FINDING OPTIMAL TRAVELLING SALESMAN TOURS THROUGH MYANMAR CAPITAL CITIES

Kyaw Moe Min*

Abstract

The travelling salesman problem (TSP) is a combinational optimization problem in which the goal is to find the shortest path between different cities that the salesman takes. The travelling salesman problem involves finding the trip of minimum cost and the processing time that a salesman can make to visit the cities in a sales territory once and only once (represented by a complete graph with weights on the edges), starting and ending the trip in the same city. This paper adopts the nearest neighbour and two sided nearest neighbour algorithm to solve the well-known travelling salesman problem. The algorithms were implemented using (C++) and MS VC++programming language. The approach can be tested on two graphs that making a TSP tour instance of 5-city, 14-city and more. The computation results validate the performance of the proposed algorithm.

Keywords- travelling salesman problem (TSP), nearest neighbour, two sided nearest neighbour

Introduction

The travelling salesman problem (TSP) consists of a salesman and a set of Myanmar capital cities. The salesman has to visit each one of the capital cities starting from a certain one and returning to the same city. The challenge of the problem is that the travelling salesman wants to minimize the total length of the trip. A salesman must make a tour of a number of Myanmar cities using the shortest path available and visit each city exactly once and only once and return to the original starting point.

The travelling salesman problem can be described as follows:

TSP = {(G, f, t): G = (V, E) a complete graph, f is a function V×V --> Z, t ∈ Z,

G is a graph that contains a travelling salesman tour with cost that does not exceed t}.

*. Dr, Associate Professor and Head, Department of Computer Studies, National Management Degree College

Example:

Consider the following set of cities:

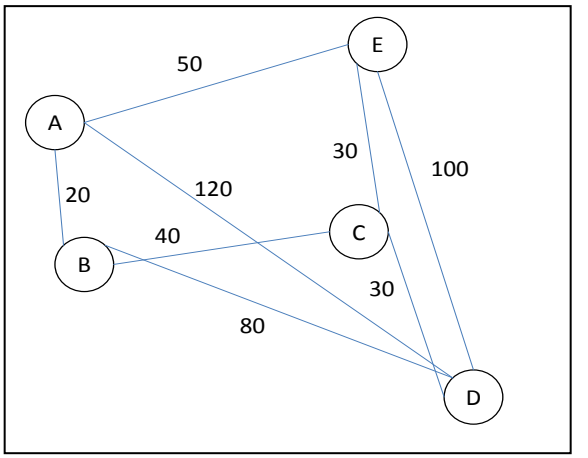

Figure 1: A graph with weights on its edges.

The problem lies in finding a minimal path passing from all vertices once. In the figure 1, the path ***Path1*** {A, B, C, D, E, A} and the path ***Path2*** {A, B, C, E, D, A} pass all the vertices but Path1 has a total length of 240 and Path2 has a total length of 310.

Statement of the Problem

The travelling salesman problem involves a salesman who must make a tour of a number of cities using the shortest path available and visit each city exactly once and only once and return to the original starting point. For each number of cities n, the number of paths which must be explored is n!, causing this problem to grow exponentially rather than as a polynomial. There are bunch of algorithms offering comparably fast running time and still yielding near optimal solutions.

Solution Methods

The following algorithms can be used to find the shortest path.

(i) Nearest Neighbour Algorithm

(ii) Two-sided Nearest Neighbour Algorithm

(iii)Nearest Insert Algorithm

(iv)Farthest Insert Algorithm

(v) Cheapest Insert Algorithm

(vi)Spanning Tree Algorithm

(vii)Chris to fides Algorithm

There will be coded Algorithms (i)Nearest Neighbour and (ii) Two-sided Nearest Neighbourin C^{++} language and apply them to find optimal or near optimal tours passing through sure specified cities of Myanmar.

Nearest Neighbour Algorithm

Input: V= {1, 2, 3,.... , n }, the set of labels of n cities

distancec_{ij}, between city i and city j.

Output: a TS tour T of optimal or near optimal length

Step 1. (Initialization)

Set visit [j]: = false for all $j \in V$.

Choose any vertex $i \in V$.

