Ant Colony Optimization: Applications to the Traveling Salesman Problem

Erin Masatsugu, Billy Cox

November 28, 2016

1 Introduction

The goal of our project was to construct a polynomial time algorithm to find good approximate solutions to instances of the Traveling Salesman Problem (TSP). The Traveling Salesman Problem poses the following question: given a graph of n nodes, what is the shortest (least costly) tour through the graph? A tour is defined as a path which starts from some point s, visits each node in the graph exactly once, and then returns back to s at the end. This is an NP-Hard problem.

We chose to use Ant Colony Optimization (ACO) algorithms, which are biologically inspired algorithms which employ reinforcement learning to solve problems. Reinforcement learning was a topic we covered in class, and the section we did on Genetic Algorithms inspired us to explore the field of biologically inspired algorithms for our final project. We read 3 different papers on ACO algorithms pertaining specifically to TSP, and decided to try it for ourselves.

2 Background

The intuition behind this algorithm comes from the way ant colonies forage for food. An ant's goal is to find food and bring it back to the nest. A key property of ants is that as an ant crawls, it leaves pheromones behind it, which other ants can detect. This pheromone evaporates, so the strength of the scent decreases over time. When an ant is looking for food, it initially wanders randomly, but when it detects a pheromone, it follows its nose with a certain probability. Because the pheromone scent strength decays over time, shorter paths which can be traversed more frequently thus accumulate stronger pheromone scents, and so eventually, the ants collectively find the shortest path between their nest and the food. [2]

3 Representations

An important part of this project was deciding how to represent the different components.

We had a Graph class in which a graph was represented by an adjacency matrix where the value at matrix[i][j] was the distance between node i and node j. Nodes were identified with unique integer values, 0 through $n - 1$.

There were a few values we needed to track pertaining to each edge ij. For these, we also used matrices, in which the entry ij denotes the value corresponding to edge ij. We had a matrix for visibility (the inverse of the length of an edge), the pheromone strength of an edge, and the probability of an edge being traversed. For the probability matrix p, when at node i, the array $p[i]$ represented the probability distribution over all of node i's neighbors. Thus, $p[i][j]$ represented the probability of moving from node i to node j.

We also had a separate Path class, where a path was represented as an array of length n. The value at path[i] represented the node to visit after visiting node i. We also had 2 helper functions, a function to add an edge to a path, and to calculate the cost of the path.

Finally, we had a Solver class with a solve() method that returned a Path object. We implemented 4 different solvers: a Greedy Solver, and 3 variations of ACO algorithms which will be explained in more detail.

4 Ant System

The first algorithm that we implemented was the basic Ant System algorithm, which follows fairly directly from the intuition of the way real ant colonies behave.

At each time iteration, k ants would individually find tours over the graph, and after finding a tour, they would update the pheromone strengths of the edges in that tour. Although each ant modified the global representation of pheromone strengths, it found a path according to a copy of the pheromone strengths made at the very beginning of the time iteration. This modification was to mimic the idea of ants moving at the same time, rather than sequentially.

We randomly assigned each ant to start out at 1 of the n nodes. Then, starting from its start point, an ant finds a path by just figuring out which node to move to next, and then where to move from there, and so on. Each ant moves from i to j with probability [1]:

$$p_{ij} = \frac{\tau_{ij} \cdot {v_{ij}}^b}{\sum_j \tau_{ij} \cdot {v_{ij}}^b}$$

τ_{ij} is the pheromone strength on edge ij
$v_{ij} = \frac{1}{w_{ij}}$ is the visibility of edge ij (the inverse of the distance)
b is a tunable parameter to adjust whether ants favor short distance or strong pheromone strength
w_{ij} is the weight of edge ij

After finding a tour, we updated the pheromone strengths based on how good the tour was. Each edge of tour is updated as follows:

$$\tau_{ij} = (1 - d) \cdot \tau_{ij} + \delta$$

[1] These and the following equations relate closely to those shown in Bonabeau, 1996

d is the rate of decay (or 1 minus the discounting factor)
$\delta = \frac{C}{l}$ is the amount of pheromone to add to edge (not dependent on i, j)
C is the cost of a reasonably good tour (in our case, the cost of the greedy solution)
l is the cost of the tour found

After t iterations, we deterministically find and return the path with the highest pheromone levels.

