Traveling Salesman Problem (TSP)

Project Proposal

Leonardo Camacho A.
Georgia Tech
leca6@gatech.edu

Anirudh Choudhary
Georgia Tech
achoudhary46@gatech.edu

Aditya Vadhavkar
Georgia Tech
avadhavkar3@gatech.edu

Abstract

NP problems are constantly being evaluated using different approximations techniques and redesigning the problem based on accuracy and run-time. Therefore, five different approaches to TSP, which is an NP problem, are implemented. The methods were chosen based on their proven accuracy, lower-bound ratio, computational run-time, and latest improvements and implementations. The five methods being compared are a Branch-and-Bound exact solution, a Heuristic construction model, and three local search approximations, which are compared through tables and graphs showing accuracy and running time expense of the algorithms. Having done so, in terms of run-time and accuracy, an improved version of Simulated Annealing (LBSA) was the best approach for the TSP problem.

Keywords

NP problems, 2-Opt, MST Approximation, Local Search, Branch-and-Bound, Simulated Annealing

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the most studied NP problems due to its flexibility to model different problems present in multiple fields of study ranging from aerospace to genetic engineering. Informally, TSP is trying to find a minimum length cost from an origin point (city) that passes through all n vertices of a graph composed of cities and gets back to the origin, generating a complete cycle. Therefore, TSP has been studied and approached on different ways in order to assimilate a given problem closer to reality. Being an NP problem, a brute force approach will prove to be inefficient as the number of vertices increases. An exact solution can be found using a comprehensive exploration algorithm like Branch and Bound but it is not computationally feasible to apply this algorithm to real life examples. Other approaches involve approximations to the solutions, which have proven to give quality results without hurting the computational running time. 2-Opt exchange is implemented in Local Search algorithms, such as Iterative Local Search and Simulated Annealing. Construction Heuristics section include MST (Minimum Spanning Tree) Approximation. Through this study, tables and plots representing accuracy and running time of the different algorithms will be provided in order to conclude the quality of the different methods on different sized graphs. From related work section it is clear that branch and bound algorithm is expected to become exponentially inefficient as the graph size increases, the local search algorithms are expected to provide a quick and accurate solution, although enhancing the quality of solution is one of the difficult tasks, and the heuristic construction are expected to be the fastest approach with high error rates.

2 PROBLEM DEFINITION

The TSP can be understood as follows: Given a graph G = (V, E), where nodes, cities, u, v ϵ V (set of vertices, and (u,v)ϵ E (set of edges), minimize the sum of costs, c(u,v), of a cycle passing through all nodes in V, while obtaining a Hamiltonian cycle.

$$\sum_{i=1}^{n} c_i(u, v) + c_n(v, u)$$

The problem is simplified by letting G being an undirected graph. Henceforth,there is no edge (u,v) where c(u,v) not equal to c(v,u), which is the condition of an asymmetric graph.

3 RELATED WORK

Construction Heuristics

Construction heuristics are algorithms which determine a tour based on some construction rules but don't improve over the tour iteratively[1]. The simplest construction heuristic algorithm is the nearest neighbour algorithm. In nearest neighbor algorithm, the salesman starts from a city and the next city is the one which hasn't been visited and is nearest to the current city. Though nearest neighbour leads to sub-optimal solutions, it is a good starting point for Local Search algorithms. The minimum spanning tree (MST) approximation computes the MST using Prim's algorithm and tries to shortcut the paths using Eulerian Tour built on top of MST. More accurate implementations include Christofides algorithm and Clarke-Wright algorithm.

Branch and Bound

Branch and Bound creates a state space containing all the nodes (cities) and their connections (edges) to other cities. The state space is represented in the form of a matrix where an element in row 'i' and column 'j' represents the distance between the city 'i' and city 'j'.

Local Search - Solution Approximation

Local search algorithms have achieved down to a percent of error of around 1-4 percent in large graphs, as different variations of Lin-Kernighan (LK) show. Even more, as shown in [6], the ability to combine different techniques on algorithms such as k-opt and stochastic methods, without affecting the run time of the model serve extensive study and implementation of such models. Some examples of the previous may be, simulated annealing, tabu search, and random walking. However, the study will be focused on simulated annealing and iterated search in order to compare an improved version of the rudimentary method that iteration provides compared to the more detailed model simulated annealing provides.

4 ALGORITHMS

4.1 Construction Heuristics

4.1.1 MST Approximation

The MST Approximation algorithm is a 2-Approx algorithm which uses Minimum Spanning Tree as the starting point and shortcuts the Eulerian tour to find the optimum path

Algorithm

(1) Given input points, Minimum Spanning Tree is computed using Kruskal's algorithm.
(2) The edges in the Minimum Spanning Tree are sorted in the increasing order of their weights. Hence, it is directly used for preorder traversal. Both permutations of vertices in MST are included for traversal.
(3) Depth First Search is implemented for traversal.

