An MST Heuristic Algorithm
to
Approximate Solutions for ATSP's

(Anonymized)

18 April 2017
CS 312-001
Professor Ryan Farrell
BYU, Computer Science

Abstract

The Traveling Salesman Problem is one of the most widely studied computational problems in the fields of mathematics and computer science. In this paper, our group did not attempt to find an efficient algorithm for this problem, but rather we attempted to create an efficient algorithm that was balanced in terms of its speed and accuracy. The algorithm presented in this paper is based on using the minimum spanning tree of the graphical representation of a given TSP to efficiently calculate an approximate route. While the underlying algorithm of our group's solution is quite simple, we built upon it with unique optimizations that allow it to more efficiently solve an ATSP.

1 Introduction

The Traveling Salesman Problem has long intrigued scholars and researchers due to the simple nature of the problem coupled with the inability to create an efficient algorithm to solve it [1, p.2]. The purpose of this report is to discuss the algorithm, created by the authors of this report, that efficiently approximates solutions for ATSP, as well as discuss a more general greedy approach for ATSP that was used as a baseline for our unique algorithm. In-depth analysis of the principles and techniques utilized in the two algorithms will be discussed, along with their respective time and space complexities. Data comparing our algorithm with the greedy approach, a branch-and-bound approach, and a random approach will be documented within this report as well. In order to fully discuss the techniques used in our MST algorithm, this report will first discuss the characteristics of an ATSP versus a normal TSP.

2 ATSP

An ATSP, or an Asymmetric Traveling Salesman Problem, is distinguished from normal TSP by its asymmetric edge weights between cities. This means that the cost to travel from city A to city B is not necessarily equal to the cost to travel from city B to city A. In the problems generated for the testing of our algorithms, directed graphs were created with certain edge weights being set to positive infinity. This means that it is quite possible to travel from one city to another, but not being able to travel in the other direction. This also means that ATSPs do not follow the triangle inequality that is traditionally used in TSP's; hence the term, ATSP. How ATSPs affect the structure of approximation algorithms will be discussed later in this report.

3 A Greedy Approach to ATSP

3.1 Implementation

Our implementation of the classic greedy algorithm is based on the greedy approach of making the next decision based what offers the most obvious and immediate benefit [2, p.127]. For a TSP, this involves starting from a city within the graph and picking the next city to move to based upon choosing the move that would incur the minimal cost. This is repeated until either we have visited every city and are able to return to the start city, or until the we reach a point where moving to any city would incur a cost of infinity. In order allow this algorithm to traverse an ATSP, this algorithm repeats itself so that each city in the problem acts as the starting city, and the least cost solution from those iterations is chosen to be the best greedy approximation. This guarantees a solution where one is possible, even in incomplete graphs.

3.2 Complexity Analysis

The crux of of this algorithm's complexity comes down to the fact that for each city in the problem, it must compare the costs to travel from that city to any other city, which creates a simple double loop in the code. This contributes a time complexity of $O(n^2)$, where n is the number of cities in the problem. The addition of doing those calculations n times, so that each city may serve as a start city, creates a triple nested loop, increasing the complexity of this algorithm to $O(n^3)$. All other functions used in our greedy algorithm that could contribute to complexity, such as calculating the route's cost and calculating costs between cities, are dominated by the triple nested loop. Thus, the overall time complexity of the algorithm is $O(n^3)$. Space-wise, the only space that is impacted by the size of the problem is the route that is generated as the solution, which gives this algorithm an overall space complexity of O(n).

3.3 Why Greedy as a Baseline?

Within the realm of approximation algorithms, the simple greedy approach that we have implemented is the most straightforward way of calculating an approximate solution to ATSP, which allows it to serve as a good baseline to other, more complex, approximation algorithms, such as the one presented in this paper.

4 An MST Approach

4.1 Why MST?

It has been shown by other researchers that using an MST heuristic for approximating solutions to symmetric TSP's has an approximation ratio of 2 [3]. This ratio is much better than the approximation ratio of O(log(n)) for the simple Greedy algorithm. Our team wanted to explore how the MST approximation ratio would change, and how beneficial an MST heuristic would be, when evaluating for asymmetric TSP's.

