Report

Part C

TSP problem is a NP-complete problem. So, I search a lot of algorithms at first.
At first, I learned the genetic algorithm which is a great algorithm. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; his student David E. Goldberg further extended GA in 1989.And I discovered some disadvantages of the GA (GA is computationally expensive i.e. time-consuming.
Then I learned the ACO algorithms (Ant colony optimization algorithms). The inventors are Frans Moyson and Bernard Manderick. Pioneers of the field include Marco Dorigo, Luca Maria Gambardella. ACO algorithm can get rapid discovery of good solutions. But it also has some disadvantages like having a difficult theoretical analysis and more experimental than theoretical research and uncertain time to convergence.
Finally, I checked the piazza posted questions and I decide to make my own algorithms base on the idea from GA and ACO algorithm.

My own Algorithm:
This idea is based on GA, because there are lots of random variable at first in GA, it needs many evolutions to get the best offspring which is the best combination of the perm. In our Greedy Algorithm, the starting point is 0 which is fixed. If the starting point is fixed, this staring point might not be the best point to start. We would choose the cost of tourValue as the standard to compare whether it is a good starting point or not. Because of this, I want to use two for-loop to find out the shortest distance between two point and put one of it as the starting point. In this situation, the cost between first two point would be the smallest and it would largely increase the probability that those points would be local minimum solution. And then the greedy algorithm would keep finding the lowest cost node to go. Finally, I want to improve my algorithm. Because after I using the cities25, cities50 and cities75 as the tests. I found that as the number of cities increases, the performance of my own algorithm became worst. I found that the disadvantage of my own algorithm would be although the cost between first two nodes are being controlled in a good condition, the cost between the final nodes would be hard to control and difficult to improve when you using the greedy algorithm. So, I think the reverse function would be a wonderful way to improve this situation, because if after reversing the perm would improves the cost of tourValue, then reverse it. Indeed, after adding this function the performance of my algorithm largely improved. Hence, I put the reverse function as the final part of my algorithm. This is also to avoid the solution is the local minimum.
The polynomial time would be (n^3).

```
def UniversalGreedy(self):

        # Overall Greedy
        OriginalPerm = [int(i) for i in self.perm]
        lenPerm = len(OriginalPerm)
```

```
        miniValue = math.inf

        # find the starting node which is the lowest cost of tourValue
        for i in range(lenPerm):
            for j in range(lenPerm):
                if (i != j):
                    if (self.dists[i][j] <  miniValue):
                        miniValue = self.dists[i][j]
                        startIndex = i
                        endIndex = j

        NewPerm = [startIndex]
        NewPerm.append(endIndex)

        OriginalPerm.remove(startIndex)
        OriginalPerm.remove(endIndex)
        nextNode = endIndex

        # Greedy Algorithm
        while (len(OriginalPerm) > 0):
            miniValue = math.inf

            for j in range(lenPerm):

                 if (j not in NewPerm):
                    if (self.dists[nextNode][j] <  miniValue):
                        miniValue = self.dists[nextNode][j]
                        index = j
            if index not in NewPerm:
                nextNode = index
                NewPerm.append(nextNode)
                OriginalPerm.remove(nextNode)

        self.perm = NewPerm


        # the reverse function to avoid the local minimum
        better = True
        while better:
            better = False
            for j in range(self.n-1):
                for i in range(j):
                    if self.tryReverse(i,j):
                        better = True
```

Part D

Euclidean

	Identity	Swap	Swap&2-opt	Greedy	myOwnAlgorithms
Cities50 Weight:100 Number:50	2361.356	1977.157	620.748	672.148	580.121
Cities100 Weight:100 Number:100	5375.158	4029.245	818.154	884.968	811.134
Cities50-plus Weight:200 Number:50	5928.347	4686.363	1328.323	1598.757	1271.218
Cities100-plus Weight:200 Number:100	11318.227	8473.558	1654.252	1882.528	1640.119

The performance of my-algorithm is the best.

In Euclidean distance: There are two factors to influence the tourValue, number of cities and size of weight. As the cities number is increasing and the weight is constant, the performances of 2-opt algorithm is better than Greedy algorithm. The performance of greedy algorithm is not stable and as the number of cities is increasing the performance would be worst because of it has large probabilities to get a local minimum value. And the performance of 2-opt algorithm is always good and stable, which will not be influenced by the increasing number of cities. My own algorithm is slightly better than 2-opt because of the optimization. I use the optimization in the beginning of algorithm and the final part of algorithm, and this would largely decrease the probability of struggling in local minimum.

As the cities number is constant and the weight is increasing, the performance in 2-opt-algorithm is better than the Greedy algorithm. And the performance of my algorithm is similar with the 2-opt algorithm but still slightly better than it. In other hands, the improvement in my algorithm is workable and predictable.

In summary, my own algorithm would be great if the cities number is not very large and weight is not very large. (row 1 in graph). And the performance would be the best all the time even if the number of cities is increasing and the weight is increasing. Moreover, the performance is stable and will not easily influenced by the increasing number of weight and nodes.

Metric

	Identity	Swap	Swap&2-opt	Greedy	myOwnAlgorithms
Cities10 Weight:10 Number:10	46	40	22	20	19
Cities20 Weight:10 Number:20	103	69	20	30	19
Cities10-plus Weight:20 Number:10	139	105	70	72	70
Cities20-plus Weight:20 Number:20	205	131	38	62	30

The performance of my-algorithm is still the best because of my improvement in part c

In Metric distance: There are two factors to influence the tourValue, number of cities and size of weight. As the cities number is increasing and the weight is constant, the performances of 2-opt algorithm is better than Greedy algorithm. The performance of greedy algorithm is not stable and as the number of cities is increasing the performance would be worst because of it has large probabilities to get a local minimum value. And the performance of 2-opt algorithm is always good and stable, which will not be influenced by the increasing number of cities. My own algorithm is slightly better than 2-opt because of the optimization. I use the optimization in the beginning of algorithm and the final part of algorithm, and this would largely decrease the probability of struggling in local minimum.

As the cities number and the weight are both increase, the performance in 2-opt-algorithm is better than the Greedy algorithm. And the performance of my algorithm is still better than 2-opt algorithm.

In summary, my own algorithm would be great if the cities number is very large and weight is very large. And the performance would be the best all the time even if the number of cities is increasing and the weight is increasing. Moreover, the performance is stable and will not easily influenced by the increasing number of weight and nodes. More important, it would not be a local minimum value.