Traveling Salesmen Short Circuit Algorithm

Stanislaw Antol, Raphael Cherney, Daniel Grieneisen, Garrett Rodrigues

Olin College of Engineering, Needham, MA 02492 USA

This paper examines the efficiency of two methods at solving traveling salesmen problems (TSP). The first method takes a brute force approach such that all paths are examined. In order to make this computationally viable, we limit our study to ten cities or less. We then compare the solution found with this method to our second method, called the Short Circuit Algorithm. This algorithm makes a trade-off between the length of the path and the time needed to solve the problem, claiming only to find a path that is no longer than twice the length of the shortest solution rather than the shortest solution. For a specific scenario, our results demonstrated that this algorithm found a path that is 1.22 times longer than the shortest path, while taking 3.56% of the time taken by the brute force method.

I. Introduction

The traveling Salesman problem (TSP) is a shortest-path optimization problem that can be represented using simple graphs. In a traditional TSP problem, we present a salesman with a list of cities including a starting city, and the objective is to find the shortest path such that the salesman visits each of the given cities once and returns to the starting city. Translated into graph theory, we can represent each of the cities as a vertex in a graph, with each of the vertices being connected by edges weighted by the distance between the respective cities. Then, finding the shortest trip for the salesman becomes the equivalent of finding a Hamilton circuit with minimum total weight in the graph.

We decided to approach the Traveling Salesman Problem (TSP) by looking at a specific example and implementing two different algorithms to solve it.

II. Travel the World

A. The Problem

Garrett owns a 1973 Learjet 24D airplane and has decided to take Stan, Dan, and Raphael on a tour of the world. On the way, they plan on leaving from Boston, MA, visiting Garrett, Stan, Dan, and Raphael's hometowns (San Francisco, CA; Chicago, IL; Carlisle, PA; and Omaha, NE), the cities that they plan on visiting in the next year (Tokyo, Japan; Aberdeen, Scotland; and Geneva, Switzerland). Finally, because they are huge amateur sports fans, they want to visit the old 2000 Olympic stadium in Sydney, Australia and the new 2012 Olympic stadium in London, England. Because of the bad economy, gas prices are high, and aircraft fuel is even more expensive. Therefore, Garrett wants to find the shortest distance trip that will allow them to visit all of the cities and end back in Boston. Additionally, Garrett made the decision today that he wants to do this and is planning on leaving within an hour, so he needs to find the best circuit as quickly as possible.

Manuscript received October 22, 2009.

III. Brute Force

A. Introduction

The first method which we tried to solve the TSP on our graph is the brute force method. Brute force is extremely simple. First, every possible Hamiltonian circuit in a graph is found. Next, a computer loops through all of these possibilities and calculates the length of each circuit. This method is guaranteed to find the shortest length Hamiltonian circuit because it checks every single one.

B. Problems with Brute Force

There is a major problem with the brute force method of solving a TSP, especially on a complete graph. On a complete graph with n vertices, there are (n-1)!/2 possible Hamiltonian circuits. For a 5 node graph, there are only 12 possible cases. This increases very quickly with the number of nodes. With a 10 node graph, there are 181440 cases that must be considered, and with 20 nodes, there are 1.2e18 cases that must be checked. Taking a brute forcing approach to finding the solution rapidly becomes too computationally expensive to finish in a reasonable amount of time. If Garrett wanted to fly the rest of his Discrete project team to 100 or 1000 cities instead of just 10, this method would quickly become too computationally intensive for him to figure out the best path and still maintain his timeline. Because of this, a number of approximate methods have been developed. These methods operate much more quickly than brute force methods on large graphs, and while they do satisfy some performance metric, they do not guarantee that the shortest Hamilton circuit will be found.

IV. Short Circuit

A. Introduction

We decided to implement the Short Circuit algorithm to see whether it could perform faster than the brute force method. The Short Circuit algorithm is guaranteed to find a circuit no longer than twice the length of the optimal circuit on a complete graph which obeys the triangle inequality [1].

1) Triangle Inequality

The triangle inequality for a graph states that for any set of three mutually connected vertices a, b, and c, the weight of any single edge is less than or equal to the sum of the weights of the other two edges, e.g. ab <= ac+cb. This is based off of the geometric properties of triangles. Specifically, the triangle inequality holds for a simple edge-weighted graph where the vertices correspond to geographic locations and the weight of an edge corresponds to the distance between the two cities which the edge connects. This is how the graph of our problem is constructed, so the triangle inequality holds.

2) Algorithm

Figure 1 illustrates the Short Circuit algorithm. Each step is further explained in the following sections.

