Ant Colony Optimization for the Traveling Salesman Problem

Marcus Christiansen, Ernesto Garcia, Konstantine Mushegian

April 18, 2017

Abstract

Ant Colony Optimization (ACO) is a probabilistic technique for finding "good" paths through graphs. An optimization problem that ACO is effective at solving is the Traveling Salesman Problem (TSP), where one seeks to find the shortest path through a graph of nodes, in which every node is visited exactly once and the path must return to the first node in the traversal. In this paper, we will introduce two popular extensions to the general ACO, Elitist Ant System (EAS) and Ant Colony System (ACS), and first compare the performance of the two algorithms for a changing number of ants on large problem sizes, before determining "good" parameters for ACS on smaller problems. We first vary the number of ants that traverse the graph space at every iteration on large problem sets. From this testing, we conclude that more ants leads to better results for both ACS and EAS when dealing with larger problems. We also conclude that ACS produces better results than EAS when running on large problem sizes. We then vary all parameters on smaller graph sizes for ACS (as this algorithm produced better results) to determine which parameter set produces the best results

1 Introduction

The Ant Colony Optimization (ACO) algorithm is a probabilistic optimization technique for stochastic combinatorial optimization that can be used to determine "good" paths through graphs. It is adapted from ant colonies found in nature; specifically from the way that they search for food using a population based search approach. ACO works similarly in that we have a predetermined amount of ants traveling through the graph space, probabilistically choosing the next path to traverse. As the ants travel through the graph, they deposit pheromones on the edges that they traverse. The ant's probabilistic decision is based on the amount of this pheromone, where the ant is proportionally more likely to choose the path with more pheromone. As other ants explore the graph, they are more likely to stop exploring randomly and follow the pheromone trail. The pheromone on each leg evaporates over time, reducing its attractive strength. As certain paths take longer to traverse, the more pheromone evaporates, reducing its attractive strength relative to shorter paths. After several iterations, due to the combination of pheromone depositing and evaporation, a "good" path is determined, as the positive pheromone feedback results in a single path being chosen.

One of the optimization problems that ACO is effective at solving is the Traveling Sales Problem (TSP). This is a mathematical problem in which one tries to determine the shortest path between nodes in which each node is visited only once. ACO is a good technique to solve the TSP, as the TSP can be modeled as a graph, and ACO, as discussed above, can be used to determine a "good" short path between the nodes. We will be using two ACO methods, the Ant Colony System, and the Elitist Ant System, to solve this problem.

The goal of our testing was to determine the affect of changing the number of ants had on the performance of both ACS and EAS when dealing with larger problem sizes, and to compare the two algorithms to determine which performed better in the given circumstances. Our testing consisted of varying the number of ants parameter for both ACS and EAS between 10, 20 and 30 ants, and running both algorithms on a series of test problems. The test problems graph sizes included 1173, 2392, 3038, 4461 and 5915. We ran each set of parameters on a specific graph 3 times so we would be able to take the average of the three runs in order to account for possible error in individuals runs. The metric with which we used to test the performance of ACO and EAS is the fraction of the respective algorithm's final best tour and the optimal tour length. The larger the fraction, the further the algorithm's final tour is from the optimal tour, thus resulting in a larger error measure. We compare the error margin for each set of parameters and conclude as to which ACO algorithm and parameters produce the best results for larger problem sizes. Additionally, as we were not able to run several tests on larger problem sizes, we also vary all parameters on smaller graph sizes for ACS (as this algorithm produced better results) to determine which parameter set produces the best results.

From our testing, we can conclude that, for both ACS and EAS, more ants generally produced better results. This coincides with our logic, as having more ants explore the graph produces more potential candidates from which to choose from. Additionally, from our ACS testing on smaller problems, we were able to determine a set of good parameters when running ACS on smaller problems. We can conclude that more ants, emphasis on distance and local pheromone dissipation, while less emphasis on pheromone levels and less global pheromone evaporation produced more accurate results.