Data Structures

We use a priority queue to extract the edges while implementing Kruskal's algorithm. We use a list of tuple of nodes and the distance between them for computing Minimum Spanning Tree and performing Depth First Search

Time Complexity

We perform 2 steps for computing the Hamiltonian cycle. In this case, since it is a dense graph, the time complexity of Kruskal's algorithm is $O(N^2 logN)$. This represents the time complexity of the algorithm since Depth First Search has time complexity of $O(E + N) = O(N^2)$.

4.2 Branch and Bound

Branch and Bound[4] creates a state space of all the cities (nodes). Every edge has a cost matrix. The

cost matrix consists of the distance between the nodes. Locations corresponding to nodes with distance from self are populated with infinity. Whenever a particular edge is selected, the row corresponding to the source node and the column corresponding to the target node of that edge in the cost matrix are set to infinity. Then for each row and column the minimum value is computed and that value is subtracted from the respective row or column. This converts the cost matrix to a reduced cost matrix which has a property that each row and column must at least have one zero element.
A node is expanded based on the value that it gets assigned by adding the cost to transform the node's corresponding cost matrix into a reduced cost matrix, the cost of converting the previous node's matrix into reduced matrix and the cost of the edge between the two nodes. Each node from the starting position is assigned a value according to the above procedure and the edge corresponding to the node with the least value is selected for further expansion. This process is repeated till the leaf node in the state space tree is reached. The final cost that we get for the leaf node is the cost of the path.

Data Structures
A two dimensional array (matrix) is used to represent a cost matrix of each node.

Time Complexity
The worst case time complexity for Branch and Bound is same as that for brute force approach which will be O(N!).

4.3 Local Search - Solution Approximation

2-OPT

The 2-OPT is probably the most basic local search [3] and, hardly-arguably, the most used heuristic to approximate the TSP problem. Therefore, it is important to point out that 2-OPT does not require the construction of an MST in order to perform well. However, it is known that the initial cycle given to the 2-OPT algorithm will affect significantly both the accuracy and running-time of it due to the simplicity of steps the 2-OPT algorithm performs. Therefore, as MST gives a minimal solution to the distances between the nodes of a given graph, it was used as reference to the creation of the initial cycle.

Algorithm

(1) Generate a solution based on greedy choices, which gives an MST and make a tour of vertices that creates a cycle from node i, passing through all nodes, and coming back to i. This cycle does not necessarily need to be Hamiltonian, aka tour is allowed to pass through vertices more than once.
(2) From route generated from the MST, remove duplicate nodes in the cycle and connect the cycle using edges present and generating new edges with cost c(v,b) based on Euclidian distance between nodes. Save the cost of the tour (sum of costs)
(3) Change tour by exchanging two edges, and record the cost of the new tour. If the cost is less than the current best one, replace the current best one with the new tour. Repeat until no further improvement is seen.

4.3.1 Iterative Method

The approach implemented has as base the basic Iterative Method for Local Search. However, in order to prevent local minimas, a series of steps and conditions were added to the method.
At first, the approach use the MST cycle, previously mentioned, in order to converge to a solution. However, instead of consistently using the best optimal solution so far at every iteration, the approach tends to randomize completely the route at its first iterations without affecting the root node. This idea finds its roots in Q-learning, as we randomize the input in order for our method to explore as many different solutions as it can find, but as the number of random cycles increases, the probability

Table 1: Experimental Results for all algorithms on all datasets ($exceptUKansasState$)**.** $Time$ **is in** $sec.$ **and** $RelativeError(RelErr)$ **is computed basis** $(AlgMinPath - OptPath)/AlgMinPath$**. Concorde TSP Solver is used to determine the baseline OptPath.**

	Branch and Bound			MST-Approx			Iterative Method (2-OPT)			Simulated Annealing		
Instance	Time	Sol. Qual.	RelErr.	Time	Sol. Qual.	RelErr.	Time	Sol. Qual.	RelErr.	Time	Sol. Qual.	RelErr.
Atlanta	600.00	3106652	**0.36**	0.01	2488307	**0.19**	9.84	2003763	**0.00**	0.14	2008531	**0.00**
Berlin	600.00	19721	**0.62**	0.00	9967	**0.24**	137.78	8130	**0.07**	6.09	7618	**0.01**
Boston	600.00	2201193	**0.59**	0.01	1093837	**0.18**	459.92	898591	**0.01**	2.17	910120	**0.02**
Champaign	600.00	212027	**0.75**	0.01	64760	**0.19**	183.06	54381	**0.03**	7.18	52836	**0.00**
Cincinnati	1.81	277952	**0**	0.00	308133	**0.1**	0.02	277952	**0.00**	0.01	278754	**0.00**
Denver	600.00	553214	**0.82**	0.02	127081	**0.21**	253.01	113351	**0.11**	19.18	101947	**0.01**
NYC	600.00	7439017	**0.79**	0.02	1935350	**0.2**	192.11	1626765	**0.04**	12.81	1598185	**0.03**
Philadelphia	600.00	3232667	**0.57**	0.01	1727842	**0.19**	25.90	1395981	**0.00**	1.20	1406972	**0.01**
Roanoke	600.00	6889302	**0.9**	0.19	804046	**0.18**	600.00	786046	**0.17**	128.33	676554	**0.03**
SanFrancisco	600.00	5543943	**0.85**	0.03	1098155	**0.26**	353.23	912013	**0.11**	20.32	848199	**0.05**
Toronto	600.00	9512153	**0.88**	0.03	1630349	**0.28**	102.34	1279954	**0.08**	20.53	1195736	**0.02**
ulysses16	600.00	7263	**0.06**	0.00	7700	**0.11**	4.20	6864	**0.00**	0.40	6859	**0.00**
UMissouri	600.00	644379	**0.79**	0.04	170535	**0.22**	211.08	148776	**0.11**	25.62	135231	**0.02**

for the algorithm to generate another randomize cycle decreases. Then, randomization steps keep taking place, but on different sections of the best cycle found so far.