B. Minimum Spanning Tree

1) Prim's Algorithm

The first step of the Short Circuit Algorithm is to find the minimum weight spanning tree of G that has n vertices. Prim's Algorithm is a method for finding the minimum weight spanning tree of a simple weighted graph. Prim's Algorithm works as follows: First, choose an edge in the graph G and add it to the tree. Select an edge of minimal weight that is incident with a vertex already in the tree that does not create a simple circuit with vertices already in tree. Repeat. Stop when all vertices have been selected (n-1 edges have been selected).

2) Proof of Prim's Algorithm

Prim's Algorithm is guaranteed to find the minimum weight spanning tree of a graph. The proof of this is available in [2].

C. Eulerian Circuit

1) Recursive Circuit Algorithm

The next step in the Short Circuit Algorithm is to double every edge in the minimum spanning tree T to create the new graph T_2. By doubling each edge, the degree of every vertex is also doubled. This implies that the degree of every vertex is even. Therefore, an Eulerian circuit, EC, exists in T_2. We used a recursive algorithm, which is best illustrated in pseudo-code, [4] to find this Eulerian circuit:

```
Pseudo Code:
find_tour(u):
        for each edge e=(u,v) in E:
                remove e from E
                find_tour(v)
        prepend u to tour

to find the tour, clear stack 'tour' and
call find_tour(u), where u is any vertex
with a non-zero degree.
```

2) Proof of Correctness of Recursive Algorithm

The complete proof that this algorithm is sufficient to find an Eulerian circuit is available at [3]. Briefly, if a graph G has an Eulerian circuit, then the degree of every vertex is even. If the edges from a circuit C (not necessarily Eulerian or Hamiltonian) are removed from G, then the degree of every vertex is still even (or zero), because there must be an equal number of edges incident to a given vertex in a cycle. Because the degree of every vertex is even, then Eulerian circuits exist in each connected component of G-C. Now a circuit can be found in each of the connected components and the procedure is repeated until all of the edges are used. This recursively finds an Eulerian circuit.