In this paper, we will be testing each of the two ant colony optimization algorithms, investigate their performance on non-trivial instances of the Traveling Salesman Problem, and determine which method and which parameters produce the better results. In Section 2, we further describe Ant Colony Optimization as well as the different components that make up this optimization technique, and provide the corresponding equations for each of these components. In section 3, we describe the TSP, and discuss why ACO is an appropriate technique to solving this problem. In section 4, we describe the two ACO algorithms, Ant Colony System and Elitist Ant System, and discuss how we are going to implement them. In Section 6, we will describe our experimental methodology, detailing the experiments that we ran, and our reasons behind them. In Section 7, we discuss and analyze the results our experiment. In Section 8, we discuss possible further work for our project, before providing some conclusions in Section 9.

2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a probabilistic technique for solving computational and numerical problems that can be reduced to finding good paths through graphs. ACO belongs to the family of ant colony algorithms, which is a part of swarm intelligence methods. ACO employs metaheuristics; metaheuristic is a type of heuristic that is designed to determine "good enough" solutions to optimization problems, especially when the available information is incomplete or when the computational capacity is limited. One caveat about metaheuristic is that it provides no guarantees about determining global optimal solutions.

ACO was originally proposed by Marco Dorigo in 1992 and is based on the behavior of ants looking for a path between their colony and the source of food. In nature, some species of ants wander randomly on the search for food; upon finding the food they return to their colony, laying down pheromone on their trail. When other ants come across the pheromone path laid down by other ants, they stop exploring randomly and follow the pheromone trail. If they find food at the end of the trail, they return to the colony while laying down more pheromones on the path and reinforcing it.

Pheromones evaporate over time and weaken the path. Since ants only lay down the pheromones when returning to the colony with food, the longer the path the more time the pheromones have to evaporate; thus, shorter paths between the colony and food are naturally rewarded by this behavior. Pheromone evaporation also prevents ants from converging to a locally optimal solution (locally closest food source). If there was no evaporation every ant in the colony would end up following the path laid by the first explorers. This would confine the exploration, since every following ant would be disproportionately attracted to the random paths laid down by the first explorers.

The net result is that once a good (short, shortest) path from the ant colony to the food source is found by one ant, other ones are more likely to follow it. The positive feedback results in all ants following the single path from the ant colony to the food source.

For our purposes, the process above is adapted to find the shortest path from one point in a graph to another. We have a starting point and destination point in a graph and we want to find the shortest path between them. Each ant explores the search space individually and builds its own path using the edges between each node of the graph. The question that arises is how does each ant decide which edges to add to its tour in order to minimize its travel distance? When we consider ACO in this way we can condense the process above into three distinct steps: probabilistically choose the next edge to add to its tour, lay down pheromones on helpful edges to aid ants building their tours in the next iteration, evaporate pheromones on edges that may not be useful to also help ants in the next iteration. Dorigo's original ant system executed these steps using the following formulations.

2.1 Choosing an Edge

An ant is going to be choosing the next edge to add to its tour probabilistically. The probability that an ant decides to add a particular edge is calculated using the following formula [4].

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_k} \tau_{ij}^\alpha \eta_{ij}^\beta} & \text{if} \in N_k \\ 0 & \text{otherwise} \end{cases}$$

where

- τ_{ij} = pheromone level on edge from node i to mode j
- $\eta_{ij} = \frac{1}{d_{ij}}$ and d_{ij} = distance from i to j
- α, β are positive constants
- N_k are nodes that have not been visited yet

Notice that α and β control the importance of the pheromone levels on edges and the distance between nodes respectively when we calculate the probability that we should choose an edge. We can make the pheromone levels overpower the ratio above by increasing α and vice versa. We will be manipulating these constants in our experiment and observe their impact on the effectiveness of the ACO algorithm

2.2 Laying Pheromones

After an ant has built its tour from the starting point to the destination it will lay down pheromones in quantities directly related to the length of the path that it took. This is expressed in the following piecewise function.

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{L_k} & \text{if (i,j)} \in \text{tour ant } k,\ L_k = \text{tour length ant } k \\ 0 & \text{otherwise} \end{cases}$$

The pheromone levels of each edge can be calculated in a straightforward fashion. It is simply the sum of all the pheromones laid down by each ant that used that edge in its tour. Notice how a shorter path leads to more pheromones being laid and therefore a higher probability that it will be chosen by ants in the next iterations.