Therefore, we can expect the approach to be better than the basic iterative method, which converges to an optimal solution and considers it a global optimal solution. However, beware that the exchange optimization step is still done with 2-opt, limiting the amount of combinations the optimization step could make, but performing faster than any other k-opt approach due to the time complexity of this approach O(n^k) [2]. Having said so, the solution reached in the period of time and number of iterations may not be the optimal, despite the randomization steps.

The initial parameters used were: $rar(randomness) = 0.99$, and $radr(randomnessdecayrate) = 0.95$.

Algorithm

(1) Find optimal solution using 2-OPT and store solution if it is the best one so far.
(2) Using the current best route found, check if by starting from a different node, keeping the same route, the solution is improved. If it is improved, then store solution. e.g. [1,2,3,4,1] to [1,3,4,2,1].
(3) As the algorithm will avoid getting stuck in a solution, no matter how good, it checks if the solution has converged by checking local best previous solution to local best current solution. If no, go to step , else continue.
(4) Check if a random probability is less than the randomness. If yes, randomize route without affecting first and last node, and multiply the decay rate to rar (randomness). Otherwise, based on the random probability, choose what section of the global optimal route should be randomized.
(5) Check if the global best solution has changed. Add one to the count if it has not, and make the count equal to 0 if it has changed. Check if time limit is met, check if number of iterations have exceeded limit. If yes, finish. Otherwise, return route from previous step to step 1.

It is important to highlight two things. The second step can be taken care of through the 2-OPT step, but it is considered to optimize the solution with a simple low computational cost change. Also, make notice that the number of iterations was not given a value, as it is important to discuss further the implications of the size of iterations. The bigger the limit the more accurate the algorithm was, but there was a lot of extra computation that was unnecessary once the optimal solution was found. On the other hand, if the limit is small, the accuracy of the approach decreased significantly for bigger graphs, but no unnecessary extra computations were performed. For the sake of accuracy, the results provided were based on a limit of 100,000 iterations.

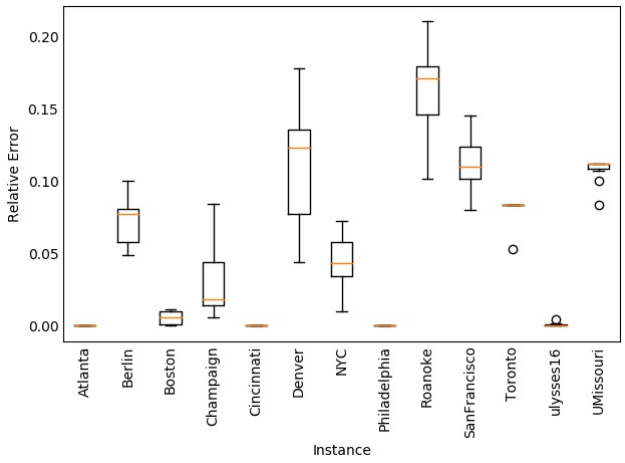

Fig. 1: Relative Errors - Iterative Method

Time Complexity

Same as 2-OPT the time complexity is $O(n^2)$. It is important to notice that the iterative method will continue to run until the time constraint is met or the number of iterations reaches 100,000.

4.3.2 Simulated Annealing

Simulated Annealing starts with a solution derived from simple heuristic (based on greedy algorithm)

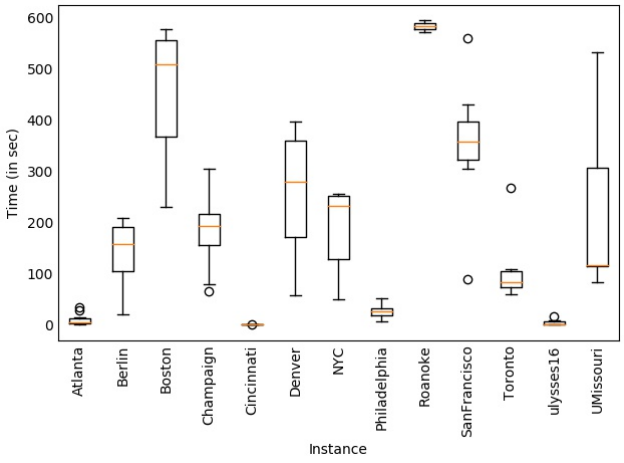

Fig. 2: Runtime - Iterative Method

for finding the shortest route. Initially, the temperature is kept higher so that the probability of accepting a worse solution is higher. This gives the algorithm a chance to jump out of any local optimum it finds early on during execution. As the temperature is reduced, the chance of accepting worse solutions reduces, therefore allowing the algorithm to gradually focus in an area of search space where optimum solution is located. To generate a new solution, we use either reversing of path between 2 randomly selected nodes or transposing of segments within the path. The probability of reversing is kept to be 80% while transposing is used mostly to move out of local minima when the algorithm gets stuck. The final solution quality depends on the solution used for initialization and hence, random path generation isn't an optimal way to start.