Set s: = i (s is the starting city)

T [1]: = s (T is the tour or sequence of cities)

visit [s] = true

tl:= 0 (tl is the length of the tour)

p: = s (p is the present city)

count: = 1.

Step 2. Choose a vertex k such that

$c_{pk} = \min \{c_{pj} \mid \text{visit}[j] = \text{false} \}$

Set count: = count + 1

T[count]: = k

visit [k]: = true

tl : = tl +c_{pk}

p: =k

Go to Step 2.

Step 3. Set T[n+1] = s.

Output T and tl. Stop.

The code will be described the above algorithm in C^{++} as follows. Firstly, the **distance matrix type** will be coded as follow.

```
int d[50][50]={ {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {0,0,153,510,706,966,581,455,70,160,372,157,567,342,398},
                {0,153,0,422,664,997,589,507,197,311,479,310,581,417,376},
                {0,510,422,0,289,744,390,437,491,645,880,618,374,455,211},
                {0,706,664,289,0,491,265,4101,664,804,1070,768,259,499,310},
                {0,966,997,744,491,0,400,514,902,990,1261,947,418,635,636},
                {0,581,598,390,265,400,0,168,522,634,910,593,18,284,239},
                {0,455,507,437,410,514,168,0,389,482,757,439,163,124,232},
                {0,70,197,491,664,902,522,389,0,154,406,133,508,273,355},
                {0,160,311,645,804,990,634,482,154,0,276,43,623,358,498},
                {0,372,479,880,1070,1261,910,757,406,276,0,318,898,633,761},
                {0,157,310,618,768,947,593,439,133,43,318,0,582,315,464},
                {0,567,581,374,259,418,18,163,508,623,898,582,0,275,221},
                {0,342,417,455,499,635,284,124,273,358,633,315,275,0,247},
                {0,398,376,211,310,636,239,232,355,498,761,464,221,247,0} };
```

Secondly, the part of nearest neighbour algorithm used will be coded as follow.

```
while(count !=n)
{min =999999;
        for(i=1;i<=n;i++)
        {if(visit[i]==0){   if(d[pc][i]<min)
                {       min=d[pc][i];
```

```
                        nc=i;
                    }}}
                count=count+1;
                T[count]=nc;
                visit[nc]=1;
                tl=tl+min;
                pc=nc;
                cout<<min<<" + ";     }
                T[n+1]=fc;
                tl=tl+d[pc][fc];
                cout<<d[pc][fc]<<" = " <<tl;
                cout<<"\n\n A near optimal tour is "<<" ";
                cout<<"{ ";
                for(i=1;i<=n;i++)
                cout<<(T[i])<<", ";
                cout<<fc<<" }";
                cout<<"\n\nThe total length of the tour is"<<"
"<<tl<<".";}
```

Finding a Near Optimal Tour through 14 Cities in Myanmar by Using the Nearest Neighbour Algorithm

By using the above nearest neighbour algorithm we find out a near optimal tour passing through 14 cities in Myanmar: Yangon, Pathein, Sittwe, Hakha, Myitkyina, Mandalay, Taunggyi, Bago, MawlaMyine, Dawei, Hpa-an, Sagaing, Loikaw, Magwe. The distances, in kilometer, between these cities can be given by the following distance matrix in figure 2.

	Yangon	*Pathein*	*Sittwe*	*Hakha*	*MyintKyina*	*Mandalay*	*Taunggyi*	*Bago*	*MawlaMyine*	*Dawei*	*Hpa-an*	*Sagaing*	*Loikaw*	*Magwe*
Yangon	0	153	510	706	966	581	455	70	160	372	157	567	342	398
Pathein	153	0	422	664	997	589	507	197	311	479	310	581	417	376
Sittwe	510	422	0	289	744	390	437	491	645	880	618	374	455	211
Hakha	706	664	289	0	491	265	4101	664	804	1070	768	259	499	310
MyintKyina	966	997	744	491	0	400	514	902	990	1261	947	418	635	636
Mandalay	581	598	390	265	400	0	168	522	634	910	593	18	284	239
Taunggyi	455	507	437	410	514	168	0	389	482	757	439	163	124	232
Bago	70	197	491	664	902	522	389	0	154	406	133	508	273	355
MawlaMyine	160	311	645	804	990	634	482	154	0	276	43	623	358	498
Dawei	372	479	880	1070	1261	910	757	406	276	0	318	898	633	761
Hpa-an	157	310	618	768	947	593	439	133	43	318	0	582	315	464
Sagaing	567	581	374	259	418	18	163	508	623	898	582	0	275	221
Loikaw	342	417	455	499	635	284	124	273	358	633	315	275	0	247
Magwe	398	376	211	310	636	239	232	355	498	761	464	221	247	0