2.3 Pheromone Evaporation

As explained earlier in the report, after each iteration there has to be some degree of pheromone evaporation. This ensures that ants know that there are edges that have been used and determined to be inefficient in finding the shortest path and therefore were not used again. To do this Dorigo utilized the following function.

$$\tau_{ij} = \tau_{ij}(1-\rho) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

where

- $0 < \rho < 1$ here ρ is called the evaporation factor

We will also be manipulating the evaporation factor to observe how it affects the ACO algorithm. A large ρ will cause the pheromones to evaporate rapidly after each iteration and a small ρ will lead to no evaporation of pheromones.

3 Traveling Salesman Problem

Traveling Salesman Problem (TSP) is a classic problem in computer science and mathematics, and has commanded much attention over the years because it is very easy to describe and significantly harder to solve. TSP asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". A simple question that is surprisingly hard to answer; TSP belongs to the class of NP-complete (non-deterministic polynomial-time hard) and is important in operations research and theoretical computer science. TSP is also important because it is representative of a larger class of problems known as combinatorial optimization problems[2]. Combinatorial optimization consists of finding the best object among the set of objects; in case of TSP our goal is to pick the best ordering of cities from the given list.

Brute force would be an obvious approach to this problem, however performing brute force becomes infeasible even for very low number of cities. For example, for a 16 city TSP there are 653,837,184,000 distinct routes that would need to be evaluated by the brute force algorithm[2].

ACO is appropriate for this problem because it builds subsequent solutions from the strongest parts of the best solutions derived from previous iterations. Given the large number of possibilities in TSP, ACO sounds like a good choice because it will by default only explore a smaller subset of possibilities which will let us obtain an answer in a feasible amount of time.

4 ACO Improvements and Extensions

Various improvements and extensions have been proposed to the original ACO algorithm proposed by Dorigo in 1992. In this paper we discuss two extensions:

the Ant Colony System and the Elitist Ant System. These extensions are described in the next two sections. We will begin by first presenting the difference between this algorithm and the original ACO algorithm, and then describe how these differences influence choosing an edge, laying pheromone and pheromone evaporation.

4.1 Ant Colony System

In the Ant Colony System (ACS), ants use a pseudorandom proportional rule, where the probability that an ant will move between cities i and j is dependent on the value of a variable q_0. With probability q_0, the available path that maximizes $\tau_{ij}\eta_{ij}^{\beta}$ is chosen, otherwise the original above-described ACO method is used. This is a greedy algorithm, where pheromone information is exploited, resulting in less exploration. Additionally, a pheromone update is made, where pheromone is only increased on the best-so-far solution [3]. See Appendix A for ACO pseudocode.

4.1.1 Choosing an edge

When choosing an edge, With probability q_0, the available path that maximizes $\tau_{ij}\eta_{ij}^{\beta}$ is chosen, otherwise the original above-described ACO method is used [4]:

With probability q_0, choose the available path that maximizes $\tau_{ij}\eta_{ij}^{\beta}$, else:

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}} & \text{if} \in N_k \\ 0 & \text{otherwise} \end{cases}$$

4.1.2 Local Pheromone Depositing and Evaporation

In ACS, a local pheromone update is performed for each ants tour, where pheromone is updated on the legs of the tour taken by the ant. The update is constructed so as to dissuade other ants to follow the same cities in the same order [4].

$$\tau_{ij} = (1-\epsilon)\tau_{ij} + \epsilon\tau_0 \text{ where}$$

- $0.0 < \epsilon < 1.0$ (typically 0.1)
- $\tau_0 = \frac{1}{n \cdot L_{nn}}$ where
  - n = number of ants
  - L_{nn} = length of a nearest neighbor tour, which is a tour constructed by starting at an arbitrary city and selecting the closest unvisited city to visit next until all cities have been visited

This increases exploration by reducing attraction of frequently used edges.