Algorithm

(1) Compute initial path S using Greedy method. Set initial temperature = 1000 and cooling coefficient = 0.99.
(2) Generate S' from S using reversing/3-opt operations.
(3) Accept S' if Δ = length(S') - length(S) $\leq$ 0 or accept S' with probability exp(-Δ/T)
(4) If maximum iterations reached or equilibrium is reached, reduce temperature.
(5) Repeat steps 2-5 until temperature > 0.0001

Data Structures

We utilize an adjacency matrix to store the distances between nodes and lists to store the paths. The complexity of using a adjacency matrix is O(N^2) but it speeds up the implementation significantly. For larger graphs, we use the 'NetworkX' package based Graph object to calculate path length which is more efficient since it stores half the edges.

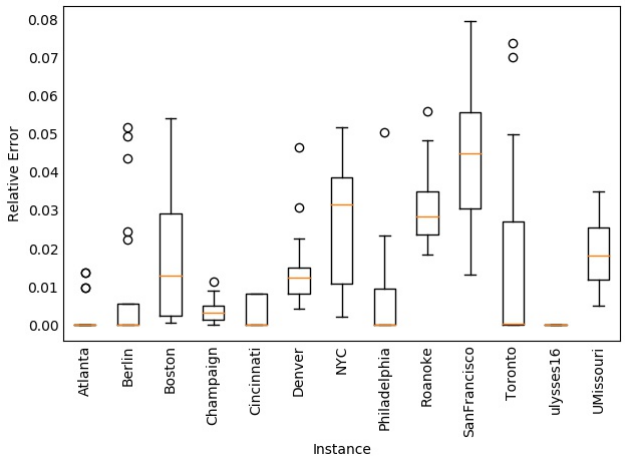

Fig. 3: Relative Errors - Simulated Annealing

Time Complexity

The number of steps at each temperature are of the order O(N). Hence, the time complexity of the annealing part is O(NK) where K is the number of cooling steps the algorithm undergoes. However, we have to compute distance matrix for all pairs of nodes initially which is O(N^2). Hence, time complexity is O(N^2).

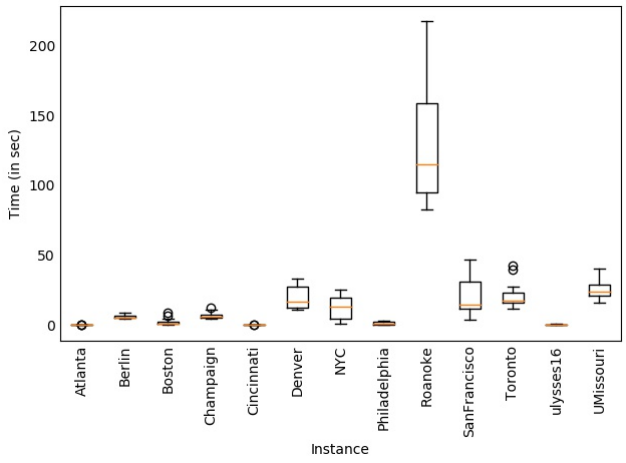

Fig. 4: Runtime - Simulated Annealing

4.3.3 List based Simulated Annealing (LBSA)

Simulated Annealing's accuracy significantly depends on the starting temperature and the cooling schedule followed. To overcome this dependency, S. Zhan et. al[7] proposed a List based Simulated Annealing (LBSA) algorithm which algorithm uses a list-based cooling schedule to control the temperature drop. Based on the initial Greedy Solution, a list of temperatures is created and the maximum temperature in the list is used in the acceptance criterion. The list is adapted iteratively as per the topology of the solution space and the algorithm is quite robust to parameter initialization. LBSA has shown competitive performance compared with state-of-the-art Simulated Annealing algorithms which are based on instance-based sampling and greedy search.

Algorithm

(1) Compute initial path S using Greedy method. Create initial temperature list by generating new solution(y) using the equation below and