Figure2: The distance relationship among the capital cities

If the city 1is chosen as the first city, then the algorithm produces the following output:

The first city ---> 1

A near optimal tour is { 1, 8, 11, 9, 10, 2, 14, 3, 4, 12, 6, 7, 13, 5, 1 }

The sum of distances = 70 + 133 + 43 + 276 + 479 + 376 + 211+ 289 +259 +18 + 168 + 124 + 635 + 966 = 4047

The total length of the tour is 4047.

If the city 2 is chosen as the first city, then the algorithm produces the following output:

The first city ---> 2

A near optimal tour is { 2, 1, 8, 11, 9, 10, 13, 7, 12, 6, 14, 3, 4, 5, 2 }

The sum of distances = 153 + 70 +133 +43 + 276+ 633 + 124 +163 +18+ 239 + 211 + 289 + 491 + 997 = 3840

The total length of the tour is 3840.

Table 1: Total distance and Time depending on the Starting City

City-Name	Starting City	Total Distance(km)	Time (ms)
Yangon	1	4047	3
Pathein	2	**3840**	**2**
Sittwe	3	4172	2
Hakha	4	3876	4
MyintKyina	5	4052	2
Mandalay	6	4207	1
Taunggyi	7	4059	2
Bago	8	3922	2
MawlaMyine	9	4728	2
Dawei	10	4016	2
Hpa-an	11	4706	3
Sagaing	12	4200	2
Loikaw	13	3922	5
Magwe	14	4092	2

By choosing the best of the above 14 tours in table 1, we obtain the following near optimal tour.

The tour = { 2, 1, 8, 11, 9, 10, 13, 7, 12, 6, 14, 3, 4, 5, 2 }

The total length of the tour = 3840 km

The total lengths on the starting city can be represented as a graph in the figure 3.