4.1.3 Global Pheromone Depositing and Evaporation

After an iteration is complete all ants have completed their tour, a global pheromone update is performed on all edges of the graph, where pheromone is evaporated from every edge, however, pheromone is also deposited on the edges that are in the current best tour [4].

$$\tau_{ij} = \tau_{ij}(1-\rho) + \rho\Delta\tau_{ij}^{bfs} \text{ where}$$

- $0.0 < \rho <1.0$, here ρ is called the evaporation factor (typically 0.1)
- $\tau_{ij}^{bfs} = \begin{cases} \frac{1}{L^{bsf}} & \text{if (i,j)} \in bsf,\ L^{bsf} = \text{length of } bsf \\ 0 & \text{otherwise} \end{cases}$

4.2 Elitist Ant System

The Elitist Ant System(EAS) differs from Dorigo's original ACO in that it gives preference to edges that are a part of the best path so far. The best path so far is reevaluated after every iteration. If a shorter is path in found in a subsequent iteration, this path is recorded and every edge in this path will be chosen with a higher probability. This is actually implemented in the second and third phases where the ants lay down pheromones and where the pheromones are evaporated. The first step where we determine the probabilities of choosing the next edge remains unchanged. The following explains the new implementation. See Appendix B for EAS pseudocode.

4.2.1 Laying Pheromones

The laying of pheromones in EAS is very similar to that of the typical ACO implementation, however, if the leg being updated is a part of the current best tour, additional pheromone is deposited on that leg. This is done in hopes that the shortest tours will be reinforced and future tours will consist of edges that belong to best path so far. The pheromone deposit for each leg is described below, where τ_{ij}^{+} will be the pheromone that is deposited on leg (i,j) [4]:

$$\tau_{ij}^{+} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bsf} where$$

- $\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \text{if (i,j)} \in \text{tour ant } k,\ L_k = \text{tour length ant } k \\ 0 & \text{otherwise} \end{cases}$
- $\Delta\tau_{ij}^{bfs} = \begin{cases} \frac{1}{L^{bsf}} & \text{if (i,j)} \in bsf,\ L^{bsf} = \text{length of } bsf \\ 0 & \text{otherwise} \end{cases}$
- e = elitism factor (typically number of ants)

4.2.2 Global Pheromone Depositing and Evaporation

In EAS, after an iteration is complete all ants have completed their tour, a global pheromone update is performed on all edges of the graph, where pheromone is evaporated from every edge, however, pheromone is also added according to the rules described above [4]:

$$\begin{aligned}\tau_{ij} &= \tau_{ij}(1-\rho) + \tau_{ij}^{+} \\ &= \tau_{ij}(1-\rho) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bsf}\end{aligned}$$

- $0 < \rho < 1$ here ρ is called the evaporation factor

5 Code Description

In order to optimize our code and avoid performing the same work again, we used several matrices to store already computed values that may have been used again. For both algorithms, we precomputed the distances between the cities in the problem so that the distances would only have to be computed once, and so that the ants could determine the distances between cities in constant time. This matrix is called distanceMatrix. For EAS, we also create a matrix to store the probability values computed when determining which city to pick next. When ever a value $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ is calculated, it is stored in a matrix, so that if needed again, it can be accessed in constant time. This matrix was re-initialized to -1 values for each iteration as the global pheromone updates change the pheromone levels and thus the probabilities will change. This optimization would not be made for ACS, as the local pheromone changes prevents us from storing already computed values, as the values change from ant to ant. This slowed down ACS significantly, as we had to compute computationally-expensive equations for every ant at every iteration.

Finally, in order to better visualize the output of the two algorithms, we implemented a simple visualizer using the Java Swing framework. This utility displays the nodes and the computed route once the algorithms are finished running. Since different problems use different coordinate scales we scale the raw coordinates (from .tsp files) to fit in the utility window. This works well most of the time, however sometimes certain nodes end up outside the bounds of the window. Implementing a scrollbar in Java Swing is notoriously confusing, thus the best way to display off-canvas nodes is to try and resize the window in such a way that will include the nodes of interest.

6 Experimental Methodology

The testing for this project consisted of varying a parameter of both ACS and EAS and comparing the resulting performance of the algorithms when solving large problems. We initially planned to vary all parameters and ultimately determine which set of parameters produced the best results for ACO and EAS. However, as each test took a long time to complete, especially the larger problems, we decided to only vary the number of ants, and determine its influence on the success of ACO and EAS when solving large problems.

After determining that ACS produced the best results for larger problems, we ran testing so as to optimize the parameters for it, running on smaller problems in order to be able to test all parameter combinations. From this, we could determine the best parameter combination for ACS for smaller sized problems. Future work includes testing the resulting "good" parameters from this testing to determine if these parameters scale well to larger problems.