5 Ant System with Elitist Ants

Our first modification was to favor more exploitation by introducing the concept of "elitist" ants. Elitist ants only reinforce the "best" edges, that is, the edges in the shortest tour found so far. Thus, we kept track of the shortest tour found and its cost, and at every time iteration, we simulated e elitist ants reinforcing that tour with the following update step to include an extra term:

$$\tau_{ij} = (1 - d) \cdot \tau_{ij} + \delta + e \cdot \frac{C}{l^*}$$

e is the number of elitist ants
l^* is the cost of the best tour so far

6 Ant Colony System

Our final modification was to promote *directed* exploration by the other ants. This modification moved away from the biological inspiration, as an ant traversing an edge actually decreases the amount of pheromone along that edge. This update occurs as follows, for every ant:

$$\tau_{ij} = (1 - d) \cdot \tau_{ij} + d \cdot \tau_0$$

The only ant that increments pheromone strengths is the ant who has found an improvement to the best path found so far. This modification promotes exploration, specifically in the area surrounding the best path found so far. This update occurs as follows, only for the ant who has found an improvement to the best tour:

$$\tau_{ij} = (1 - d) \cdot \tau_{ij} + \delta$$

$\tau_0 = \frac{1}{n \cdot C}$ is the initial τ given to all edges
$\delta = \frac{1}{l^*}$ is the new update factor

Additionally, we modify how ants find paths; when ants explore, they only explore the c nearest unvisited neighbors, which also promotes more exploitation. With probability r, the ant takes the most desirable edge (exploitation). Else, it explores like in Ant System, except it only considers the candidate list, the c closest nodes. If there are no unvisited candidates, the ant takes the closest unvisited node. Let $q \sim \text{Unif}(0, 1)$ be a value sampled before an ant leaves a node. Each ant moves from i to j where:

$$j = \begin{cases} \text{argmax}(\tau_{ij} \cdot {v_{ij}}^b) & q \leq r \\ \text{sample from}(p_{ij}) & q > r \end{cases}$$

7 Experiments

7.1 Methods and Models

We used 4 different graphs (with known best solutions), found in online datasets:

- Small: 15 nodes, known best solution: 284.38
- Medium: 29 nodes, known best solution: 27603
- Large: 48 nodes, known best solution: 33523.7
- Extra-Large: 131 nodes, known best solution: 564

We ran each of the 3 variations of ACO discussed above on each of these graphs for varying numbers of iterations: 1, 5, 10, 25, and 50, in each run benchmarking how long the program took to terminate. We used a MacBook Pro with 2.5 GHz Intel i7 processor and 16GB of RAM.

For each of the variations, we were interested in 2 relationships:

1. Time (sec) vs. Number of Nodes
2. Accuracy vs. Number of Iterations

   We represented accuracy as the cost of the found solution divided by the cost of the optimal solution. Note that in the bar graphs below, the different colors represent different numbers of iterations.

To test our algorithms, we first determined the path that first ant in a given iteration traveled on a known graph. We then established breakpoints and traced the various pheromone values, ensuring that they strengthened or lost strength as expected for the later ants in the iteration. We also tested by printing the graph at each iteration, ensuring that the refinement at each iteration made sense given the previous iteration, and that the best path cost was being gradually reduced throughout the algorithm.