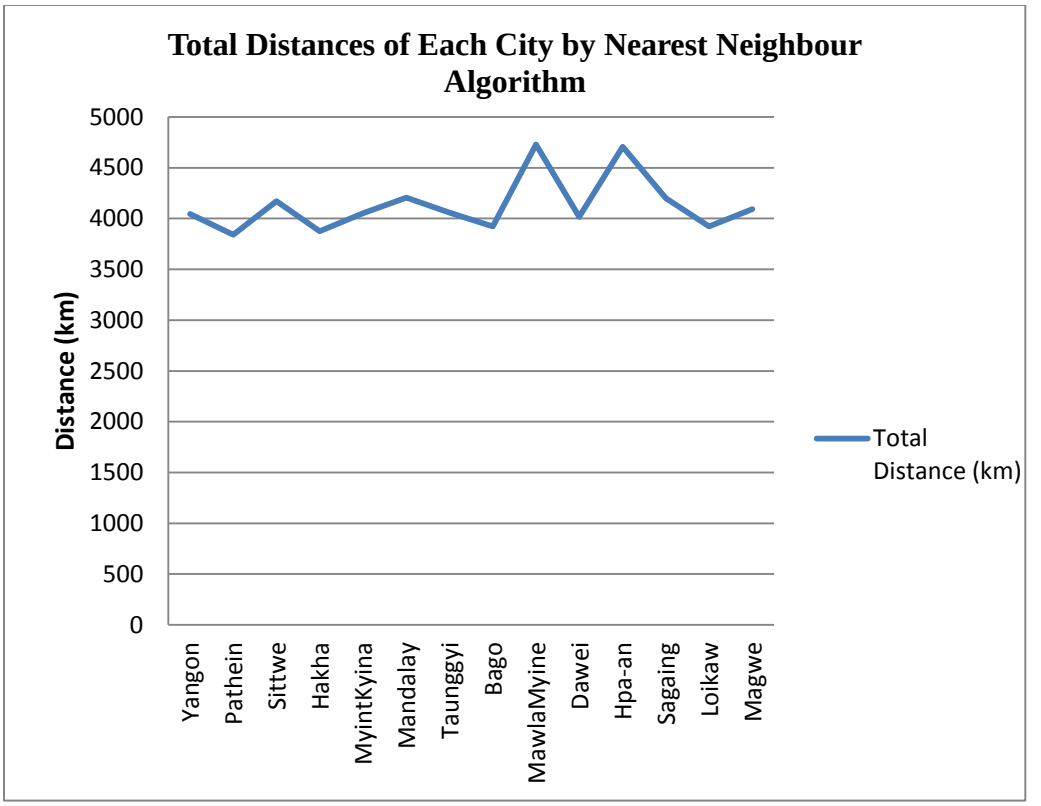

Figure 3: Total lengths of the tour on the starting cityby nearest neighbor algorithm

Two Sided Nearest Neighbour Algorithm

Input: V= {1, 2, ... , n }, the set of lables of n cities
distance sc_{ij}, between city i and city j

Output: a TS tour T

Step 1: (Initialization)
Set visit[j]:= false for all j ∈ V.
Choose any vertex i∈ V.
Set s: = i (s is the starting city)
T[1] = s (T is the tour or the sequence of cities)
visit[s]: = true

tl: = 0 (tl is the length of the tour)

p_l: = s (p_l is the present left city)

p_r: = s (p_r is the present right city)

count:= 1

Step 2. If count = n, set T[n+1]: = T[1] (count is the number of already visited cities)

Output T and tl.

Step 3. Choose a vertex k such that

$c_{p_r k} = \min\{ c_{p_r j} \mid \text{visit}[j] = \text{false} \}$

Choose a vertex m such that

$c_{p_l m} = \min\{ c_{p_l j} \mid \text{visit}[j] = \text{false} \}$

If $c_{p_r k} \leq c_{p_l m}$, then set

count: = count + 1

T[count] = k

visit[k] = true

tl = tl + $c_{p_r k}$

p_r : = k

Otherwise

for j = count to 1

Set T[j+1]: = T[j]

T[1]: = m

count: = count+1

visit[m]: = true

tl:= tl + $c_{p_l m}$

p_l: = m

Go to Step 2.

We can code the above algorithm in C^{++} as follows.

```
visit[sc]=1;
```

```
T[1]=sc;
inttl=0;
intplc = sc;
intprc = sc;
int count=1;
int min1,min2,nrc,nlc;
while(count !=n)
 {min1 =999999;
        for(i=1;i<=n;i++)       {
        if(visit[i]==0)  {      if(d[prc][i]<min1)
                {       min1=d[prc][i];
                        nrc=i;}}}
        min2=999999;
        for(i=1;i<=n;i++)       {
                if(visit[i]==0)  {      if(d[plc][i]<min2)
                        {min2=d[plc][i];
                        nlc=i;  }}}
        if(min1<=min2)          {       count=count+1;
                T[count]=nrc;
                visit[nrc]=1;
                tl=tl+min1;
                prc=nrc;}else{ for(i=count;i>=1;i--)
                T[i+1]=T[i];
                T[1]=nlc;
                count=count+1;
                visit [nlc]=1;
                tl=tl+min2;
                plc=nlc;}
          }
```

```
                    T[n+1]=plc;
                    tl=tl+d[prc][plc];
                    cout<<"\n\n A near optimal tour is "<<" ";
                    cout<<"{ ";
                    for(i=1;i<=n;i++)
                    cout<<(T[i])<<", ";
                    cout<<plc<<" }";
                    cout<<"\n\nThe total length of the tour is"<<"
"<<tl<<".";}
```

Finding a Near Optimal Tour through 14 Cities in Myanmar by Using the Two-sided Nearest Neighbour Algorithm

By using the above nearest neighbour algorithm we find out a near optimal tour passing through 14 cities in Myanmar: Yangon, Pathein, Sittwe, Hakha, Myitkyina, Mandalay, Taunggyi, Bago, MawlaMyine, Dawei, Hpa-an, Sagaing, Loikaw, Magwe. The distances, in kilometer, between these cities can be given by the **Figure (2)** distance matrix.

