CSE 6140 Algorithms Project- Partial Report

Eisha Nathan
Georgia Tech
enathan3@gatech.edu

Lanssie Ma
Georgia Tech
lanssie.ma@gatech.edu

Vishnupriya
Georgia Tech
priya71189@gmail.com

ABSTRACT

The Traveling Salesman Problem (TSP) is a known NP-Complete problem wherein given a complete graph G=(V,E), the goal is to travel to all the vertices by only visiting each vertex once.

1. INTRODUCTION

In this partial report paper, we provide an update on our progress of five different algorithms: exact, heuristic and approximation, which attempt to give a reasonable estimate for solving the Traveling Salesman Problem (TSP).

2. ALGORITHMS

Five algorithms for determining solutions to the TSP were implemented; the implementation of each and comparison of results are below.

2.1 Branch and Bound

Branch and bound is an exact algorithm that, given enough time, will produce the optimal solution because it will check the entire exponential space of all possible solutions (worst case). This algorithm will build a full solution from scratch by starting with smaller partial solutions and slowly 'expand' a partial solution to include more nodes. At each iteration it 'chooses' the next partial solution to expand by using a combination of the Held-Karp bound and the longest partial solution. This combination ensures we are looking at the most promising partial solution (by looking at the tightest Held-Karp bound) while also ensuring we find *any* feasible solution in a reasonable amount of time since we also choose based on length of current partial solutions.

2.2 MST-Approximation

This approximation algorithm first finds the minimum spanning tree (MST) of the entire graph G and then randomly selects a node as a starting point. To get the approximate tour it then performs a preorder walk on the minimum spanning tree. The pre-order walk we chose was a depth first search. The order in which nodes are visited from the DFS becomes our solution, a TSP solution, where the edge values are taken from the original graph G and summed together. The final edge from the last node visited and our randomly chosen root is also added to the total cost.

2.3 Nearest Neighbors

The nearest-neighbors algorithm employs a greedy heuristic to build an approximate solution from the beginning. Given a partial tour, the algorithm will at any given point pick the closest unvisited neighbor to extend the partial tour. explain the algorithm some more The pseudo-code is as follows:

```
s = random vertex v ∈ V
while not all vertices are visited:
    w = closest unvisited neighbor of v
    extend path from v to w
    v = w
extend path from v to s
```

The algorithm performs a nearest neighbor search starting with each vertex as an initial point and chooses the best tour from there.

2.4 Hill Climbing

Hill Climbing is a local search algorithm that starts with a randomly initialized solution and tries to make the solution 'better' with each iteration by 'climbing the hill' of the plot of the quality of all the feasible solutions in the entire solution space. The algorithm uses a 2-opt exchange at each iteration to look at all the possible variations of a route wherein a 'variation' means a portion of the solution route is reversed.

2.5 Iterated Local Search

This algorithm starts with a randomized initial solution, perturbs the current position, then performs a local search. The current path is altered using the 3-opt exchange.

3. EMPIRICAL EVALUATION

The algorithms are implemented in Python using the NetworkX graph package software. This enables easy creation, manipulation and study of complex networks [1]. It provides methods to create nodes and edges and also provided useful functions for data extraction and manipulation. Some algorithms such as depth first search and finding minimum spanning trees has been implemented in Networkx documentation, and been used in our project. The visualization of the results can also be done using this software.

4. RESULTS

See Table 1 for partial results for Nearest Neighbors and MST-Approx algorithms obtained thus far.

5. FUTURE WORK AND CONTRIBUTIONS

So far we have implemented 4/5 of the algorithms. Their progress' is detailed below. Nearest Neighbors, Hill Climbing, and MST-approx have been completely implemented. The branch and bound algorithm is completed and is in the process of optimization. The Iterated local search algorithm still needs to be implemented. Currently to date, Eisha Nathan has implemented the Branch and Bound, Nearest Neighbors, and Hill Climbing Algorithms. Lanssie Ma has implemented the MST-Approx algorithm. Vishnupriya will implement the Iterated Local Search algorithm. All members will continue writing the report in the near future.

Table 1: Experimental results for all five algorithms on all six datasets. Time is in milliseconds, 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			Nearest Neighbors			MST-Approx			Hill Climbing			Iterated Local Search		
	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr	Time	Length	RelErr
burma14	-	-	-	.39	3841	.16	.40	4009	.21	-	-	-	-	-	-
ulysses16	-	-	-	.61	7943	.16	.48	6564	.043	-	-	-	-	-	-
berlin52	-	-	-	15.65	8181	.084	3.89	9539	.26	-	-	-	-	-	-
kroA100	-	-	-	110.03	24698	.16	14.18	29108	.37	-	-	-	-	-	-
ch150	-	-	-	348.70	7113	.09	36.46	8553	.31	-	-	-	-	-	-
gr202	-	-	-	937.83	47060	.17	61.33	51707	.29	-	-	-	-	-	-