4.2 Implementing an MST Heuristic

Our implementation of an MST heuristic involves four main steps. First, every edge in the graph is placed in a priority queue, giving priority to the lowest-costing edge. Second, we use Prim's algorithm to create a minimum spanning tree. This is done by arbitrarily picking a start node and using the priority queue to repeatedly add the current lowest cost edge connected to the tree, one edge at a time, until all vertices are included. Third, this minimum spanning tree is transformed into an eulerian tour by doubling each of the edges. Fourth, the eulerian tour is transformed into an ATSP solution by traversing its edges, replacing each edge that would visit an already visited node with an edge to the next node along the eulerian tour that hasn't yet been visited. This results in an ATSP solution because each node will be visited only once, and it will end at the start node because that is what the eulerian tour already does. To ensure a solution is found in incomplete graphs, our algorithm repeats steps 2 through 4 with each city as a starting node for the MST (up to 50 cities) and returns the best solution found.

4.3 Complexity Analysis

The overall time complexity of this algorithm is O(n^3) for the worst case scenario, but most of the time it will actually be an average of O(n^2 log(n)). This complexity is mainly due to the way we stored edges on the queue, and how Prim's Algorithm is done. All edges are initially added to the queue once at the beginning, by traversing the distance matrix of the graph and adding each edge to an ordered set. The traversal of the distance matrix is n^2, and each addition of an edge to the ordered set is log(n) because the set is a tree structure, thus initializing the queue is a total of O(n^2 log(n)). Then, the MST is formed by executing prim's algorithm. This algorithm starts with a single node, locates the smallest cost edge that connects to that node, and then repeats this for each new node until all nodes are reached. Even though the queue, which is an ordered set of edges, stores the smallest cost edge at the head, we must still traverse through the queue until we find the first edge that actually connects to the current node. Potentially, the worst case scenario would be for a complete traversal of the queue to find the necessary edge. This queue is of size n^2 (because it was created from the distance matrix), and so in the worst case (yet highly unlikely) scenario the entire queue is traversed for each new node added to the MST, resulting in a O(n^3) time complexity. However, from our own testing, we found that in practice a retrieval of the appropriate edge from the queue is more or less a O(n) operation, and so the average complexity for prim's algorithm here is O(n^2), which does not overpower the O(n^2 log(n)) complexity from building the queue. Because we are running this algorithm on potentially incomplete graphs, prim's algorithm is actually done several times with different starting nodes to ensure a solution is found, but this simply adds a constant factor compared to large n. As for the rest of our algorithm, both creating the eulerian tour and traversing it to shortcut the edges are O(n) and therefore do not affect the overall complexity of the algorithm. Thus, the total time complexity for our MST heuristic is an average of O(n^2 log(n)), with a worst case scenario of O(n^3).

Space complexity is more simple. Initializing the queue is of course O(n^2) because of the use of the distance matrix. After than, nothing is storing or accessing any structure more complex than the solution route, which is size n at its completion. Thus, the total space complexity of the algorithm is O(n^2) due to the initializing of the queue.

4.4 Pros and Cons

The main strength of this algorithm is its potential to find a solution close to the optimal. Starting with the minimum spanning tree of the graph gives the ATSP solution a great head start compared to the greedy algorithm. This algorithm should also be faster because our implementation of prim's algorithm will generally be much closer to n^2 than n^3 (as discussed above), and the greedy algorithm will *always* be closer to n^3.

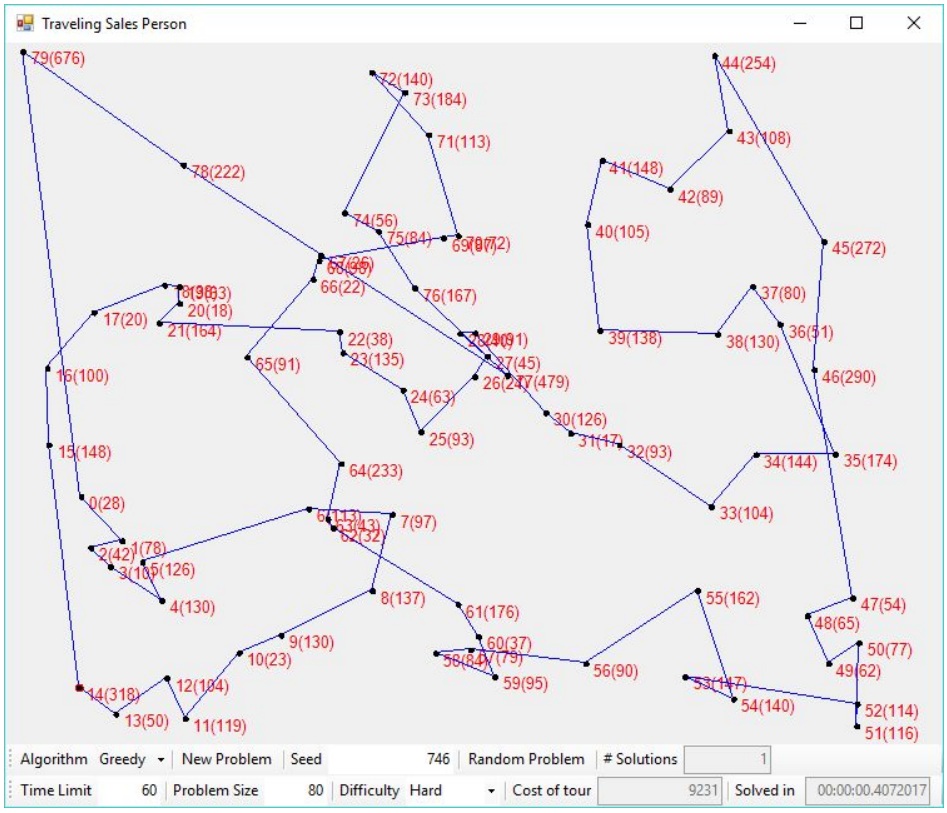
The cons of this algorithm lie in the shortcutting of the eulerian tour. In order to transform the eulerian tour into an ATSP solution, we must remove the edges that lead to already-visited nodes, while still