7.2 Results

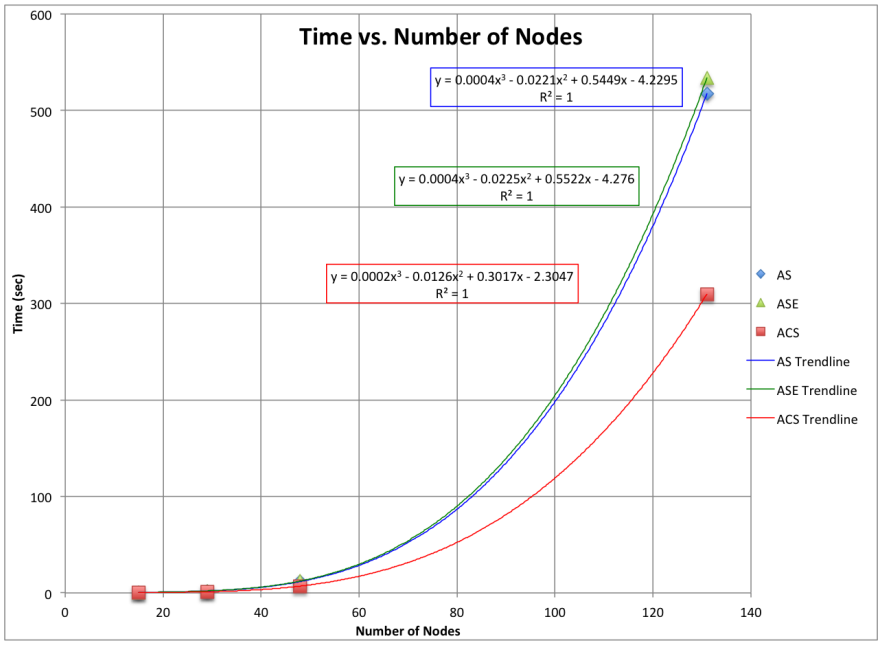

Figure 1: Timing

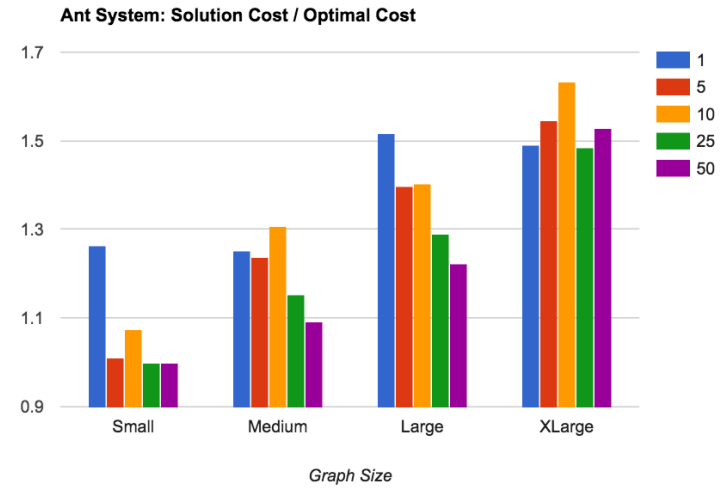

Figure 2: Ant System

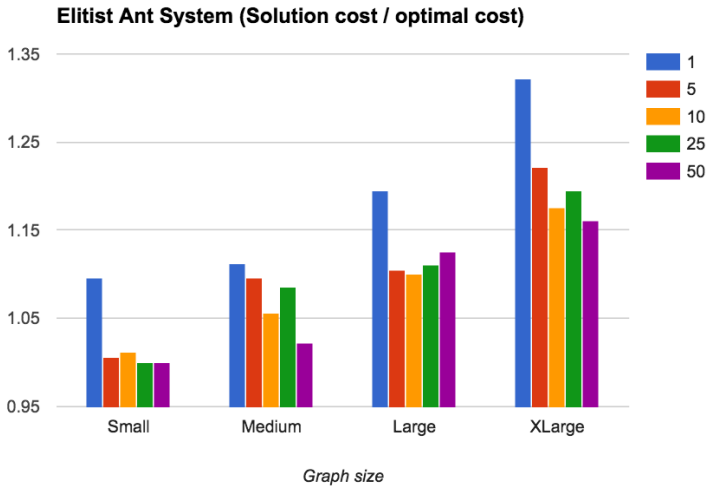

Figure 3: Ant System with Elitist Ants

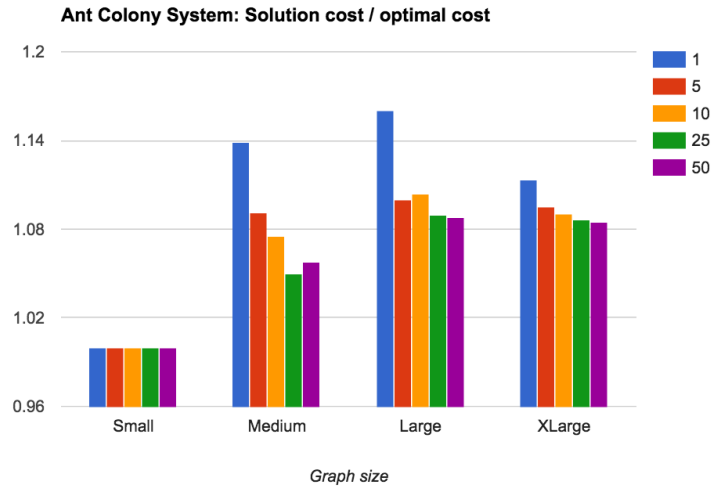

Figure 4: Ant Colony System

7.3 Discussion

Our trendlines show a runtime of $O(n^3)$. This makes sense, since we used $O(n)$ ants per iteration, and each of these ants did $O(n)$ work in selecting a next node over the course of $O(n)$ nodes.

Our results show us that while Ant System finds fairly good solutions (1.07 times the optimal solution on the small graph after 10 iterations in 0.21 seconds), and Ant System with Elitist Ants provides a significant improvement on accuracy with an insignificant impact on time (1.01 times the optimal solution in 0.23 seconds), they did not perform well as the graph grows (1.63 times the optimal solution on the xlarge graph after 10 iterations in 517 seconds for AS, and 1.18 times the optimal solution in 534 seconds for ASE). As the graph grows, Ant Colony System shows better performance in terms of both accuracy and time (1.09 times the optimal solution in 310 seconds).

An interesting takeaway from this trend is that while ant colonies were a good inspiration for this algorithm, we had to move away from the model a bit (become less realistic in terms of the way ants actually work) in order to improve performance. For example, one of the benefits of using ACO is that you implicitly handle the exploitation vs. exploration tradeoff (because of the pheromone strength decay). However, each of our modifications tweaked the tradeoff (encouraging exploitation in ASE and encouraging directed exploration in ACS), showing that the ant modeling can still be improved.

It would have been ideal to be able to run more iterations on larger graphs, but due to limited computing resources, this was not reasonable.

Another extension that we would have liked to explore was parallelizing the work done on all of the algorithms to improve runtime. This is possible because the ants all work on their own copy of pheromone strengths at any given iteration, so it should be possible to parallelize the work of each ant. Because we had as many ants as there were cities, this would significantly improve the runtime of the algorithms.

