**The Traveling Salesman Problem: How The Ants Solve It**

**Gregory S, Justin S, Cory S**

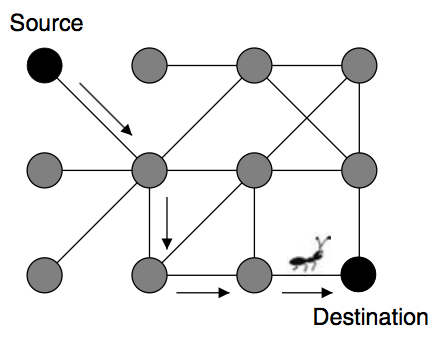
Computer Science Department

Brigham Young University

3361 TMCB, Provo, UT 84602

**Abstract**

The dilemma faced by the traveling salesman, wherein he has a finite number of cities to visit and a desire to minimize the total distance traveled by visiting each one once without backtracking, is a popular mathematics problem in combinatorial optimization. Our goal is to implement an Ant Colony Optimization (ACO) algorithm so as to improve upon a simple brute force solution of selecting the nearest neighbor. We discovered that despite ACO taking longer than Greedy to identify optimal paths, the actual time cost as a function of problem size increases at roughly the same rate, and ACO finds a more optimal path than Greedy does.



**Figure 1.** Ants build a path from a source to a destination node (Dorigo & Stützle, 2004).

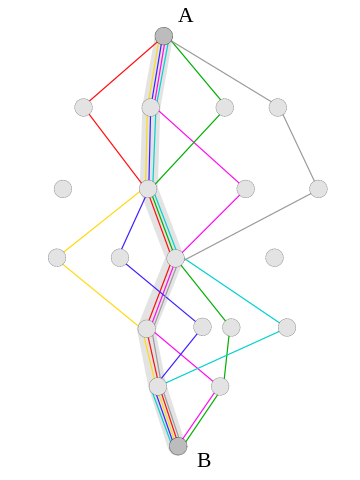
**Introduction**

Though the origins are unclear, the Traveling Salesman Problem, or TSP, was mathematically formulated by several mathematicians in the 1800s and later studied in depth in the 1930s. Practical applications range from planning/logistics to DNA sequencing to aircraft scheduling (Xu, 2017). It is the most well-known combinatorial optimization problem. It is well-studied, and its existence has even affected pop culture (Lanzone & Lanzone, 2012).

There are many algorithm optimization models used to solve the TSP, many of which are better than a simple greedy approach. There is also a subset derived from the observation of ant behavior. To see how well the Ant Colony Optimization Algorithm, or “ant algorithm,” performed, we implemented both and compared the results with (in addition to a branch and bound approach from previous work.

These ant algorithms mimic the behavior of ants by coordinating artificial agents to solve computational problems. TSP problems can be represented as a graph with paths connecting some or all of the nodes. Each ant builds a path from a source to a destination node by following potential paths between each node (see Figure 1). The ant colony is a distributed system wherein the whole colony is capable of solving complex tasks that are impossible for a single ant to solve.

Ants have limited vision, and some species are completely blind. Their primary means of communication is through stigmergy, which is “a form of indirect communication mediated by modifications to the environment” (Dorigo & Stützle, 2004). When searching for food, ants achieve this type of communication by leaving a trail of special chemical signals. Other ants detect these chemical signals, called pheromones, and can follow the path to a food source. These pheromones slowly break down; consequently, there is a direct correlation between the strength of the pheromone trail and the frequency of travel. In other words, less-traveled paths disappear and optimal paths persist. In the absence of any pheromone trails, ants tend to wander randomly in search of food. But even with trails, ants will still deviate on occasion, and this may result in a more optimal path.



**Figure 2.**The shortest path in a graph emerges from the combination of many paths (Dréo, 2006).

**Algorithm Explanation**

We first implemented the greedy algorithm to use as a comparison against ACO. Simply put, all of the edges are calculated and sorted in an array. We then start with the shortest and find a path.

