CSE6140 Project Metric Traveling Salesperson Problem

Chuchu Bai
cbai30@gatech.edu

Abhinaya Shetty
ashetty39@gatech.edu

Dawei He
davidecnu@gmail.com

1. INTRODUCTION

Metric Traveling Salesperson Problem is a NP-complete problem whose solution could be found through several different algorithms. In this project, we use 5 algorithms to find solutions for the given dataset.

2. PROBLEM DEFINITION

Traveling Salesman Problem is that given a complete graph G with node V and edge cost $c(u, v)$ defined for every pair of nodes, find the shortest simple cycle that visits all nodes in V. Specifically, this project focuses on metric TSP, which means all edge costs are symmetric and fulfill the triangle inequality:

$$c(u, v) \leq c(u, w) + c(w, v),\ \forall u, v, w \in V$$

Also, in this project, according to the given data type, there are two kinds of TSP. One is 2-dimensional Euclidean TSP. The nodes correspond to points in a 2-dimensional space, and the cost function is the Euclidean distance. The Euclidean distance between two points $p = (x_i, y_i)$ and $q = (x_j, y_j)$ is

$$d(p, q) = \sqrt{((x_i - x_j)^2 + (y_i - y_j)^2)}$$

The other one is GEO TSP. The data provided are the longitude and latitude. After converting them in radian, the distance are calculated in the following way:

$$q_1 = \cos\left(longitude_i - longitude_j\right)$$

$$q_2 = \cos\left(latitude_i - latitude_j\right)$$

$$q_3 = \cos\left(latitude_i + latitude_j\right)$$

$$d_{ij} = R * \arccos(0.5 * (1 + q_1) * q_2 - (1 - q_1) * q_3) + 1$$

where $R = 6378.388$.

3. RELATED WORK

For the greedy construction heuristic algorithm, Rosenkrantz and others implemented the nearest neighbor method in their paper[1]. This algorithm chooses a node not in the subtour nearest to any node in the sub tour. The worst case ratio of this method is shown to approach 2 as the number of cities increases. And according to this paper, there is also another strategy called farthest insertion[2], which has a different strategy in selection step. This farthest insertion algorithm could get a better guarantee that the worst case ratio i.e. $2\log(n) + 0.16$.
For the 2-approximation MST algorithm, there is a data structure which has good time complexity[3]. This data structure is an indexed priority queue of generic keys.
For the simulated annealing algorithm, there are lots of papers which discuss about the parameters selection. Moonwon Park and Yeong-Dae Kim find a procedure which could give good parameters without much complexity nonlinear programming[4].

4. ALGORITHMS

We implement 5 algorithms to find solution of TSP: greedy construction heuristic, 2-approximation using MST, local search1(hill climbing), local search2(simulated annealing) and branch-and bound.

4.1 Greedy Construction Heuristic

For this algorithm, we use the farthest-insert strategy. Then we read papers to find an improvement and we noticed that in 1974, *Rosenkrantz*[1] found a Nearest Insertion Algorithm. Based on this algorithm, we improved its performance by using an innovative strategy in the selection step. The algorithm we finally implemented is as follows.
The time complexity of this algorithm is $\mathbf{O(n^2)}$.
Let $NEARINSERT$ be the tour cost got by the nearest Insertion Algorithm in [1]. It is theoretically proven that

$$\frac{COST(NEARINSERT)}{OPTIMAL} < 2$$

. We change the strategy of selecting node in Step 2 and Step 3. Concretely speaking, in Step 2, in the original paper, they use $d(i_0, i_1) = \min_{k \in U_{out}} d(i_0, k)$, $\min_{i \in U_{in}} d(i, r) = \min_{j \in U_{out}} \min_{i \in U_{in}} d(i, j)$ to choose i_1, r in Step 2, Step 3, respectively. From the results of experiments for their algorithm and ours it shows that ours is even better.