Our algorithm involved many tunable parameters; the above data uses the values suggested by Bonabeau et al. In an attempt to achieve better performance, we wrote an additional script to experiment with the parameters ourselves and attempt to learn which values were optimal, which is discussed in the next section.

7.4 Learning Parameters

We examined the constants in the Elitist Ant System and Ant Colony System variations. For the Elitist Ant System variation, we examined the visibility factor b, the decay rate of pheromones d, and the number of elitist ants e, which we refer to as the *elitism* (see equations in sections 5 and 6). We ran Elitist Ant System on the medium graph (29 nodes) for a total of 225 different combinations of values close to those suggested by Bonabeau et al. For each combination of parameters, we took the average of 5 trials, each of which ran the algorithm for 8 iterations. See table 1 for the combinations of parameters that yielded the ten lowest-cost paths.

For the Ant Colony System variation, we examined b and d as in Elitist Ant System, in addi-

tion to the exploitation probability r (see equations in section 7). As with Elitist Ant System, we ran ACS on the medium graph, again taking the average of 5 trials of 8 iterations each, for each of 405 combinations of parameters. Table 2 gives the combinations of parameters that yielded the ten lowest-cost paths.

For both algorithms, the b-value among optimal solutions was quite consistent, but the other parameters were subject to a good deal of variation. In Ant Colony System, good solutions were achieved with a wide variety of r- and d-values, though they seemed to be close together whenever a good solution was achieved (this makes sense, since as pheromones decay increases, exploitation probability must increase in order to detect it). Interestingly, among all the combinations of parameters in both algorithms, the best solution was achieved by Elitist Ant System. Brief testing of these new parameters for Elitist Ant System on the extra-large graph confirmed that Ant Colony System still performed better as the size of the graph increased.

The results for ASE and ACS are shown below.

