Traveling Salesman Problem

CSE 6140 Fall 2014 - Group A

Ran Cui, Richard Ding, Ashwini Khare, and Vikas Kumar
{crboa,richard.ding,ashwini,vkumar78}@gatech.edu

ABSTRACT

The "Traveling Salesman Problem" is one of the oldest and most fundamental problems that has shaped the field of computer science. In this paper, various approaches for solving this problem are proposed and compared. After a formal problem definition, related work on the problem is reviewed, and several algorithms are then proposed that produce solutions for the problem. These algorithms include two construction heuristics (MST Approximation and Farthest Insertion), three local search approximation algorithms (2-OPT, Iterated Local Search, and Simulated Annealing) as well as an exact Branch and Bound algorithm. Runtimes and relative errors for each of the algorithms are compared using tables, QRTDs, SQDs, and box plots. Strengths and weaknesses of the algorithms are discussed, and implications and future directions for research are offered in the concluding remarks.

Categories and Subject Descriptors

F.1.m [**Theory of Computation**]: Computation by Abstract DevicesMiscellaneous

General Terms

Algorithms, Design, Experimentation

Keywords

Traveling Salesman Problem; Construction Heuristics; Local Search; Branch and Bound

1. INTRODUCTION

Perhaps no other problem in computer science has been as extensively studied as the "Traveling Salesman Problem." The informal definition of the problem is as follows: Given a set of cities and the distance/cost of travel between each possible pairs, the problem is to find the best possible way of visiting all the cities and returning to the starting point while minimizing the travel distance/cost [6]. Since its beginnings in the 1800s, various approaches have been proposed for solving the problem. Methods producing the optimal solution include the brute-force method as well as branch and bound algorithms. Approximation algorithms include local search methods, and construction heuristics include the Minimum Spanning Tree Approximation and Farthest Insertion.

In this study, these algorithms are implemented and empirical evaluation of each algorithm is performed using tables, QRTDs, SQDs, and box plots. We find that the branch and bound performs well for small instances (n $<$ 20) but quickly becomes intractable. Construction heuristics are fast but produce large error rates. Local search algorithms seem to offer the best compromise between speed and accuracy.

This paper is organized as follows: Section 2 contains a formal problem definition; Section 3 contains a review of previous work; Section 4 describes each of the algorithms implemented; Section 5 describes the results of the empirical evaluation; Sections 6 and 7 are for discussion and conclusions, respectively.

2. PROBLEM DEFINITION

Given a graph $G = (V,E)$ that has a nonnegative cost $c(u,v)$ associated with each edge $(u,v) \in E$, the traveling salesman problem is to find a hamiltonian cycle (tour) with minimum total cost as represented by the equation $c(A)= \sum_{(u,v)\in A} c(u,v)$, where $A \subseteq E$ [2]. The problem can be further classified as symmetric (sTSP) or asymmetric (aTSP): in the symmetric case, $c(u,v) = c(v,u)$ for all $(u,v) \in E$; in the asymmetric case, this equality does not hold for at least one $(u,v) \in E$ [6].

3. RELATED WORK

3.1 Construction Heuristics

Construction heuristics are algorithms that build a solution from scratch usually using a greedy process [4]. The first solution generated is submitted. Construction heuristics are known to compute results very quickly (most operate on the order of $O(n^2)$), however their relative error may be large; for example, for the nearest neighbor algorithm, the approximation ratio is proportional to the logarithm of N where N is the number of vertices [4], although in practice often better results are obtained.

In the nearest neighbor algorithm, the tour is constructed so as to select the nearest node to the current source node. The minimum spanning tree (MST) approximation applies

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Table 1: Experimental results for all five algorithms on all six datasets. Time is in seconds, and Relative error (RelErr) is computed as $(AlgPathLength - OPTPathLength)/OPTPathLength)$. Bold values for RelErr highlight the closest algorithm to the optimum for a given dataset.