We chose ACO because it is flexible with many applications and there is a lot of room for future work. It is also more complicated than a greedy approach. As with other implementations of the ACO algorithm, individual ants are programmed to simulate the random choices made as each “ant” travels from one node of a graph to another. In the context of solving a TSP, once an ant has identified a (semi-)optimal path, this path is marked with pheromones by altering the semi-random decision tree used by each ant when determining the next node. Subsequent iterations of the algorithm cause improvements as future ants avoid less optimal paths and identify more optimal ones. (See Figure 2.)

An issue that must be solved for any implementation of Ant Colony Optimization is how much weight should be given to distance or pheromones when an ant is deciding which path to take. The decision is probabilistic and weighted according to the shortness of the distance to other nodes as well as the amount of pheromone to other nodes. In our algorithm, we found that at first, the pheromone scent was too strong and after one ant found a solution, then all other ants followed that same path, even if it wasn’t a very optimal path. Once we added more weight to distance, we started seeing more optimal results. Adding more weight to distance and discounting the weight of pheromones allowed ants to better explore new variants of paths because they were incentivized to explore new nodes that were a shorter distance away (Dorigo, Maniezzo, & Colorni, 1996).

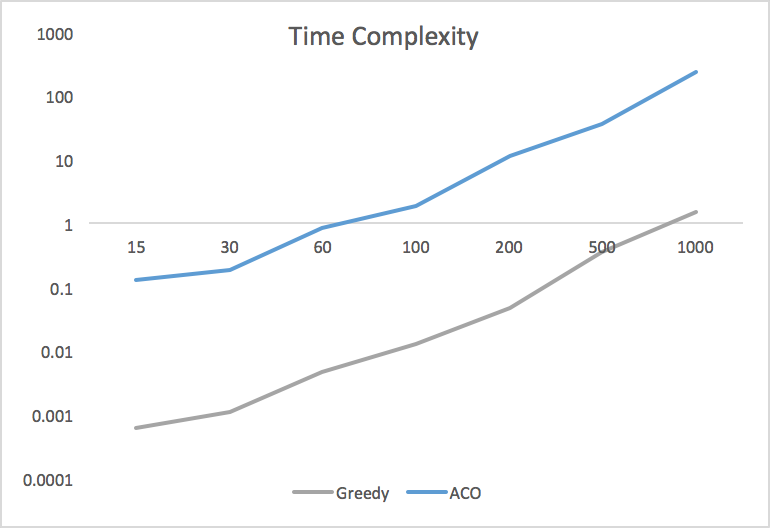
Another parameter we needed to adjust in our algorithm was how many ants we had exploring concurrently. We experimented with a variety of quantities (5, 10, 20, and the number of cities in the problem), but we found that the optimal number of ants was only one. By sending only one ant at a time, the algorithm could find solutions faster and could better compare solutions and direct ants to more optimal routes.

**Complexity**

The space complexity for the greedy algorithm is where *n* is the number of cities. We have to store an array of edges (size ) and several arrays of cities (size ). Time complexity is because we have to iterate over the array of edges and each city.

As for our ACO algorithm, for each city, we stored an array that contained the distances to every other city, an array with the amount of pheromone to every other city, and the probabilities of visiting every other city. For every ant, we stored the path of cities visited and the cost of that path. So our space complexity was , where is the number of nodes and is the number of ants sent in a batch.