4.2 2-Approximation MST

The 2-approximation using MST we implemented is sim-

Algorithm 1 Greedy Construction Heuristic

$U_{out} \leftarrow V, U_{in} \leftarrow \emptyset, Cost \leftarrow 0$
Randomly start with a sub-graph consisting of node i_0
$U_{out} \leftarrow U_{out} \setminus \{i_0\}, U_{in} = U_{in} \cup \{i_0\}$. Find $i_1 \in U_{out}$ such that

$$d(i_0, i_1) = \max_{k \in U_{out}} d(i_0, k)$$

$U_{out} \leftarrow U_{out} \setminus \{i_1\}, U_{in} = U_{in} \cup \{i_1\}, Cost \leftarrow d(i_0, i, i_1)$.
While $U_{out} \neq \emptyset$ **do**
Select $r \in U_{out}$ such that

$$\min_{i \in U_{in}} d(i, r) = \max_{j_i \in U_{out}} \min_{i \in U_{in}} d(i, j_i).$$

$U_{out} \leftarrow U_{out} \setminus \{r\}, U_{in} = U_{in} \cup \{r\}$.
Find the $edge(i, j)$ in the sub-tour that minimizes $d(i, r) + d(r, j) - d(i, j)$.
$Cost \leftarrow Cost + d(i, r) + d(r, j) - d(i, j)$
End While Return $Cost$, U_{in}

ply listed as follows.

- **Step 1** Find a MST T using Prim's algorithm
- **Step 2** Start at a root node
- **Step 3** Visit its children in a depth first manner

And in our algorithm, we use a data structure - an indexed priority queue that stores the vertices and their minimum distances from adjacent nodes.[2]. This was selected because of its good time complexity which is better than PriorityQueue for the following operations:

- **Insert**: add a new vertex to the queue. Time complexity: $\mathbf{O}(\log \mathbf{n})$.
- **isEmpty**: check if the queue is empty which means the MST has been created. Time complexity: $\mathbf{O}(\mathbf{c})$.
- **delMin**: pick up the vertex at least distance from the current node and delete it. Time complexity: $\mathbf{O}(\log \mathbf{n})$.
- **contains**: check if one vertex has been visited already. Time complexity: $\mathbf{O}(\mathbf{1})$.
- **decreaseKey**: update the vertex distance with the new minimum value. Time complexity: $\mathbf{O}(\log \mathbf{n})$.

The time complexity of the pre-order traversal is $\mathbf{O}(\mathbf{n})$ as each node is visited exactly once. And also, there is an approximation guarantee that

$$\frac{COST(2 - Approx)}{OptimalCost} \leq 2$$

4.3 Local Search - Hill Climbing

The first local search algorithm we implement is hill climbing. First, we choose a random tour as the initial solution. Then, rather than applying 2-OPT exchange or 3-OPT exchange, we combine these 2 strategies together. This combination could avoid sinking into a local optimal solution. The following are the Pseudo code:

4.4 Local Search - Simulated Annealing

Since we have implemented a good greedy construction heuristic algorithm, which leads to results close to the optimal solution, we decided to use this result as the initial solution of simulated annealing. Also,there are other details to implement this algorithm.

Algorithm 2 Hill Climbing

Initialize: $currTour \leftarrow RandomTour, currCost \leftarrow Cost(RandomTour)$
While not get opt value and not reach cut-off time **do**
$LM1 \leftarrow$ local minimum got by 3-Opt from the random solution
$LM2 \leftarrow$ local minimum got by 2-Opt from $LM1$
While$(LM1 \neq LM2)$ **do**
$LM1 \leftarrow$ local minimum got by 3-Opt from $LM2$
if$(LM1 = LM2)$: **break**
$LM2 \leftarrow$ local minimum got by 2-Opt from $LM1$
End While
$currTour \leftarrow LM1$
$currCost \leftarrow Cost(LM1)$
Choose another random solution
End While
Return $currTour$, $currCost$

- Initial temperature: 15000
- Cooling Rate: 0.0001
- Way to get new solution: 2-exchange
- Stop criteria: reach Optimal solution, reach cut-off time, or temperature < 0.001

The Pseudo code is as follows:

Algorithm 3 Simulated-Annealing

$optSol$, $currSol \leftarrow greedySol, optCost$, $currCost \leftarrow greedyCost, T \leftarrow 15000, CoolingRate \leftarrow 0.0001$
While not get opt value, not reach cut-off time and $T > 0.001$ **do**
$newSol \leftarrow$ Exchange 2 edges (i, j) and (k, l) in $currSol$
$newCost \leftarrow currCost - d(i, j) - d(k, l) + d(i, k) + d(j, l)$
if $newCost < currCost$:
$optSol$, $currSol \leftarrow newSol$
$optCost$, $currCost \leftarrow newCost$
else if $\exp \frac{newCost - currCost}{T} <$ a random number in $[0, 1]$:
$currSol \leftarrow newSol$, $currCost \leftarrow newCost$
$T \leftarrow T \times (1 - CoolingRate)$
End While
Return optCost, optSol

The result from simulated annealing algorithm could have the guarantee that

$$\frac{Cost(SA)}{Optimal} \leq 2$$

because the initial solution is given by the greedy algorithm. The time complexity for each new solution is $O(n)$ because we only need randomly choose 2 edges which costs $O(1)$ and then make a new tour which costs $O(n)$.
The space complexity for this algorithm is also not to large. We only need 3 arrays for current tour, new tour and current optimal tour and 3 integers for current cost, new cost and current optimal cost.

4.5 Branch and Bound

The branch and bound algorithm starts at $City_1$ by default. And its lower bound decided by

$lb = MST + Cost(PartialSolution)$+Lower bound on exiting a & b

The MST is found by Prim's algorithm as the 2-approx algorithm does. And the Pseudo code is as follows: The strength

Algorithm 4 Branch-and-Bound(P)

$Route \leftarrow \{sourcenode\}$
Search:
  If size(route) < number of nodes:
    Set remNodes to the set of unvisited nodes
    Identify the subproblem to be soleved
    If Cost after adding the node to rout < currOpt:
      Add a $problem[node, lowerbound]$ to the $problemset$
    If $problem - set$ is not empty:
      Solve each problem in the set in a recursive depth first manner by picking the most promising problem from the set first
    Else: **Return** current best solution as optimal solution
  Else: Check if new best solution is found:
    If Cost < best solution:
      Update the best solution to the new solution

of branch-and-bound is that it could find the optimal solution. And for our algorithm, it could return optimal solution for 2 TSPs in under 5 minutes. However, for those big graph with a large number of nodes, this algorithm will cost a lot of time to get the result, which might be a weakness.

5. EMPIRICAL EVALUATION

5.1 Experiment Environment

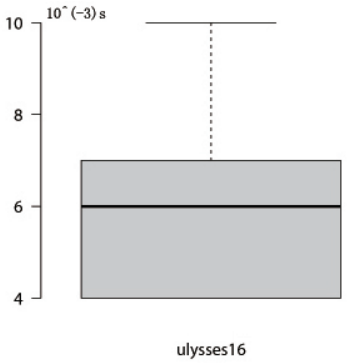
Laptop: Linux ThinkPad-X220 3.13.0-40-generic
System: #69-Ubuntu SMP x86-64 GNU/linux
Language: JAVA

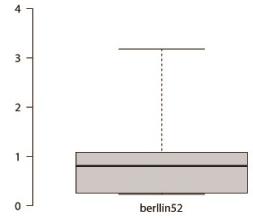
5.2 Result Table

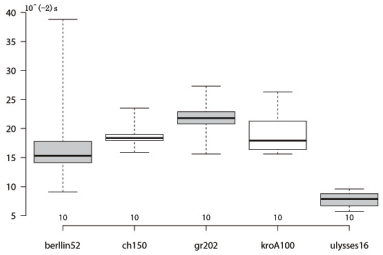
The table is in the last of the report. And for the 2 local search algorithms, the number in the brackets is the standard deviation about running time and length.

5.3 Boxplot

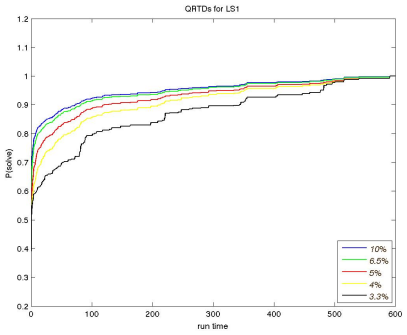
The following boxplot shows the running time distribution for local search algorithms. Because for burma14, because the initial solution obtained from the greedy algorithm is the optimal solution, so this algorithm did not take time to run. For the other 3 dataset, the hillclimbing algorithm could not stop in 600 seconds, so we only make the box plot for them.