Dataset	Branch and Bound			MST Approximation			Farthest Insertion			2-OPT			Simulated Annealing		
	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr
burma14	2.82	3323	**0**	2.91e-5	4162	**0.25**	2.72e-5	3808	**0.15**	0.026	3507	**0.05**	4.89e-2	3347	**0.01**
ulysses16	600	6870	0.001	4.42e-5	7815	**0.14**	3.36e-5	7598	**0.11**	0.050	6840	**0.01**	6.00e-2	6789	**0.01**
berlin52	-	-	-	2.85e-4	10220	**0.36**	4.27e-1	9518	**0.26**	0.198	8345	**0.11**	1.12e-1	7806	**0.04**
kroA100	-	-	-	6.58e-4	28592	**0.26**	2.22e-3	30592	**0.44**	0.771	23161	**0.09**	2.16e-1	22505	**0.06**
ch150	-	-	-	1.41e-3	9206	**0.41**	6.50e-3	8767	**0.34**	1.238	7085	**0.09**	2.97e-1	7094	**0.09**
gr202	-	-	-	6.58e-3	52862	**0.32**	1.24e-2	52818	**0.32**	2.071	43456	**0.08**	4.17e-1	43501	**0.08**

the MST algorithm (usually Prim's algorithm) to the graph to compute the shortest tour starting from a root node. Other construction heuristics include the Greedy algorithm, the Clarke-Wright algorithm, and the Christofides algorithm.

3.2 Local Search Approximations

A number of local search techniques have been developed to solve the TSP. Such algorithms typically perform better than tour construction heuristics; it has been shown that 3-OPT algorithms can get within 3-4 percent of the optimal solution, and 'variable opt' algorithms such as the Lin-Kernighan algorithm may perform within 1-2 percent [4]. Classic local optimization techniques such as the 2-OPT and 3-OPT exist, in addition to more recently developed variants such as simulated annealing, iterated local search, and tabu search.

3.3 Exact Solutions - Branch and Bound

One of the first Branch and Bound algorithms was proposed by Little et al.[5]; it was based on row- and column-reducing the cost matrix of the instance and calculating the lowerbound of each solution branch. If a particular branch had a lowerbound that was higher than an already obtained solution, that branch was pruned, therefore saving time.

3.4 Other Approaches

Other approaches to TSP include neural networks and genetic algorithms [4].

4. ALGORITHMS

4.1 Construction Heuristics

4.1.1 MST Approximation

The MST Approximation is implemented by shortcutting a Eulerian tour.

Algorithms and Data Structures:.

1. The Minimum Spanning Tree is obtained with Prim's algorithm and stored in an adjacency list.
2. The edges are pushed into heap to ensure the convenience of finding the edge with the minimum weight.
3. The preorder traversal of the tree is implemented with depth first search (DFS).

Guarantee:.

The MST Approximation is a 2-Approximation algorithm (its result has a relative error of no more than 100 percent) [2].

Complexity:.

There are two steps in finding the Hamiltonian Cycle. The running time of Prim is $O(n^2 log n)$. The running time of DFS is $O(n^2)$. Overall the running time is $O(n^2 log n)$, where n is the number of vertices. The space complexity is the space required by the Adjacent List, which is $O(n^2)$.

4.1.2 Farthest Insertion

The Farthest Insertion has been done using the following steps:

Algorithms and Data Structures:.

1. Randomly choose a node i as a starting point.
2. Find the farthest node from i
3. Find a node k not in the subtour that is farthest from any of the subtour nodes
4. Find an edge [i,j] of the subtour to which the insertion of k gives the smallest increase of length
5. Go back to step 3 until a Hamiltonian cycle is formed(all the nodes are visited).

A deque was used to store the tour and a two-dimensional array was used to store the distance matrix.

Guarantee:.

In the worst case, the Farthest Insertion algorithm performs as follows: $(AlgFarthestInsertion/OPTPathLength) = 2ln(n) + 0.16$ [1].

Complexity:.

Step 3 make up most of the time complexity and it takes $O(n^2)$ to finish each time. So the total running time is $O(n^2)$.A deque was used to store the tour and a two-dimensional array was used to store the distance matrix so the space complexity is $O(n^2)$.

4.2 Local Search Approximations

4.2.1 2-OPT

Figure 1: QRTD Plot for the 2-OPT Algorithm.