If we choose the city 5 as the first city, the city 10 as the second city, and so on, then the algorithm produces the following outputs:

The starting city ---> 5

A near optimal tour is { 5, 6, 12, 7, 13, 14, 3, 4, 2, 1, 8, 11, 9, 10, 5 }

The total length of the tour is 4052.

The starting city ---> 10

A near optimal tour is { 10, 9, 11, 8, 1, 2, 14, 3, 4, 12, 6, 7, 13, 5, 10 }

The total length of the tour is 4016.

Table 2: Total distance and Time depending on the Starting City

City-Name	Starting City	Total Distance (km)	Time (ms)
Yangon	1	4016	2
Pathein	2	4016	3
Sittwe	3	3906	3
Hakha	4	**3876**	**3**
MyintKyina	5	4052	2
Mandalay	6	3921	2
Taunggyi	7	3921	2
Bago	8	4016	1
MawlaMyine	9	4016	2
Dawei	10	4016	4
Hpa-an	11	4016	2
Sagaing	12	3921	2
Loikaw	13	3921	3
Magwe	14	3908	2

By choosing the best of the above 14 tours in table 2, we obtain the following **near optimal tour.** The tour = { 4, 5, 12, 6, 7, 13, 14, 3, 2, 1, 8, 11, 9, 10, 4 }

The total length of the tour = 3876 km

The total lengths on the starting city can be represented as a graph in the figure 4.

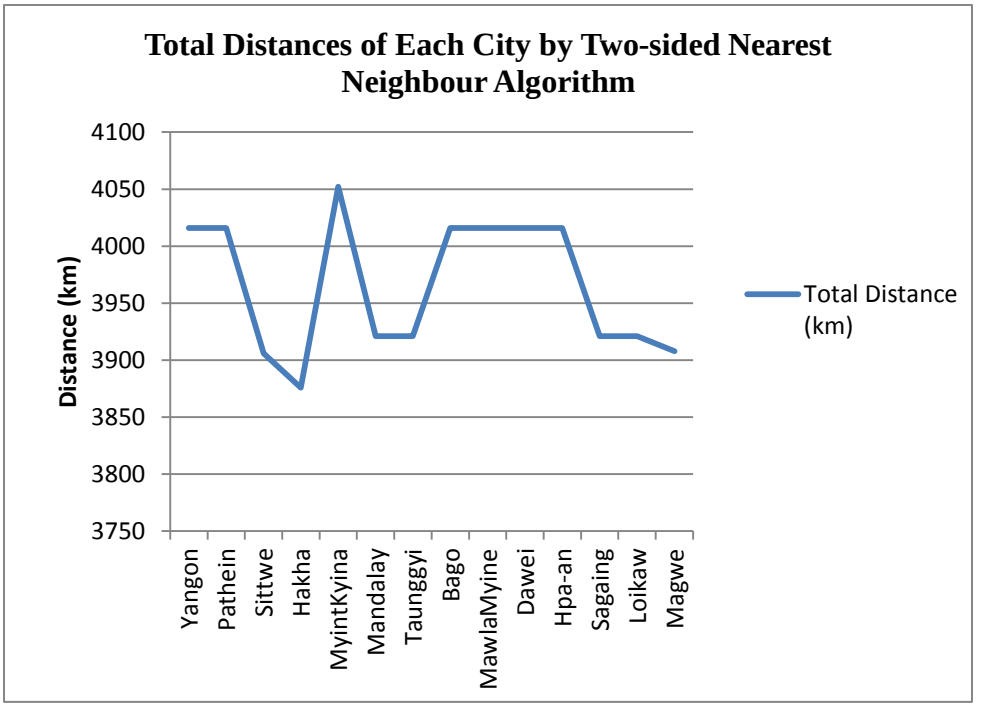

Figure 4: Total length of the tour on the starting city by two-sided nearest neighbor algorithm

Conclusion

We have coded the nearest neighbour algorithm and the two-sided nearest neighbour algorithm in C++ language used them to find near optimal tours through the 14 cities in Myanmar.