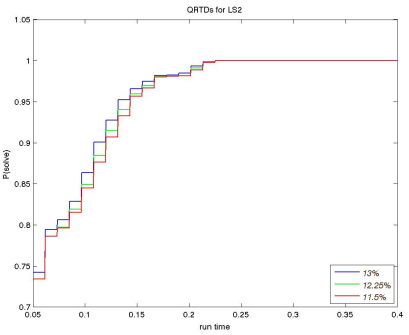




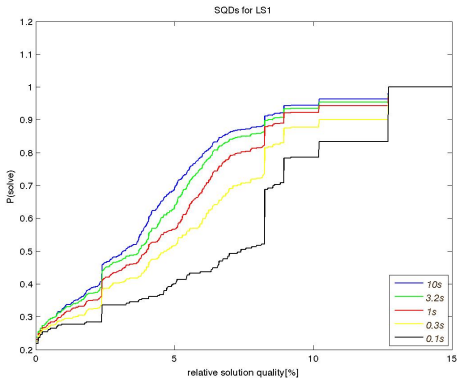


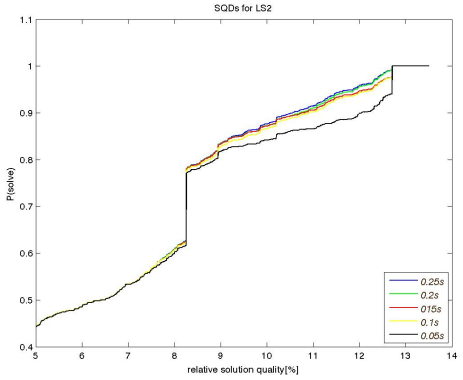
5.4 QRTDs





5.5 SQDs





6. DISCUSSION

In the result table, we could see that except 2-approx, each algorithm could reach some optimal solutions for some dataset. Because the greedy algorithm performs so well that two local search algorithm could get better results based on this wonderful initial solution. Also, our algorithms get quite good solution. Except 2-Approx algorithm, all the results obtain less than 20% of relative errors from the optimal solution, which is much smaller than the guarantee bound of some algorithms.
For the boxplot, we could see that for different dataset has different running time range. For the 16 dimension dataset, the time range is quite large while the berlin52 has smaller time range with local search 1.

From the graphs above for SQD and QRTD, we can see that for the first lcoal search algorithm — Hill Climbing, even if we only run the algorithm in a short time, for example, 0.1 s, the probability of achieving an approximation whose relative error is less than 10% is about 0.8. Hence, we can see that the Hill Climbing algorithm is very effective. Of course, in some sense this relies on the result of the greedy algorithm since we choose its solution as the initial solution for Hill Climbing. But we also do the experiment without using the greedy algorithm result. The experiment also shows that the performance of the Hill Climbing is like the behavior shown in the graph. However, the graph also demonstrates that if we want to update the approximation obtained in first several seconds, the time cost is very expensive and the effect is not good. This implies that in the essential part of Hill Climbing is the take-turns using of 2-opt and 3-opt. This part dramatically determines the final result. If we are fortunate enough, we can obtain the optimal solution in several seconds. Otherwise, we need a lot of time to approach the optimal solution. And since we choose random tours for updating. The convergence of the algorithm to the optimal solution is not guaranteed. From the QRTDs plots, we could see that hill climbing algorithm has the similar tendency with different relative errors. But the smaller the relative error, the slower the probability increases, especially for the relative error 3.3%. And when the running time reaches 500s, almost all curves reach very close to 100% probability. Thus, we could know that we could spend less time to get the similar result. For SQD plot, with diffrent running time, hill climbing has different curves. Especially, for 0.1s running time, there are several huge increasing of the probability, for example, when the relative error reaches about 8% and 12%. But for others, such huge increasing is not common.