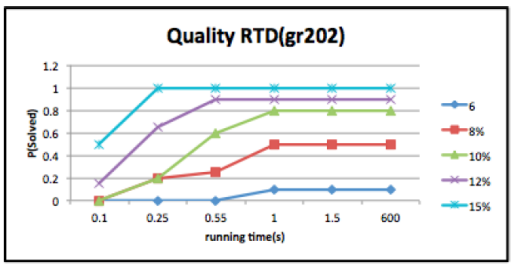
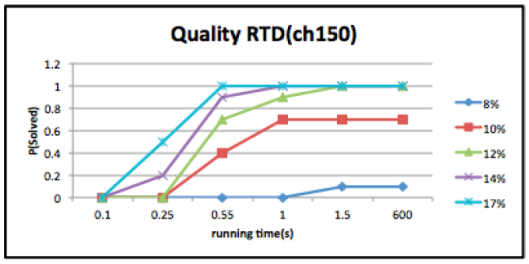


Figure 2: QRTD Plot for the 2-OPT Algorithm.



Algorithms and Data Structures:.

1. Start with a Random Solution. For comparisons, we will start with the solutions given by greedy and MST and perform swaps.
2. Using 2OPT, try swapping any two edges between nodes and evaluate the objective function. If the path length is less than the one previously without swapping, keep the swapped solution, otherwise reject the swapping
3. We continue 2OPT till no more improvements can be achieved, we get a local minimum solution with it.

Guarantee:.
The performance guarantee for 2-OPT is as follows: $(AlgFarthestInsertion/OPTPathLength) = (1/4)\sqrt{N}$ [4].

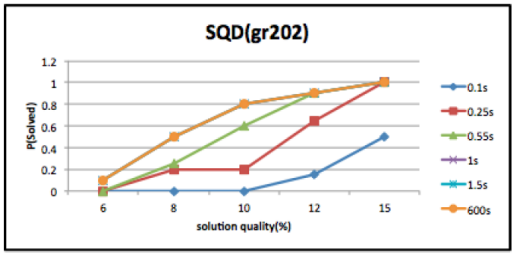
Complexity:.
The complexity of 2-OPT is $O(n^2)$, since at every step the 2-OPT algorithm must examine $O(n^2)$ solutions [3].

4.2.2 Iterated Local Search

Algorithms and Data Structures:.

1. Find locally optimal solution using stochastic 2-OPT and with 5 percent chance of 4-opt perturabtion
2. Perturb solution using a double-bridge move, reverse double bridge move and a random 4-opt, with 45,45,10 percent chance repectively.
3. Repeat until 10mins limit

Figure 3: SQD Plot for the 2-OPT Algorithm.



Guarantee:.
The Guarantee for 4-opt is $(1/4)(N)^{1/2k}$ and for 2-opt, it's $(1/4)\sqrt{N}$ [4].

Complexity:.
The time complexity of the 2-OPT algorithm is $O(n^2)$. However, in iterated local search there is a user-defined endpoint.

4.2.3 Simulated Annealing

There exists better solution than what we get after 2OPT operations, as it is essentially a greedy heuristic and the search will remain stuck on a nonoptimal configuration, called a local minimum Using Simulated Annealing, we accept worse solutions early in the search. As the search continues, the probability of accepting a bad solution decreases. Hence, we select two random edges and swap them to obtain a solution, even if the solution gives a longer path early on in the search, we accept it and keep on decreasing the probability of accepting a solution after a 2OPT operation if it results in a longer path. Algorithm 2 OPT with Simulated Annealing

Algorithms and Data Structures:.

1. Take initial solution S of greedy method. Set an initial temperature c.
2. Set $S' =$ 2-OPT operation on S.
3. Calculate $\Delta =$ length(S') - length(S).
4. If $\Delta \leq 0$, Set $S = S'$, otherwise set S = S' with probability $e^{\Delta/c}$.
5. If equilibrium reach or $n(n-1)$ iterations completed, reduce c. ($n =$ number of nodes).
6. If $c > 0$, repeat steps 2-5.

Guarantee:.
There is a polynomial bound on the cooling schedule, thus, Simulated Annealing is only an approximation algorithm.

Complexity:.
The number of steps at each temperature is proportional to N^2.So,even if the number of distinct temperatures considered does not grow with N, we will still have an algorithm whose running time is at least $\Theta(N^2)$ with a large constant of proportionality and lesser than $O(n^3)$

4.3 Exact Algorithms - Branch and Bound

Figure 4: SQD Plot for the 2-OPT Algorithm.

