -> Xian -> Jinan -> Chongqing -> Chengdu -> Beijing
Plus, the place with a star is the capital Beijing.

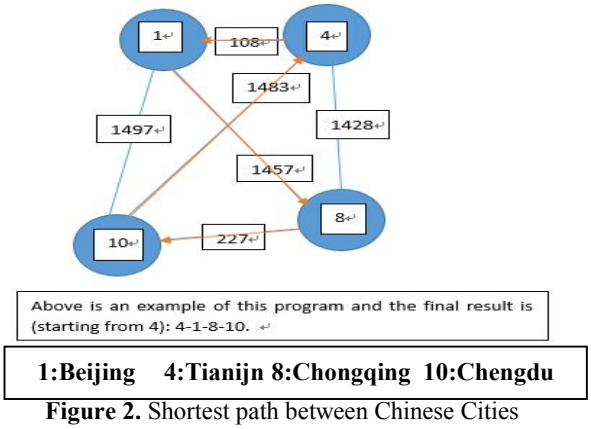

Figure 2. Shortest path between Chinese Cities

The distance, time between cities such as Beijing to Chengdu is measured by selecting the bubbles. And finally we have calculated the shortest path between Beijing and Chengdu as shown in Fig.2. The implementation source code of TSP on basis of MST and prim's approach as shown in Appendix.