Table 2: Comparison of relative errors and times for Local Searches

	Iterative Method (2-OPT)			Simulated Annealing			LBSA		
Instance	Time	RelErr.	Best Qual.	Time	RelErr.	Best Qual.	Time	RelErr.	Best Qual.
Atlanta	9.84	0.00	**2003763**	0.14	0.00	**2003763**	1.00	0.00	**2003763**
Berlin	137.78	0.07	**7930**	6.09	0.01	**7542**	4.42	0.01	**7542**
Boston	459.92	0.01	**893536**	2.17	0.02	**894197**	3.14	0.00	**893536**
Champaign	183.06	0.03	**52943**	7.18	0.00	**52643**	8.47	0.00	**52643**
Cincinnati	0.02	0.00	**277952**	0.01	0.00	**277952**	0.15	0.00	**277952**
Denver	253.01	0.11	**105079**	19.18	0.01	**100860**	19.66	0.02	**100498**
NYC	192.11	0.04	**1570522**	12.81	0.03	**1558565**	8.82	0.02	**1555060**
Philadelphia	25.90	0.00	**1395981**	1.20	0.01	**1395981**	1.86	0.00	**1395981**
Roanoke	600	0.17	**729978**	128.33	0.03	**667833**	96.03	0.03	**663202**
SanFrancisco	353.23	0.11	**880831**	20.32	0.05	**820995**	27.86	0.03	**810196**
Toronto	102.34	0.08	**1242504**	20.53	0.02	**1176151**	35.33	0.05	**1176151**
ulysses16	4.20	0.00	**6859**	0.40	0.00	**6859**	0.62	0.00	**6859**
UMissouri	211.08	0.11	**144831**	25.62	0.02	**133393**	35.15	0.03	**133456**

updating the current solution(x) if y is better

$$t = \frac{-(f(y) - f(x))}{ln(p_0)} \tag{1}$$

(2) Generate S' from S using the best operation out of swap,reverse and insert. When the algorithm gets stuck at a minima, transposing is used along with these operations.
(3) Determine the acceptance probability using $p_i = -\Delta/t_i$. Generate random number and if its less than p_i, accept the bad solution.
(4) Also, compute the temperature based on formula in equation (1) and insert it in the list. This temperature is always less than the maximum temperature taken from the list since the random number is less than the probability.
(5) Repeat steps 2-5 until the maximum iteration count is reached. If the number of iterations during which minimum distance remains constant exceeds a threshold, stop the algorithm.

Data Structures and Time Complexity

The data structures and time complexity are similar as in the implementation of Simulated Annealing. The length of temperature list is fixed at 120 and the number of iterations 'K' is limited to 10000.

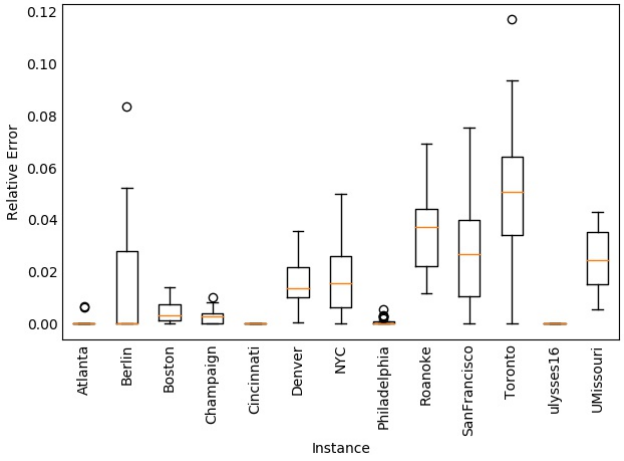

Fig. 5: Relative Errors - LBSA

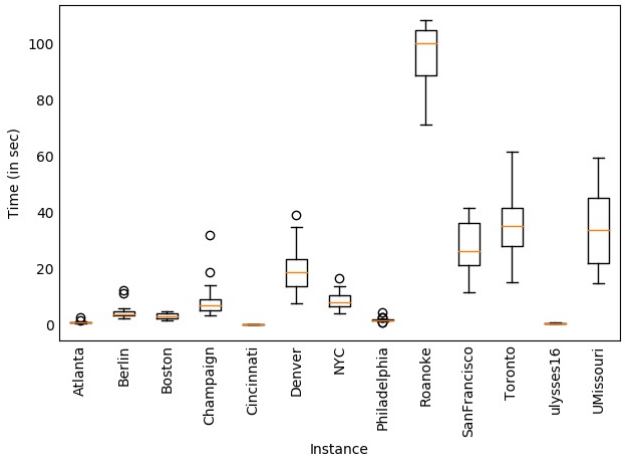

Fig. 6: Runtime - LBSA

5 EMPIRICAL EVALUATION

Platform Details : The programming platform is Python 3.6. The system has a RAM of 8 GB and 2.7 GHz Intel i7 processor with Microsoft Windows platform.

The programs are run using the given .tsp files and this procedure is repeated multiple times for Local Search algorithms. We have plotted SQD, QRTD and box plots to study the time vs solution quality behaviour of various algorithms. For an analysis of timings and relative errors, please refer to Table 1 and Table 2. Table 2 gives an analysis of relative error along with the best solution found for Local Search Algorithms. Box plots for the *SolutionComputationTime* and *RelativeErrors* for each algorithm are included in Figures 1 - 6. We have utilized **Concorde software toolkit** [5] to compute the baseline solution with which we compare our best solution. All results have been obtained within the execution time limit of 10 minutes.