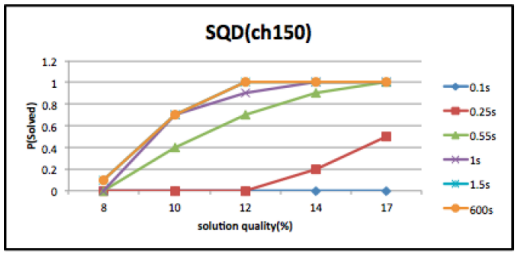
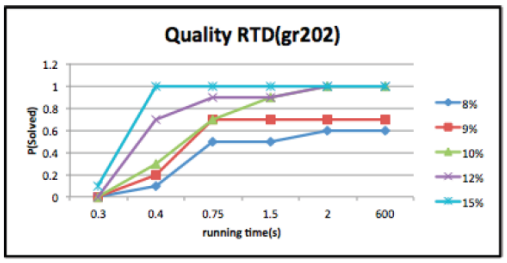
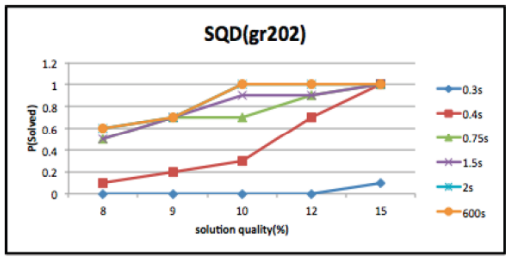
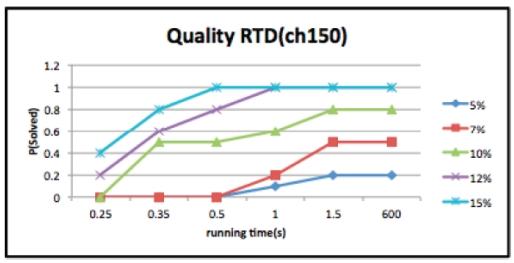


Figure 5: QRTD and SQD Plots for the Simulated Annealing algorithm.







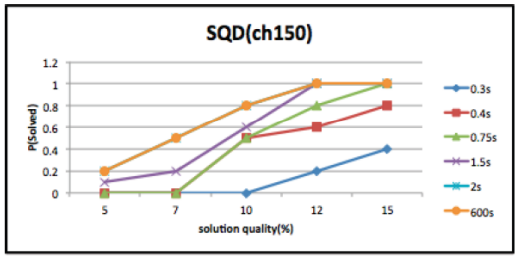


Figure 6: Percentage Error Box Plots for the 2-OPT algorithm.

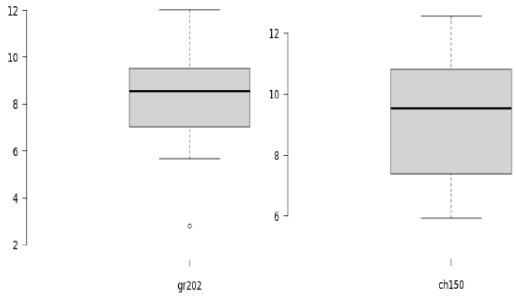
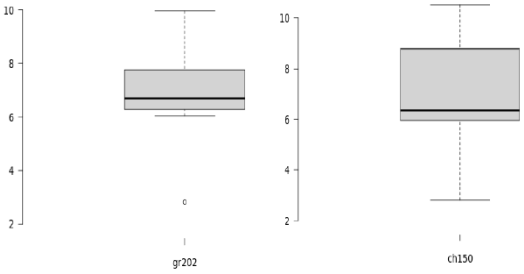


Figure 7: Percentage Error Box Plots for the Simulated Annealing algorithm.



Algorithms and Data Structures:.

1. A n x n matrix C will be used to store the cost of traveling from city i to city j as cij.

2. The "2 shortest edges" bounding function will be used, where the lower bound for a vertex will be the sum of the two shortest outgoing edges divided by 2.

3. The solutions will be iterated by choosing the most promising solution and then in turn expanding that solution.

Guarantee:.