Appendix : Program

Declare variables and the structure of Vertex at the beginning of the program.

```
# include<stdio.h>
# include<stdlib.h>
# include<time.h>
# include<math.h>
# define Max 11

int cn,tt,start;                    // cn-city Numbers
double arry1[Max][Max];             // adjacent matrix,used to store distances
double fn=0,gn=0,hn=0;              // heuristic function
double f1=0,g1=0,h1=0;
int arry3[Max];                     // mark the cities have been
int arry4[Max];

//  Define Vertex DataType
struct Vertex
{
  double x;   //longitude
  double y;   //latitude
}City[Max];

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////

// main function
int main()
```

```
{
 printf("City[1]=Beijing
City[2]=Shanghai\nCity[3]=Guangzhou
City[4]=Tianjin\nCity[5]=Wuhan
City[6]=Jinan\nCity[7]=Xian
City[8]=Chongqing\nCity[9]=Nanchang
City[10]=Chengdu\n");
 void CityCoordinate();
 double CityCost(int,int);
 void TSP();
 double MaxLengh();

 int i,j;

 CityCoordinate();

 printf("\n");
 printf("\n");

  for(i=1; i<Max; i++)
  {
   tt=0;
   for(j=i; j<Max; j++,tt++)
   {
    if(i==j)   arry1[i][j]=0;
    else   arry1[i][j]=CityCost(i,j);
   }
  }

 TSP();

 printf("\nBest route£º%d¡ú",start,start);
 for(i=2;i<=cn;i++) printf("%d¡ú",arry3[i]);
 printf("%d\n",arry3[cn+1]);

 printf("Overall Distances %.2f km\n",fn);
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////

void CityCoordinate()
{
 int i,j,hh=0;

  City[1].x=116.46,City[1].y=39.92;
  City[2].x=121.48,City[2].y=31.22;
  City[3].x=113.14,City[3].y=23.08;
  City[4].x=117.20,City[4].y=39.13;
  City[5].x=114.17,City[5].y=30.35;
  City[6].x=117.00,City[6].y=36.40;
  City[7].x=108.57,City[7].y=34.17;
  City[8].x=106.33,City[8].y=29.35;
  City[9].x=115.55,City[9].y=28.40;
  City[10].x=104.04,City[10].y=30.40;
 for(i=1;i<Max;i++)
 {

  for(j=1;j<i;j++)
    if(City[i].x== City[j].x&& City[i].y== City[j].y)
      i=i-1;

  hh++;
   if(i%2!=0)  hh=0;
   if(hh==0)   printf("\n");
   printf("Coordinate of City[%d]: (%.2f,%.2f);   ", i,
City[i].x, City[i].y);

 }
}

double CityCost(int i,int j)
{
 int hh=0;
 float x1,x2,y1,y2,Distance,t1,t2,t;

 x1= City[i].x; y1= City[i].y;
 x2= City[j].x; y2= City[j].y;


 t1=(x1-x2)*85.39; // 1 longitude=85.39km, 1
latitude=111km
 t2=(y1-y2)*111;
 t=t1*t1+t2*t2;
 Distance=sqrt(t);

 arry1[i][j]=Distance;

 hh++;
 if(0!=tt%2)  hh=0;          // beautify the output, make the
format specification
 if(0==hh)   printf("\n");

 printf("Distance between %d and %d(km)£º%3.2f      ", i, j,
Distance);
 return arry1[i][j];
}

// USING MST
void TSP()
{
 int Mnode;                          // starting point, searching
level's father node
 int h,i,k,l,m,n,nn;
 int x,y=0;
 int arry2[Max]={0,0, 0,0, 0, 0,0, 0,0, 0}; // marking array, 0
means already been, while 1 means not
 double temp1=100,temp2=100;
 double layer1[Max];                 // Initialize searching
level's node
 double layer2[Max];                 // Initialize searching
level's successor node

printf("\nInput how many cities you will be£º");
 scanf("%d",&cn);
 printf("\n");

 printf("Input the number of cities you will visit£º\n");
 for(h=1;h<=cn;h++)
 {
  scanf("%d",&x);
  if(0==arry2[x]) arry2[x]=1;   // Avoiding repeat
  else if(1==arry2[x]) h=h-1;
 }

 printf("\n");
```

```
 for(i=1;i<Max;i++)
   if(1==arry2[i])
     printf("%d ",i);
 printf("\n");

 printf("Input the number of beginning city£º");
 scanf("%d",&start);
 printf("\n");

 arry2[start]=0;                    // Initial
 arry3[1]=start;
 arry3[cn+1]=start;
 Mnode=arry3[1];

for(i=1;i<Max;i++) printf("%d ",arry2[i]);   // Output arry2[]
 printf("\n");
// Searching route
for(n=2;n<=cn;n++)                  // Find city2¡«cn
{
 for(nn=1;nn<Max;nn++)              // Initialize
layer1[]ºÍlayer2[]
 {
    layer1[nn]=0;
    layer2[nn]=0;
 }

 for(k=1;k<Max;k++)                 // Search all the successor
node of Mnode
    if(1==arry2[k])
    {
      gn=g1+arry1[Mnode][k];
      hn=arry1[k][start];
      fn=gn+hn;
      layer1[k]=fn;
    }

 for(l=1;l<Max;l++)                 // Search the first successor
node y and initialize it
   if(0!=layer1[l])
   {
    y=l;
    break;
   }

 for(i=1;i<Max;i++) printf("%d ",arry2[i]);   // Output arry2[]
 printf("\n");

 for(m=y+1;m<Max;m++)  //Compare and find the best
successor
 {
   if(layer1[y]==layer1[m]) //If the cost of two successor are
same£¬search  for their next layer nodes
   {
    Mnode=y;                        //Regard y as father node at first
    arry2[y]=0;
    for(k=1;k<Max;k++)              // Search all the successor
node of y
      if(1==arry2[k])
      {
        gn=g1+arry1[Mnode][k];
        hn=arry1[k][start];
        fn=gn+hn;
        layer2[k]=fn;
      }
    for(l=1;l<Max;l++)              // Find out subsequent nodes
with minimum cost of 'y'
      if(0!=layer2[l]&&temp1>layer2[l])    temp1=layer2[l];

    for(nn=1;nn<Max;nn++)    layer2[nn]=0;   //Initialize
layer2[]

    Mnode=m;                        //Regard m as father node
    arry2[y]=1;
    arry2[m]=0;
    for(k=1;k<Max;k++)              //Search all the successor
node  of y
      if(1==arry2[k])
      {
        gn=g1+arry1[Mnode][k];
        hn=arry1[k][start];
        fn=gn+hn;
        layer2[k]=fn;
      }
    for(l=1;l<Max;l++)              // Find out subsequent nodes
with minimum cost of 'm'
      if(0!=layer2[l]&&temp2>layer2[l])    temp2=layer2[l];

    arry2[y]=1;    arry2[m]=1;

    if (temp1>temp2)  y=m;
   }
   else if(0!=layer1[m]&&layer1[y]>layer1[m])  y=m;
 }                  //Compare and find the best successor

 for(i=1;i<Max;i++) printf("%d ",arry2[i]);
 printf("\n");

 g1=g1+arry1[Mnode][y];
 Mnode=y;
 arry2[y]=0;
 arry3[n]=y;
 }
 for(i=1;i<Max;i++) printf("%d ",arry2[i]);
 printf("\n");

fn=g1+arry1[y][start];
}
```