While, MST-Approximation algorithm gave a lower bound in the range 0.1 - 0.25, Local Search algorithms performed significantly better albeit with higher Execution Time. LBSA achieved highly accurate solutions in many instances as expected. Branch and Bound algorithm could finish execution only for *Cincinnati.tsp* instance with an exact solution and gave a solution for ulysses16.tsp within 6% in 10 minutes. *Roanoke.tsp* seems to be a relatively complex problem to solve given the computation time required by all algorithms is the highest and *BranchandBound* has highest *RelativeError* for that instance within 10 mins.

6 DISCUSSION

Branch and Bound algorithm is a comprehensive exploration algorithm that tries to expand all possible combinations of states and return the best possible route corresponding to least path cost. Due to it's nature of exploring all possible states, it becomes computationally unfeasible to get an output for large graphs using Branch and Bound although, it is guaranteed to give an optimal route if it is allowed to run for a sufficiently long time.

The ideal compromise between execution time and solution quality are Local Search algorithms.Three of these were analysed. Iterative Method involved 2-OPT moves with randomization while Simulated Annealing involved 2-OPT, 3-OPT and additional moves like Swap, Insertion which have been found to be quite effective [7]

As we can see from the comparisons made in Table 1, and Table 2, the best performance regarding accuracy and execution time was LBSA. This is further supported by Figures 7 and 9, and Figures 8, and 10, in which we can visually compare both approaches. The comparison was made based on the worst performance of Local Search algorithms, which in our case is Iterative method, against the best one, LSBA.

LBSA reduces the execution time as well as gives more accurate results compared to Simulated Annealing. This is because the temperatures which its initialized is more adaptive to the search space and the rate of cooling is slower. Moreover, for Simulated Annealing, doing the transpose operations (equivalent to 3-opt) when its stuck in local

minima helps in achieving better solution faster. Also, initiating with a nearest neighbour based solution vs randomized solution is a key factor and we observed significantly lower runtime when initializing with a sub-optimal solution.

7 CONCLUSION

We have analyzed 5 algorithms for solving the Traveling Salesman problem over various instances of TSPLIB datasets. We achieved quite good accuracy using Local Search Methods and evaluated the merits and drawbacks of our approaches. We hope to fine tune the implementation of LBSA for a highly optimized algorithm in the future

REFERENCES

[1] Marshall Bern and David Eppstein. 1997. Approximation Algorithms for NP-hard Problems. PWS Publishing Co., Boston, MA, USA, Chapter Approximation Algorithms for Geometric Problems, 296–345. http://dl.acm.org/citation.cfm?id=241938.241946

[2] Andrius Blazinskas and Alfonsas Misevicius. 2011. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. *Kaunas University of Technology, Department of Multimedia Engineering, Studentu St* (2011), 50–401.

[3] Matthias Englert, Heiko Röglin, and Berthold Vöcking. 2007. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1295–1304.

[4] Leo Liberti. 2015. Branch and Bound for the Travelling Salesman Problem. (2015). http://www.enseignement.polytechnique.fr/informatique/INF431/X09-2010-2011/AmphiLL/branch_and_bound_for_TSP-notes.pdf

[5] University of Waterloo. 2003. Concorde Software Toolkit. (2003). http://www.math.uwaterloo.ca/tsp/concorde/

[6] César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. 2011. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research* 211, 3 (2011), 427–441.

[7] Shi-hua Zhan, Juan Lin, Ze-jun Zhang, and Yi-wen Zhong. 2016. List-Based Simulated Annealing Algorithm for Traveling Salesman Problem. *Intell. Neuroscience* 2016 (March 2016), 8–. https://doi.org/10.1155/2016/1712630

8 APPENDIX

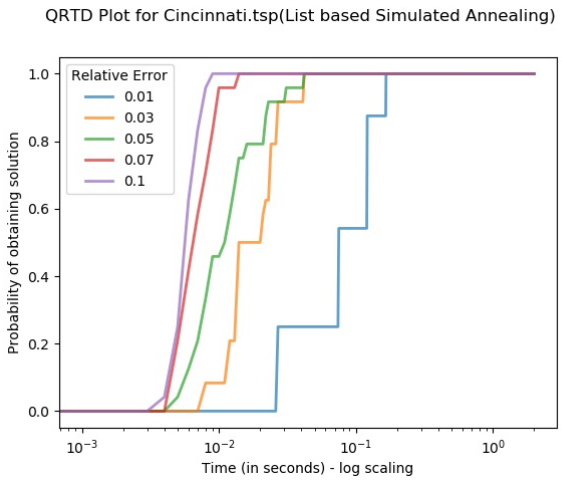

Fig. 7: QRTD Plot - Cincinnati (LBSA)

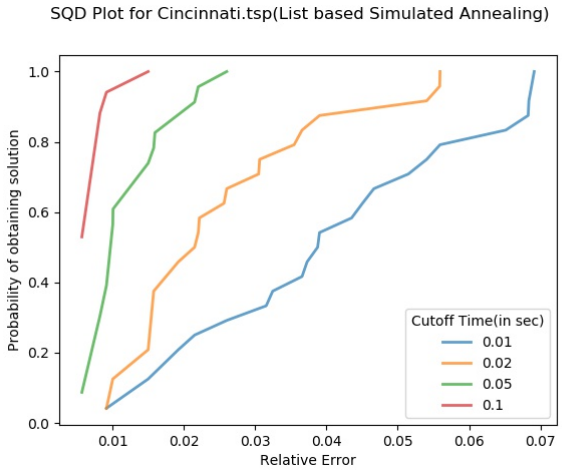

Fig. 8: SQD Plot - Cincinnati (LBSA)