The branch and bound algorithm finds the optimal solution with no error.

Complexity:.

Because the algorithm in essence checks all possible solutions, it may be that its time complexity is the same as the brute-force, which is O($n!$).

5. EMPIRICAL EVALUATION

Platform Details : All programs are written in C++, compiled with $g++v.4.3$ on Mac OSX $v10.9.4$. The system has a RAM of 4 GB and 1.4 Ghz Intel i5 processor.
The experimental procedure involved running the program with the given *.tsp* files, sometimes in multiple iterations so as to get the average result. To study the behavior of different algorithms and heuristics, we plotted QRTD, SQD, and box-plots.
For timings, solutions, and relative error rates for the five

main algorithms, see Table 1. For the results of a sixth algorithm, Iterated Local Search and its comparison to Simulated Annealing, see table 2. Box plots for the Simulated Annealing, see Figure 7.
The lower bound of Approximation Algorithms had a relative error ranging from 0.14 to 0.41. We obtained a relatively lesser error for smaller tsp files with small number of nodes as compared to the bigger ones. The branch and bound algorithm seeks to find the exact solution. Hence, we got the optimum path within the 10 minute cut-off for two of the smallest graphs(*burma16 & ulysses*).

Table 2: Comparison between results of Iterated Local Search and Simulated Anealing.

Dataset	Iterated Local Search			Simulated Annealing		
	Time	Length	RelErr	Time	Length	RelErr
burma14	600	3327	**0.01**	4.89e-2	3347	**0.01**
ulysses16	600	6865	**0.01**	6.00e-2	6789	**0.01**
berlin52	600	8076	**0.07**	1.12e-1	7806	**0.04**
kroA100	600	22046	**0.04**	2.16e-1	22505	**0.06**
ch150	600	7176	**0.10**	2.97e-1	7094	**0.09**
gr202	600	42300	**0.05**	4.17e-1	43501	**0.08**

6. DISCUSSION

In this study several approaches for solving the TSP have been implemented, and experiments were done to compare the approaches in terms of time and error rate.
The construction heuristics were found to be the fastest but also the poorest performing. This is in agreement with stated lower bounds of these algorithms, which produce worst-case error rates of 100 percent or more (see Algorithms section). Their speed may be due to the fact that these algorithms simply return the first solution constructed, rather than comparing different solutions and repeatedly picking the more optimal one. There appeared to be no clear difference between the two construction heuristics implemented for this study.
Exact solutions, on the other hand, theoretically should find the optimal solution, but at a very high time cost. The Branch and Bound algorithm worked well for the smallest dataset we used. It found the optimal solution in under three seconds. However, for the other datasets the Branch and Bound did not finish in under ten minutes. Perhaps the difficulty in implementing Branch and Bound as well as its time cost are tradeoffs for its nonexistent error.
The best compromise between time and error seemed to be the local search approximation algorithms. Three of these algorithms were tested in this study. The 2-OPT represents the more classical variants of these algorithms; it performed favorably compared to the construction heuristics. The simulated annealing performed even better, with error rates below 10 percent and taking less than 1 second to complete. The iterated local search, because it has no clear endpoint, was cutoff at ten minutes and it produced error rates comparable to those of simulated annealing.

7. CONCLUSIONS

We discuss various algorithms and heuristics for solving the TSP problem. We consider the merits and trade-offs of each of the popular technique and present our own evaluation.We hope that our work provides clarity and knowledge towards solving this complex problem.

8. ACKNOWLEDGMENTS

We would like to thank Professor Bistra Dilkina and the TAs of the CSE 6140 course, CSE Algorithms, for Fall 2014.

9. REFERENCES

[1] Traveling salesman problem - insertion algorithms. `http://www2.isye.gatech.edu/~mgoetsch/cali/VEHICLE/TSP/TSP015__.HTM`. Accessed: 2014-12-05.

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2009.

[3] H. Hoos and T. Stutzle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufman, 2004.

[4] D. Johnson and L. McGeoch. The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley and Sons, London, 1997.

[5] J. Little, K. Murty, D. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.

[6] R. Matai, S. Singh, and M. L. Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. In P. D. Davendra, editor, *Traveling Salesman Problem, Theory and Applications*. 2010.