For Simulated Annealing, we can see that not like the Hill Climbing algorithm. The algorithm itself ends in very short time. Hence, it is not like the Hill Climbing, which continuely searching for new better solution. Simulated Annealing algorithm totally relies on the first several seconds(in most case, with in 1 seconds). The graph for QRTD shows that the curves are more similar to each other. And when time reaches 0.25s, all curves comes to 100% probability. Therefore, in the same cut-off time, we could get smaller relative error so that these two local search algorithms are efficient.For the SQD plot, we could see that hill climbing performs quite different from simulated annealing. And for the SQD, all curves have an obvious jump to a higher probability when the relative error reaches about 8.5%.

7. CONCLUSION

In this project, we try to solve the intractable symmetric TSP in five different methods, which are Greedy Heuristic, MST Approximation, Branch and Bound, two Local Search methods — Hill Climbing and Simulated Annealing. In the process of doing the project, we also introduce some innovative ideas for our algorithms.
For example, we use different selection strategy, which greatly improve the performance of the algorithm. Furthermore, it provides a good initial solution for our local search algorithms; we also use the idea of taking turns in applying 2-opt and 3-opt in the hill climbing algorithm. We propose this idea because the disadvantage of hill climbing is that it is easy to arrive at a local minimum and cannot improve any more. We think that a local minimum for 2-opt might not be a local minimum for 3-opt and vice versa. So by using the above idea, it is not easy for our algorithm to get stuck at a local minimum. And this can help us get a good solution even if eventually a local minimum is reached. If this happens and the time is still enough, we start from another random initial solution and repeat the process to achieve another solution which is local opt for both 2-opt and 3-opt. If the new local minimum is less, then we update our result. We also successfully realize the other algorithms. And for these algorithms to perform better, we also do some research. For example, we carefully choose the coefficients which are used in the SA algorithm; we well design the branch and bound scheme to search as many solution as

possible in a limited time, etc.
The empirical analysis of the results above can help us better understand the time-accuracy trade-offs across the different types of algorithms. From the table, we can see for every algorithm we get a good approximation for the optimal solution. In many cases, we even obtain the optimal solution. From these results, we can see our algorithms are good implementation of the methods we choose to realize.

8. REFERENCES

[1] Rosenkrantz D J, Stearns R E, Lewis P M. Approximate algorithms for the traveling salesperson problem[C]//Switching and Automata Theory, 1974., IEEE Conference Record of 15th Annual Symposium on. IEEE, 1974: 33-42.

[2] http://www2.isye.gatech.edu/~mgoetsch/cali/VEHICLE/TSP/TSP015_.HTM

[3] http://algs4.cs.princeton.edu/43mst/IndexMinPQ.java.html

[4] Park M W, Kim Y D. A systematic procedure for setting parameters in simulated annealing algorithms[J]. Computers & Operations Research, 1998, 25(3): 207-217.

Table 1: Experimental results for all five algorithms on all six datasets.

Dataset	Branch and Bound			Farthest Insert			2-approx MST			Local Search I			Local Search II		
	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr
burma14	4.158	3323	0	0	3323	0	0.003	4003	0.205	0.000(0)	3323(0)	0	0.000(0)	3323(0)	0
ulysses16	587.562	6859	0	0	7023	0.024	0.004	7788	0.135	0.006(0.002)	6859(0)	0	0.078(0.012)	6859(0)	0
berlin52	600	8578	0.137	0.001	8312	0.102	0.017	10402	0.379	1.035(0.888)	7542(0)	0	0.173(0.080)	7783(158.86)	0.032
kroA100	600	25347	0.191	0.006	23186	0.089	0.036	30516	0.434	600(0)	21292(0)	0.0004	0.187(0.033)	22185.9(386.368)	0.042
ch150	600	6958	0.066	0.031	7067	0.083	0.043	9126	0.398	600(0)	6634(8.238)	0.016	0.186(0.019)	7063(12.6)	0.082
gr202	600	46673	0.162	0.031	45263	0.127	0.043	52615	0.310	600(0)	41282(82.641)	0.028	0.235(0.038)	43647(525.981)	0.087