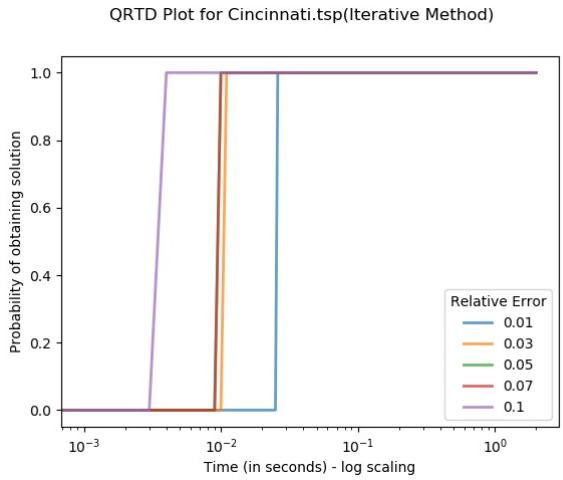

Fig. 9: QRTD Plot - Cincinnati (Iterative Method)

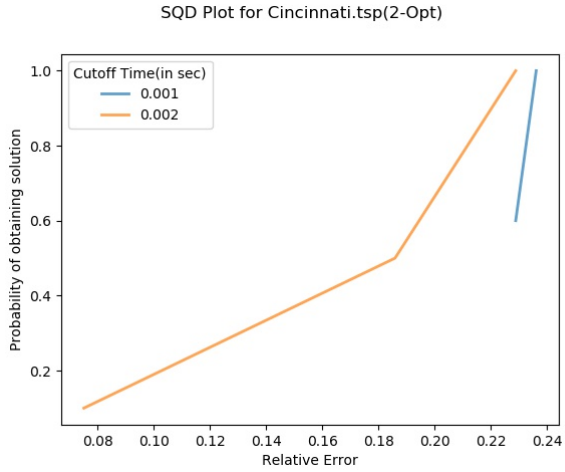

Fig. 10: SQD Plot - Cincinnati (Iterative Method)

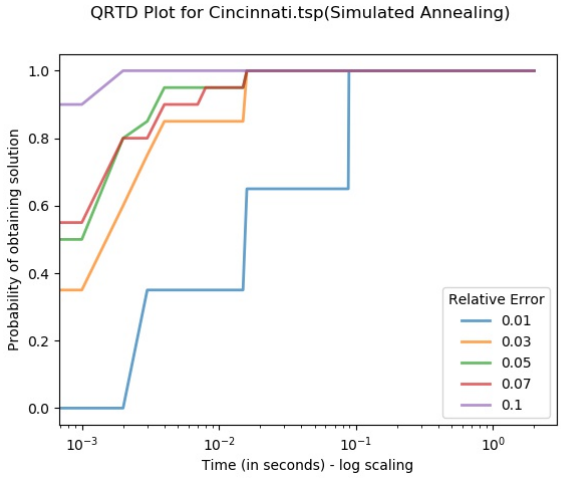

Fig. 11: QRTD Plot - Cincinnati (SA)

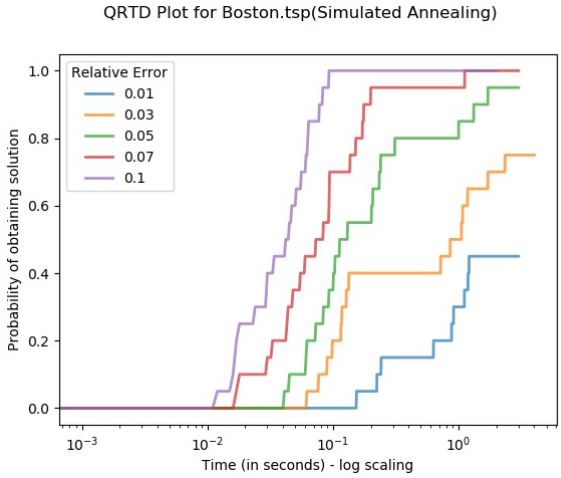

Fig. 12: QRTD Plot - Boston (SA)

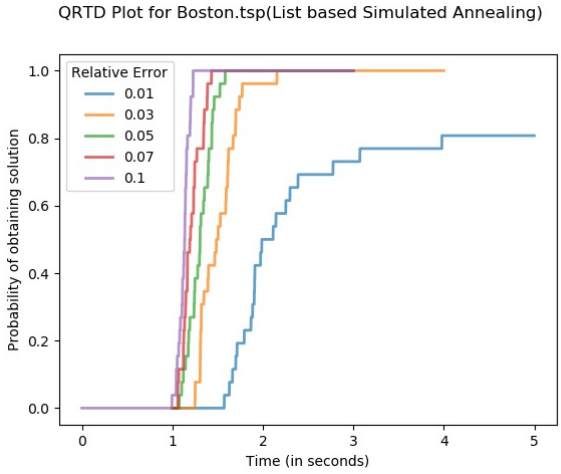

Fig. 13: QRTD Plot - Boston (LBSA)