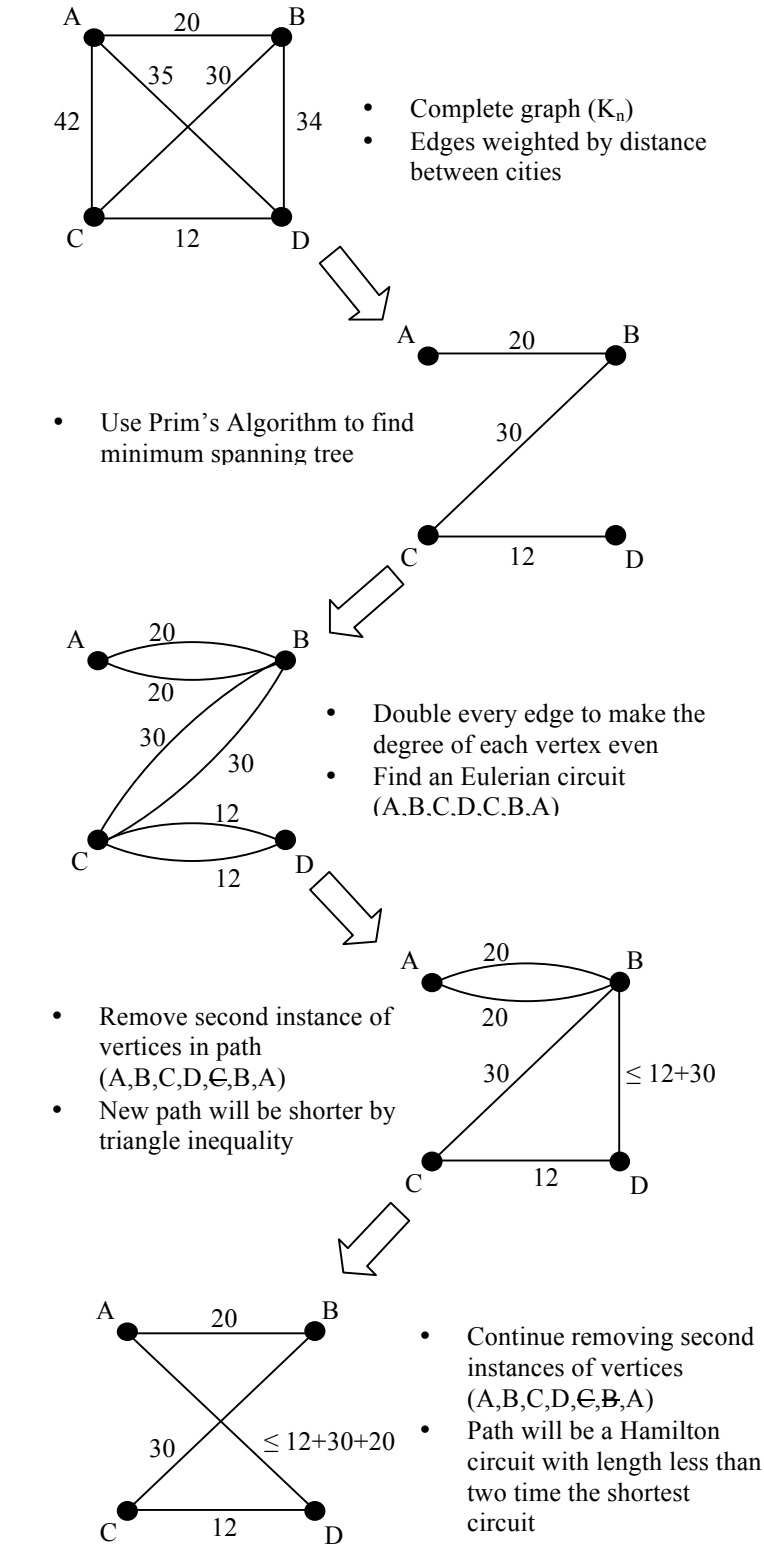

Fig. 1. Short Circuit Algorithm. This shows the steps in the Short Circuit Algorithm.

D. Short Circuit

1) Short Circuit Algorithm

Now that we have the Eulerian circuit EC, we can construct a Hamiltonian circuit C. V is the list of vertices in EC in the order that they appear in the circuit. Go through V and find any instance that a vertex is listed more than once. Eliminate the second instance of the vertex. This effectively shortens the circuit. For example, if V is (a,b,c,b,d,…), then 'b' is listed twice. The circuit follows the edges ab,bc,cb,bd,…. When the second 'b' is eliminated, V becomes (a,b,c,d,…) and follows the edges ab,bc,cd,… This effectively removes edges cb and bd and replaces them with edge cd. Because G is a complete graph, edge cd exists, and so it can be used in C. Since the triangle inequality holds for G, cb+bd ≥ cd, so replacing the two edges with a single edge will either shorten the circuit or keep it the same length. Once every vertex is listed exactly once in V, Hamiltonian cycle C is created by following the vertices in V in order.

2) Proof of Maximum Bound

The maximum bound on the Short Circuit Algorithm is 2k, where k is the length of H, the shortest Hamiltonian circuit in graph G. To prove this, start with H. Now, create spanning tree P by removing any single edge from H. The sum of the edges of P, l(P), is less than k, the sum of the edges of H. P is a spanning tree, so at a minimum, its length is equal to that of T, the minimum spanning tree G. The sum of the edges of T, l(T), is less than or equal to the sum of the edges of P. This implies that the sum of the edges of T is less than k, i.e.

$$l(T) \leq l(P) \leq k.$$

Finally, the length of C, the circuit created by the Short Circuit Algorithm is less than twice the length of T. This depends on the fact that T obeys the triangle inequality. Because of this, whenever two edges in T_2 (the minimum spanning tree, T, but with every edge repeated twice) are replaced by a single edge, the length of the resulting circuit is shorter. This process is applied multiple times to T_2 to create C. Therefore,

$$l(c) \leq 2(l(T)).$$

Thus,

$$l(c) \leq 2(l(T)) \leq 2(l(P)) \leq 2k.$$

V. Test and Results

A. Test

In order to compare these two algorithms, we implemented both of the algorithms in MATLAB. A print out of the code is included in the attached supplementary material. By implementing these algorithms, we can look at two different aspects of the problem.

The first is the resultant paths, including the distance of the paths. The resultant paths for the Discrete Team traveling problem from the two algorithms are shown in Fig. 2 with path lengths of 25,060 miles and 30,459 miles, for the actual shortest Hamilton circuit and the approximate Short Circuit path. Thus, the shortest circuit algorithm's path is 1.22 times larger than the brute force algorithm. The circuit found using the brute force method is: Boston, Aberdeen, London, Geneva, Tokyo, Sidney, San Francisco, Omaha, Chicago, Carlisle, Boston. The circuit found using the Short Circuit Algorithm is: Boston, Aberdeen, London, Geneva, Carlisle, Chicago, Omaha, San Francisco, Tokyo, Sidney, Boston. These results confirm our theorem about the Short Circuit Algorithm. The minimum spanning tree of our graph is 16,706 miles, giving us an upper bound of 33,412 miles, which is larger than both calculated paths.