covering each node once. The general method used for shortcutting (which is what we implemented) involves traversing the eulerian tour's edges, keeping track of nodes that have been visited, and repeatedly adding the edge to the first child node that hasn't yet been visited. For a TSP problem that satisfies the triangle inequality, shortcutting is very effective because it replaces a path of several edges between nodes with a single edge, which is always shorter by the definition of triangle inequality. However, the nature of an ASTP problem removes this guarantee. Therefore, this shortcutting method becomes inefficient, and results in polluting the minimal nature of the MST that we started with. This effect can be seen in the following data.

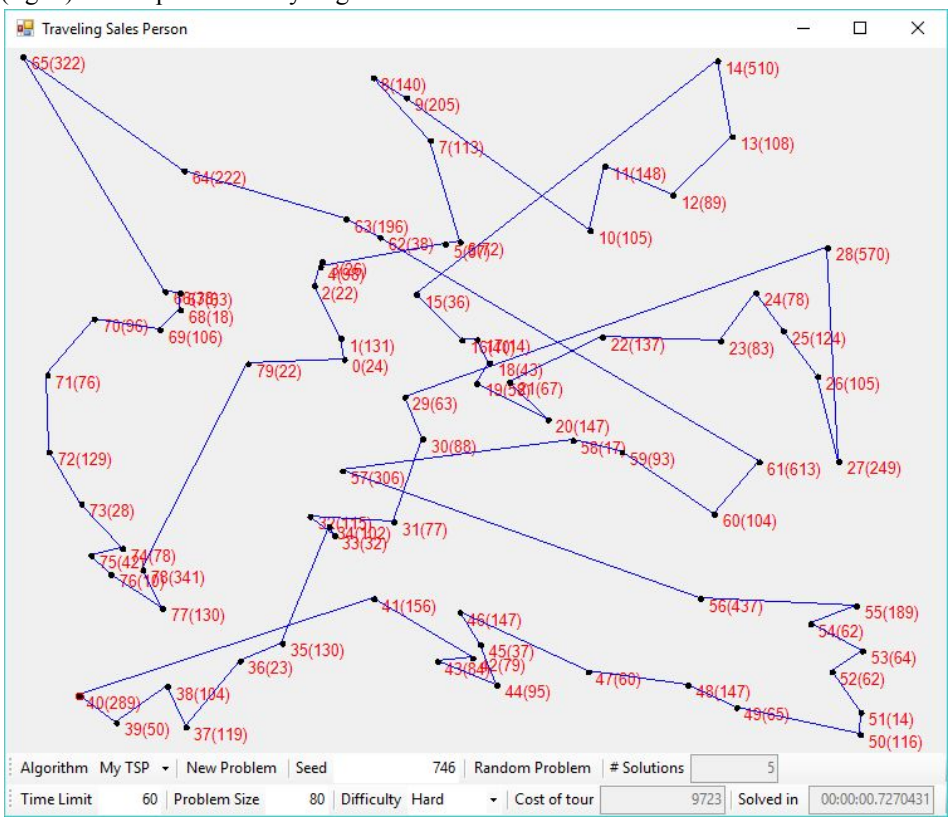
5 Empirical Performance

	Random	Greedy			Branch and Bound			MST		
# of Cities	Path Length	Time (sec)	Path Length	% Improve	Time (sec)	Path Length	% Improve	Time (sec)	Path Length	% Improve
15	8224.6	0.0019038 6	4074	49.96%	0.2414704 2	3690. 8	8.91%	0.0032603 4	3954	2.80%
30	16602.4	0.0142128 8	6015	63.78%	TB	TB	TB	0.0174667	5751	4.32%
60	31103.8	0.1136	7896	74.50%	TB	TB	TB	0.184	8532	-8.13%
100	52558.8	0.6074	10985	79.09%	TB	TB	TB	0.797	11397	-3.95%
200	106137. 2	6.5626	15917	84.97%	TB	TB	TB	6.1256	16989	-6.76%
400	206617. 4	80.6384	24578	88.10%	TB	TB	TB	58.4564	27378	-11.39 %
800	414606	600	36735	91.14%	TB	TB	TB	600	42058	-14.49 %
5000	2602290	600	111712. 5	95.71%	TB	TB	TB	TB	TB	TB

(fig. 1) *Table of empiric data collected to show performance of our algorithm compared against a branch-and bound algorithm, a random algorithm, and a greedy algorithm. % Improve for greedy is its improvement of accuracy over the random algorithm. % improve for Branch-and-Bound and MST are their improvements in accuracy over the greedy algorithm.



(fig. 2) *Example of Greedy Algorithm



(fig. 3) *Example of our MST Algorithm

5.1 Data Analysis

Initially for smaller n (30 or less) the MST heuristic finds improved solutions compared to the greedy algorithm. This is because with smaller n, there will be less shortcutting necessary, and the final solution will be closer to the actual minimum spanning tree. But as we can see from the data, our MST heuristic algorithm continually worsens in comparison to the greedy algorithm's solutions. We believe this to be a direct result of the effects of shortcutting the eulerian tour in an ATSP graph. As n grows larger, the number of MST edges that need to be swapped for "shortcut" edges grows as well. Because the graph is asymmetric and does not satisfy triangle inequality, these replacement "shortcut" edges are not necessarily shorter than the MST subpath they are replacing. Thus, a larger amount of shortcutting results in distancing the solution's cost from the MST cost by an increasing amount. This effect proves large enough to actually cause the MST heuristic algorithm to find increasingly worse solutions compared with the greedy algorithm, even though starting with the MST is a clear head start for reaching an optimal solution.