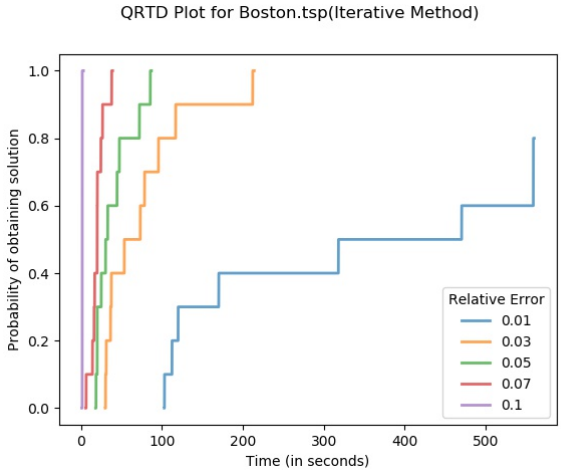

Fig. 14: QRTD Plot - Boston (Iterative Method)

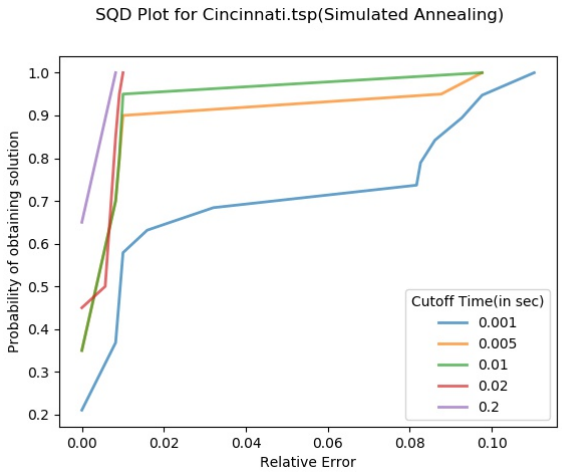

Fig. 15: SQD Plot - Cincinnati (SA)

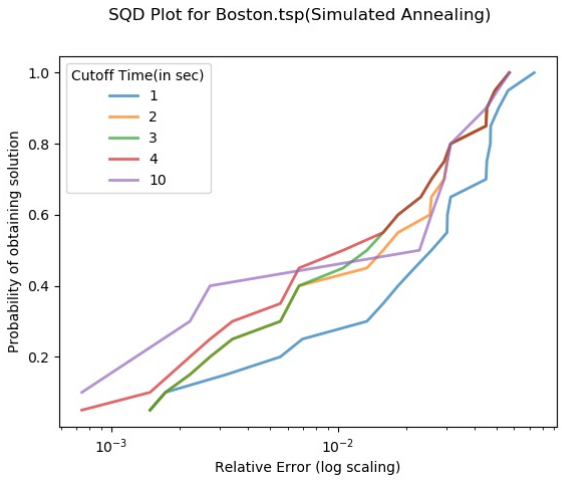

Fig. 16: SQD Plot - Boston (SA)

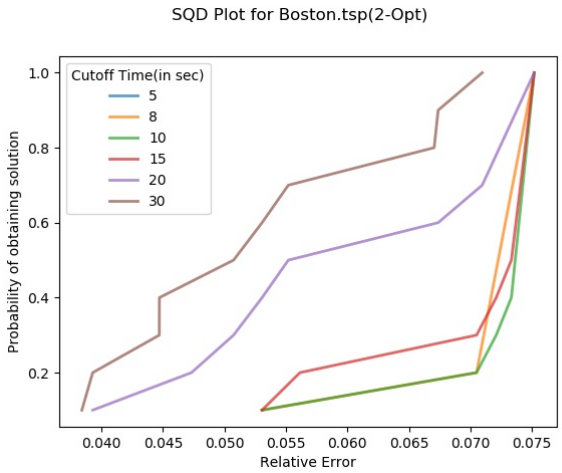

Fig. 17: SQD Plot - Boston (Iterative Method)

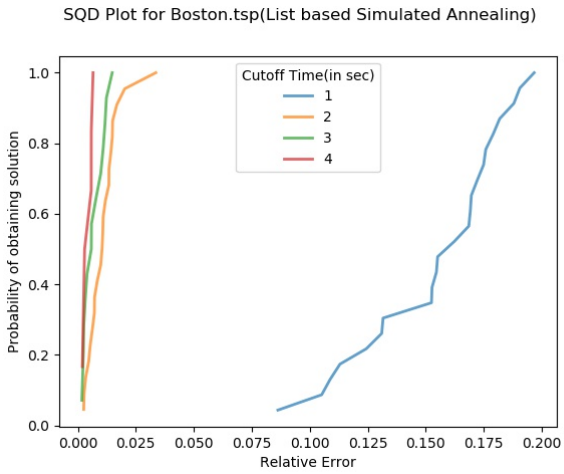

Fig. 18: SQD Plot - Boston (LBSA)