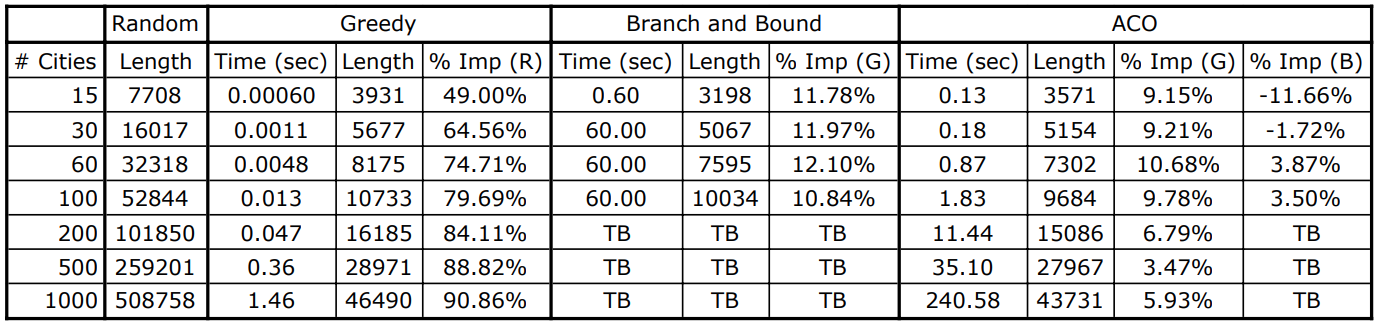
Our algorithm, generally explained, consists of the following: an ant must visit every city, and at every city, the probability must be calculated for every other city. This lends to a time complexity , again where is the number of nodes/cities and is the number of ants sent concurrently. We repeat this times, where is the number of iterations until convergence. Overall, the time complexity of our algorithm is .



**Figure 3.** Time cost plotted against problem size on a logarithmic scale to normalize range.

**Analysis of Results**

As expected, the performance of our version, when compared with the Branch and Bound algorithm, was less optimal with smaller node quantities. However, as soon as the problem size reached 60 or above, our algorithm performed better than Branch and Bound and solved the problem in far less time. Aside from timing out on , the longest our algorithm ever took to solve a problem was 35 seconds (). (See Table 1.)



**Table 1.** Solutions generated by four algorithms: optimal path formed by (1) randomly selecting nodes (2) greedily selecting nodes (3) branch and bound and (4) ant colony optimization. Greedy improvement over Random establishes a reasonable baseline. We then compare B&B with Greedy to determine improvement over a simple brute-force model (i.e. Greedy). Finally, we compare performance of ACO with both Greedy and B&B.

In our test scenarios, we found that the ACO algorithm consistently generated better solutions than the Greedy algorithm, though as the problem size increased, the time appeared to increase more rapidly than Greedy. At first glance, the better score may not be worth the extra time: consider tests at , where there is not even a 4% improvement in total cost, but nearly 100% increase in time taken to calculate. That said, plotting Greedy time and ACO time against problem size on a logarithmic scale demonstrates that, in fact, the time cost increases at roughly the same rate for both. (See Figure 3.)

**Future Work**

The hardware we used for testing was unable to handle problems where . Further work can be done to improve testing hardware to allow better comparison with other algorithms, as well as refactor the code to allow multi-threading ants. The code is partially architected to allow multi-threading; we were unable to complete this functionality due to resource constraints. There is also additional work to be done in order to identify solutions with an evolving topology. This would more closely simulate the changing environment faced by real ants when seeking new sources of food as others become depleted.

# References

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization.* Cambridge, MA, USA: MIT Press.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics, 26*(1), 29–41.

Dréo, J. (2006, 05 29). *Ant colony optimization algorithms.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Ant\_colony\_optimization\_algorithms#/media/File:Aco\_shortpath.svg

Lanzone, A., Lanzone, T. (Writers), & Lanzone, T. (Director). (2012). *Travelling Salesman* [Motion Picture]. USA.

Weise, T., Jiang, Y., Lässig, J., Chiong, R., & Athau, R. (2014). Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the TSP. *2014 IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS)*, (pp. 148–155). Orlando.

Xu, B. (2017, June). An efficient Ant Colony algorithm based on wake-vortex modeling method for aircraft scheduling problem. *Journal of Computational and Applied Mathematics, 317*, 157–170.