Travelling Salesman Problem Report

Abhinav Dixit (B16003) Mohd. Nadeem (B1064)

TSP is NP-Hard problem and exact alogorithm have solution of order O($(n-1)!$). Using Dynamic Programming, it can be solved in order O($n^2 2^n$) by Held-Carp algorithm. There are some heuristic and approximation algorithm which can give good solution but not the best. We have used Greedy approach(constructive algorithm) to construct initial path and then applied 3-OPT problem to optimize the path.

3-OPT Algoritthm

3-opt analysis involves deleting 3 connections (or edges) in a network (or tour), to create 3 sub-tours. Then the 7 different ways of reconnecting the network are analysed to find the optimum one. This process is then repeated for a different set of 3 connections, until all possible combinations have been tried in a network. A single execution of 3-opt has a time complexity of O(n^3). Algorithm is given below, Source(https://en.wikipedia.org/wiki/3-opt).

```
      def all_segment(self):
        return [(i,j,k) for i in range(len(self.tour)) for j in
range(i+2,len(self.tour)) for k in range(j+2, len(self.tour) + (i>0) )]

    def reverse_if_better(self,i,j,k):

        A = self.tour[i-1]
        B = self.tour[i]
        C = self.tour[j-1]
        D = self.tour[j]

        E = self.tour[k-1]

        F = self.tour[k%self.n]

        d0 = self.distance[A][B] + self.distance[C][D] + self.distance[E][F]
        d1 = self.distance[A][C] + self.distance[B][D] + self.distance[E][F]
        d2 = self.distance[A][B] + self.distance[C][E] + self.distance[D][F]
        d3 = self.distance[A][D] + self.distance[E][B] + self.distance[C][F]
        d4 = self.distance[F][B] + self.distance[C][D] + self.distance[E][A]

        if d0>d1:
            self.tour[i:j] = reversed(self.tour[i:j])

            return -d0+d1
        elif d0 > d2:
            self.tour[j:k] = reversed(self.tour[j:k])
            return d2-d0
        elif d0>d4:
            self.tour[i:k] = reversed(self.tour[i:k])
            return d4-d0
        elif d0>d3:
            tmp = self.tour[j:k]
            tmp=tmp+self.tour[i:j]
```

```
            self.tour[i:k] = tmp

            return d3-d0
        return 0

    def three_opt(self):

        delta = 0

        for(a,b,c) in self.all_segment():

            self.t1=time.time()
            if (self.t1-self.t0)>=float(280):

                print("\n")
                for i in range(len(self.tour)):
                 print((self.tour)[i], end=" ")
                sys.exit(0)

            delta += self.reverse_if_better(a,b,c)

        self.cost += delta
        self.t1=time.time()
        if (self.t1-self.t0)>=float(280):
           print("\n")
           for i in range(len(self.tour)):
                print ((self.tour)[i], end =" ")
           sys.exit(0)


        if delta<0:

            self.three_opt()
```

Constructive Algorithm

In this algorithm, shortest edge is added to the tour connected to last visited vertex and the edge should not end the vertex which is visited. Order of this algorithm is O(n^2) Algorithm is as follows.

```
      def greedy_util(self, visited, i):
        mn = 1000000
        pos = 0
        for x in range(self.n):
            if(visited[x] == 0 and self.distance[i][x] < mn and x != i):
                mn = self.distance[i][x]
                pos = x
        return pos

    def generate_greedy(self):
        visited = np.zeros(self.n)
        visited[0] =1
        pos = self.greedy_util(visited,0)
        self.cost += self.distance[0][pos]
        self.tour.append(0)
```

```
self.tour.append(pos)
visited[pos] = 1

for x in range(self.n-2):
    tem = self.greedy_util(visited,pos)
    visited[tem] = 1
    self.cost += self.distance[pos][tem]
    self.tour.append(tem)
    pos = tem

self.cost += self.distance[self.tour[-1]][0]
```

The initial Tour is as follows :

Tour [1,2,3,.....,n] , where n is the number of cities in TSP.

Using 3-OPT algorithm without constructive algorithm, we get the following cost of tours as results in 5 minutes :

No. of Cities	100	250	500
Euclidian distances	1640.0056	2655.9648	3644.1378
Non Euclidian distances	5286.4762	12812.749	25334.6048

Using 3-OPT algorithm with constructive algorithm, we get the following cost of toursas results in 5 minutes :

No. of Cities	100	250	500
Euclidian distances	1655.1869	2618.3620	3525.0922
Non Euclidian distances	5274.1999	12797.9136	25327.11978

Conclusion

3-OPT gives good result without using greedy approach but in general result was better when used with greedy algorithm.