Solution cost	Visibility factor b	Pheromone decay d	Elitism e
28066	6	0.3	4
28402	7	0.4	3
28411	6	0.4	6
28423	5	0.1	5
28456	7	0.2	3
28555	6	0.1	3
28626	7	0.7	6
28645	5	0.1	6
28667	7	0.2	7
28678	7	0.5	7

Table 1: Parameter tuning for Elitist Ant System.

Solution cost	Visibility factor b	Exploit prob r	Pheromone decay d
28490	4	0.4	0.5
28579	4	0.1	0.3
28595	4	0.3	0.3
28761	4	0.2	0.2
28781	4	0.6	0.4
28803	4	0.4	0.7
28830	4	0.2	0.1
28895	4	0.2	0.8
28902	3	0.1	0.2
28908	4	0.1	0.4

Table 2: Parameter tuning for Ant Colony System.

References

[1] Andre Rohe. "VLSI Datasets," 2013. Available: http://www.math.uwaterloo.ca/tsp/vlsi/index.html.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford UP, 1999.

[3] Gerhard Reinelt. "TSPLIB - A Traveling Salesman Problem Library," *ORSA Journal on Computing*, Vol. 3, No. 4, 1991.

[4] Ivan Brezina Jr. and Zuzana Cickova. "Solving the Travelling Salesman Problem Using the Ant Colony Optimization." *University of Economics in Bratislava Management Information Systems*, Vol. 6, No. 4, 2011.

[5] Marco Dorigo and Luca Maria Gambardella. "Ant colonies for the traveling salesman problem." *BioSystems*, 1996.

[6] University of Waterloo. "DJ38 - Djibouti Computation Log," 2001.
Available: http://www.math.uwaterloo.ca/tsp/world/djlog.html.

[7] Zar Chi Su Su Hlaing and May Aye Khine. "Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm." *International Journal of Information and Education Technology*, Vol. 1, No. 5, 2011.

A Program Trace

Here, we will show our program solving TSP on the extra-large graph (131 nodes, optimal cost is 36485.84), displaying the best path found over all the ants in a certain iteration after 1, 10, 25, 50 and 100 iterations.

Iterations	Cost	Cost/Optimal Cost
1	41286.22	1.23
10	39490.35	1.18
25	38213.93	1.14
50	41216.62	1.23
100	36485.84	1.09

Table 3: Intermediate solutions of ACS on a large graph.