VI. Results and Discussion

For example, a salesman wants to visit Beijing, Tianjin, Chengdu and Chongqing, and start his trip in Tianjin. Result is shown as follow.

```
Input how many city you would visit: 4

Input the numebers of cities you are about to visit:
1 4 8 10

1 4 8 10
Input the number of starting city: 4

1 0 0 0 0 0 0 1 0 1
1 0 0 0 0 0 0 1 0 1
1 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0

The best route starting from 4 is: 4->1->8->10->4
The total distances: 1685.31 km
```

Figure 3: Find optimum route

Conclusion

This kind of problem is a typical NPC combinatorial optimization problems (NPC = Non - deterministic Polynomial complete, is the uncertain and complete problem of Polynomial complexity. The mathematical description of TSP is: In a graph with weights, find the minimum Hamilton loop. In cities of number N, every two cities have connected path, whose quantity shall be N * (N - 1) / 2. For undirected connected graph containing n vertices, the quantity of complete graph's edges is also n * (n - 1) / 2, therefore, we can use fully connected undirected graph which contains n vertexes to image the known conditions of TSP problem. A minimal spanning tree is a minimal connected subgraph of a connected graph, it contains all the n vertices of connected graph, a minimum spanning tree, is a spanning tree whose price is the least among all the spanning tree of this graph. Therefore, for solving TSP problem, we can use the method of calculating the minimum spanning tree using prim's algorithm.

References

[1] Gutin, G., Yeo, A., & Zverovich, A. (2002). Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics*, *117*(1), 81-86.

[2] Lenstra, J. K., Kan, A. R., & Shmoys, D. B. (1985). *The traveling salesman problem: a guided tour of combinatorial optimization* (Vol. 3). New York: Wiley.

[3] Takahashi, H., & Matsuyama, A. (1980). An approximate solution for the Steiner problem in graphs. *Math. Japonica*, *24*(6), 573-577.

[4] Huang, L., Wang, K. P., Zhou, C. G., PANG, W., DONG, L. J., & PENG, L. (2003). Particle Swarm Optimization for Traveling Salesman Problems [J]. *Acta Scientiarium Naturalium Universitatis Jilinensis*, *4*, 012.

[5] Pettie, S., & Ramachandran, V. (2002). An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, *49*(1), 16-34. (mst)

[6] J. B. Kruskal. On the shortest spanning subtree and the traveling salesman problem. Proc. Am. Math. Soc., 7:48.50, 1956.

[7] R.L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. Annals of the History of Computing, 7(1):43–57, January 1995.

[8] Arogundade, O. T., Sobowale, B., & Akinwale, A. T. (2011). Prim Algorithm Approach to Improving Local Access Network in Rural Areas. *International Journal of Computer Theory and Engineering*, *3*(3).

[9] Pop, P. C., & Zelina, I. (2004). Heuristic Algorithms for the Generalized Minimum Spanning Tree Problem. *Proceedings of ICTAMI*.

[10] Pop, P. C., Sitar, C. P., & Zelina, I. (2004). Efficient Algorithms for the Generalized Minimum Spanning Tree Problem. In *Proceedings of 4-th International Conference on Applied Mathematics, Baia Mare, Romania* (pp. 23-26).

[11] Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, *3*(4), 376-384.

[12] Held, M., & Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical programming*, *1*(1), 6-25.

[13] Mosheiov, G. (1994). The travelling salesman problem with pick-up and delivery. *European Journal of Operational Research*, *79*(2), 299-310.