Nevertheless, the choice to compute the shortcutting this way was intentional on our part, for the sake of reduced time complexity. The shortcutting method we implemented is fast: it simply picks the first edge found that connects to the next unvisited node along the eulerian tour, resulting in a traversal no more than 2n. There may be ways that the shortcutting could be done more efficiently, perhaps by inspecting a set of possible local shortcut edges and choosing the smallest cost edge each time, but this would likely requires data structures or loops that would add to the algorithm's time complexity. An examination of the data shows that as n grows large (200 cities and above), the MST algorithm proves to indeed be faster. This was predicted by our observation that the average MST complexity is O(n^2 log(n)), versus the greedy's O(n^3). The reason that the greedy algorithm is quicker for smaller n is simply because the MST's extra operations (such as building the n^2 queue and traversing it repeatedly during Prim's algorithm) overpower its advantages for a small number of cities. Clearly, the choice to prioritize time allows both the greedy *and* MST algorithms to compute solutions with much higher n than the Branch and Bound algorithm. Although B&B finds a significantly improved total path cost, it can only do so for the smallest number of cities in reasonable time. All of this shows that there is a constant trade off between time and accuracy when computing approximate solutions to ATSPs.

6 Conclusion

Although an MST heuristic can result in TSP solutions with an approximation ratio of 2, we have found that applying such an algorithm to an ATSP brings significant complications. The heart of these complications lie in how the transformation from eulerian tour to ATSP solution is implemented. Future improvements in accuracy can likely be made by focusing on a shortcutting method that selects an ideal edge without significantly adding to the time complexity. The speed of the algorithm may also be improved by experimenting with different types of queue implementations. Because this queue stores every single edge of the graph, which need to be accessed quite frequently, perhaps a structure that can add or access edges in constant time would make a significant difference. Finally, we believe a combination of the MST heuristic with other TSP algorithms, such as 2-opt, are worth looking into. Combinations could potentially combine the great accuracy advantage of starting with an MST with other advantages that perform better than shortcutting the eulerian tour.

References

[1] D. Applegate, R. Bixby, V. Chvatal and W. Cook, *The Traveling Salesman Problem*. Princeton: Princeton University Press, 2011.

[2] S. Dasgupta, C. Papadimitriou and U. Vazirani, *Algorithms*. Boston: McGraw-Hill, 2008.

[3] V. Deineko, A. Tiskin, *Minimum-weight tree shortcutting for Metric TSP*. University of Warwick.