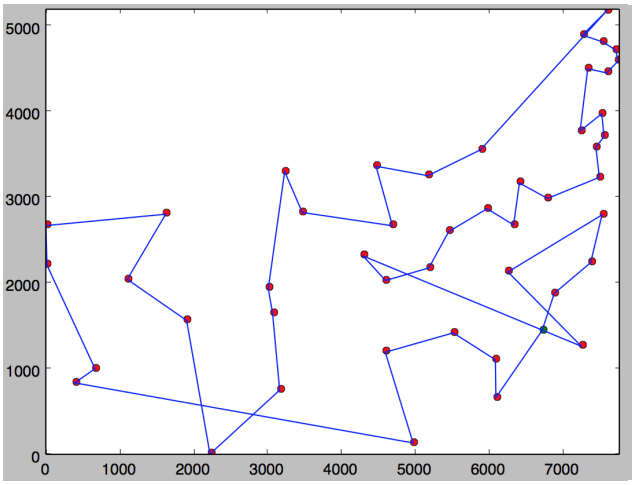

Figure 5: 1 iteration

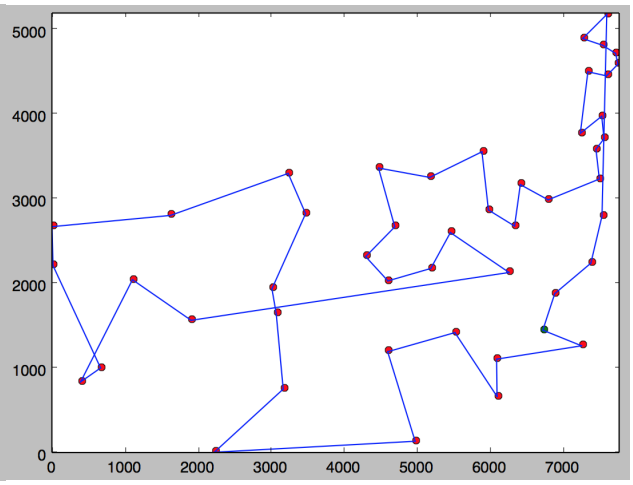

Figure 6: 10 iterations

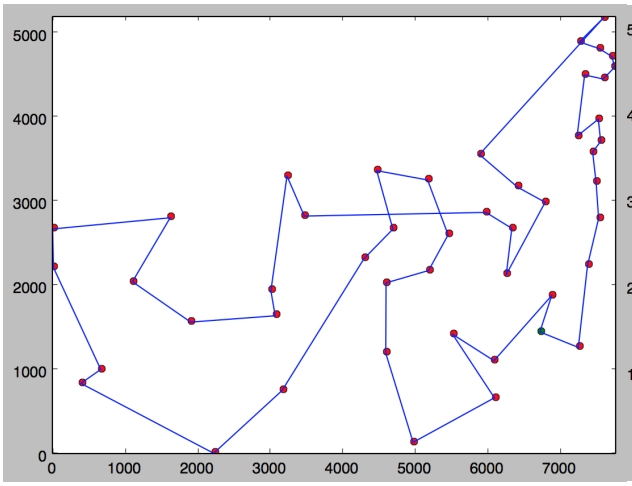

Figure 7: 25 iterations

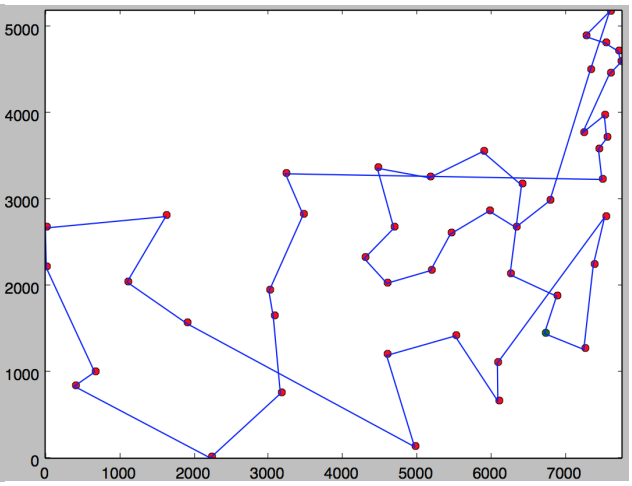

Figure 8: 50 iterations

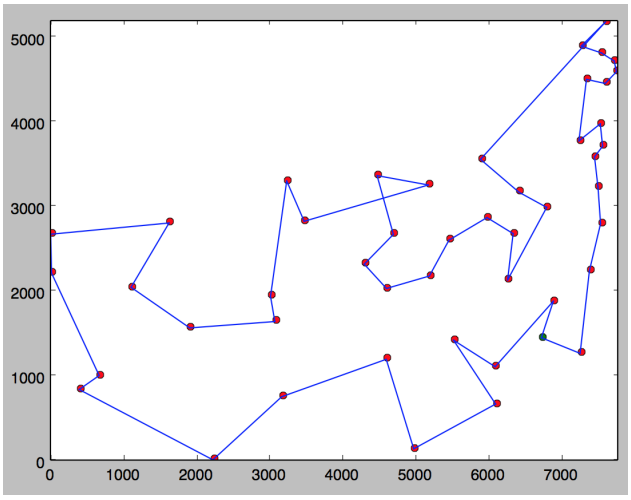

Figure 9: 100 iterations

B System Description

Our executable program ”solve.py” takes 3 command line arguments: problem size, solver, and number of iterations. The acceptable values for each of these arguments are:

- problem size: small, medium, large, xlarge
- solver: AS, ASE, ACS
- number of iterations: a non-zero, positive integer

These instructions are also displayed if you attempt to run the program with incorrect arguments.

The terminal outputs the cost of the solution and the time it took to calculate. There will also be a graphical pop up display of the graph and the tour that was found.

C Group Makeup

We both worked together (pair programming) on this project, with the exception that Billy wrote the script to find optimal constant values.