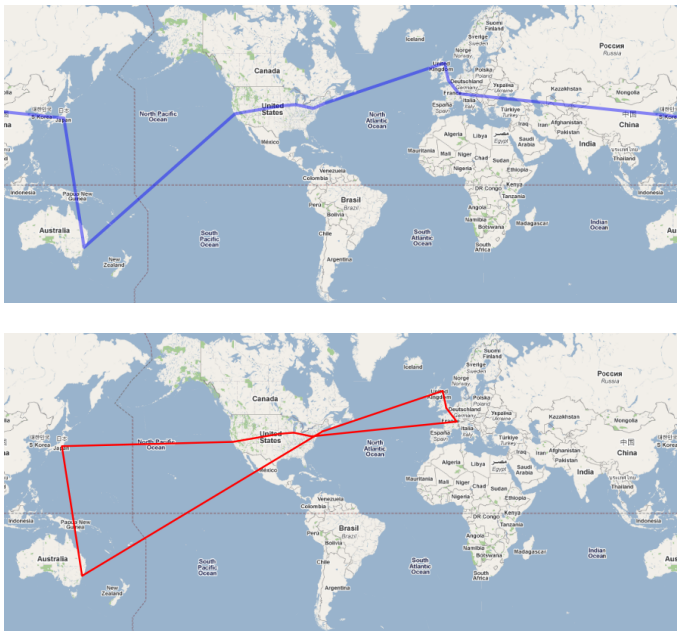

Fig. 2. TSP results. The top image shows the result of the brute force algorithm. The bottom image shows the result of the Short Circuit Algorithm.

The second is the computation time needed to run these programs. We wanted to observe the behavior of the programs for a varying different number of cities. We generated the data for these runs by removing a city from the original tour until we had the minimum size of 3. We then ran both of our programs for the tour sizes from 3 to 10. To minimize the effect of noise (e.g. other programs running on the computer) on the data, we ran each program for each tour size 100 times and averaged the data. The results of this experiment are seen in Fig. 3. The results show a trend of the runtime of the brute force algorithm becoming much longer than the Short Circuit Algorithm for graphs larger than 9. Although we would have liked to increase the tour size beyond 10, our implementation in MATLAB would run out of memory above 10. This was primarily because we used a built-in function to get a list of all of the permutations. If we only looked at one permutation at a time, the program would be much more scalable. Looking at the specific case of touring all 10 cities, the Short Circuit Algorithm finished in 3.56% the time of the brute force method.

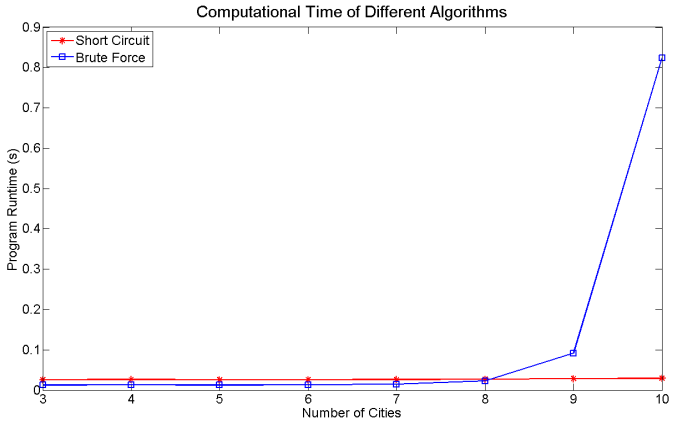

Fig. 3. TSP computation time results. This figure shows the difference in runtime for the two algorithms as a function of number of cities.

VI. CONCLUSION

The Traveling Salesman Problem is applicable in a wide variety of fields. In this paper, we explored an algorithm that approximates a solution to the Traveling Salesman Problem. The Short Circuit Algorithm is only valid for complete graphs, which limits its applicability, but, as we showed in this paper, can provide very significant runtime improvements if one does not necessarily need the shortest possible path. However, sometimes the shortest possible path is needed for a given application. This need drives researchers to continue discovering more efficient methods of solving the TSP than the method we implemented.

REFERENCES

[1] Arthur M. Hobbs. *Applications of Discrete Mathematics (Chapter 15)*. New York, NY: McGraw-Hill, 2007.
[2] Kenneth H. Rosen. *Discrete Mathematics and Its Applications,* 2nd ed. New York, NY: McGraw-Hill, 2002.
[3] Raahid bin Muhammed. http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/eulerTour.htm. Oct 22, 2009.
[4] http://www.algorithmist.com/index.php/Main_Page. Oct. 22, 2009.
[5]
[6] Shu Lin, Daniel J. Costello Jr. *Error Control Coding,* 2nd ed. Saddle River, NJ: Pearson Education, Inc, 2004.