Table 3: Total distance and Processing Time depending on Starting City by the two Algorithms

City-Name	Starting City	nearest neighbour algorithm		two-sided nearest neighbour	
		Total Distance(km)	Time (ms)	Total Distance(km)	Time (ms)
Yangon	1	4047	3	4016	2
Pathein	2	**3840**	**2**	4016	3
Sittwe	3	4172	2	3906	3
Hakha	4	3876	4	3876	3
MyintKyina	5	4052	2	4052	2
Mandalay	6	4207	1	3921	2
Taunggyi	7	4059	2	3921	2
Bago	8	3922	2	4016	1
MawlaMyine	9	4728	2	4016	2
Dawei	10	4016	2	4016	4
Hpa-an	11	4706	3	4016	2
Sagaing	12	4200	2	3921	2
Loikaw	13	3922	5	3921	3
Magwe	14	4092	2	3908	2

We can choose the optimal tour path in order to table 3. By using the nearest neighbour algorithm, the total tour distance is less than that distance of the two-sided nearest neighbor used. The total lengths of the tour on the starting city by two algorithms can be represented as a graph in the figure 5.

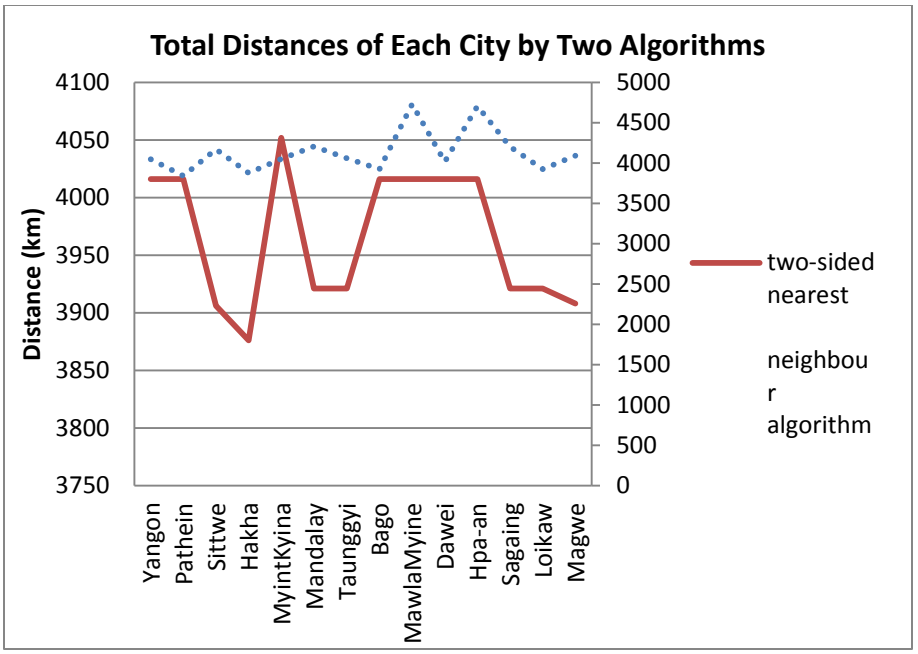

Figure 5: Total lengths of the tour on the starting city by two algorithms

We observe the processing time in both the nearest neighbour and the two sided nearest neighbour programs. The processing time is less. Moreover, these two algorithms can be applied to solve the travelling salesman problems containing more and more cities in Myanmar.

References

Amanur (2012):"The Travelling Salesman Problem, Amanur Rahman Saiyed, Indiana State University, Terre Haute, IN 47809 , USA, asaiyed@sycamores.indstate.edu, April 11, 2012

Lawler, L.(1976): "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart& Winston, New York,1976

Lawler, J.K.(1985): "The Travelling Salesman Problem. A Guided Tour of Combinatorial Optimization", A.H.G Rinnooy Kan & D.B. Shmoys and Lawler, J.K. Lenstra, Wiley, New York, 1985.

Richard L.(2018):"Fundamentals of Programming C++", Richard L. Halterman, School of Computing, Southern Adventist University, Pages-746, February 28, 2018.

Tim Bailey(2005):"An Introduction to the C Programming Language and Software Design", Tim Bailey, Pages-153, July 12, 2005.