6.1 Large Problem Size Testing

One goal of our testing was to determine the affect of changing the number of ants had on the performance of both ACS and EAS when dealing with larger problem sizes, and to compare the two algorithms to determine which performed better in the given circumstances. The number of ants with which we tested both algorithms include: 10, 20 and 30 ants. The size of the problems we experimented with include 1173, 2392, 3038, 4461 and 5915. The other parameters were set to the rule-of-thumb parameter settings and kept constant for all test runs. We recorded the final cost of the best tour after a set number of iterations, and divided it by the optimal tour cost to determine the error of our final solution. This error would become the measure of the success of our algorithms, and the metric against which we would compared different tests. Each test was run three times to determine the average performance for each set of parameters.

We compared the resulting error measure for each set of parameters and algorithms, and determined which set of parameters and ACO algorithm produces the best results. Ultimately, we make a recommendation as to what number of ants and algorithm should be run when wanting to solve larger problem sizes.

The parameters that were used to perform experiments on each of the two ant systems[4] for the large problem testing are listed below.

6.2 Large Problems Experimental Parameters

6.2.1 General Parameters

- Number of ants:
  - 10
  - 20
  - 30
- Number of iterations:
  - 200
- α - Pheromone influence
  - 1
- ρ - Pheromone evaporation factor
  - 0.1
- β - Heuristic influence
  - 4

6.2.2 Ant Colony System Parameters

- ϵ - Factor controlling "wearing away" of pheromones
  - 0.1
- q_0 - Probability that an ant chooses the best leg for the next leg of the tour, instead of choosing probabilistically.
  - 0.9

6.2.3 Elitist Ant System Parameters

- e - Elitist Ant System elitism factor
  - Number of ants for a given run

Each combination of parameters was run 3 times, resulting in a total of 90 tests with a running time of 5 days.

6.3 ACS Parameter Optimization

As we were not able to run several tests on the larger problems, we varied the parameters for ACS on smaller problems, as this algorithm produced the best results when running on larger problems. We varied the parameters largely so as to determine the best parameter combination to achieve the best results for ACS. We ran on problems of size 48, 52, 127, 130.

We compared the resulting error measure (as described previously) for each set of parameters and algorithms, and determined which set of parameters produces the best results. Ultimately, we make a recommendation as to what parameters should be used when running ACS on smaller problems.

The parameters that were used to perform experiments on the ACS algorithm are listed below:

6.4 ACS Parameter Optimization Experimental Parameters

- Number of ants:
  - 20
  - 30
  - 40
- Number of iterations:
  - 100
  - 200
  - 300
- α - Pheromone influence
  - 1
  - 2
  - 3
- ϵ - Factor local evaporation of pheromones
  - 0.1
  - 0.2
  - 0.3
- ρ - Global pheromone evaporation factor
  - 0.1
  - 0.2
  - 0.3
- β - Heuristic influence
  - 2
  - 3
  - 4
- q_0 - Probability that an ant chooses the best leg for the next leg of the tour, instead of choosing probabilistically.
  - 0.7
  - 0.8
  - 0.9

Each combination of parameters was run once resulting in a total of 2187 tests with a running time of 20 hours.

6.5 Testing Automation

We decided to write a Python script that would let us run a large number of experiments on the Bowdoin College computing network, Dover. We chose Python for this because the language is simple and expressive enough that we could build a comparatively complicated script in a relatively short amount of time. Since some of the problems took quite a long time to run, we parallelized the testing script according to the number of computing cores available on the machine, which significantly sped up the testing. The results of our automation attempts can be found in auto-test.py file that is included in the project repository.

In order to keep track of the progress made by the testing framework, we set up a small notification system that sends a text message every 15 minutes with the progress made by the script. This way we could make sure that the tests kept running in our absence; it is also a useful tool to detect crashes during testing. The notification tool relies on Twilio and can be found in notify.py file that is included in the project repository.

Once the automated testing framework was completed we set out on our search. Below we present the parameters that we experimented with for each algorithm. We decided to try every possible combination of all parameters in order to find the best one.

7 Results

7.1 Format

7.1.1 Large Problem Testing

We present the results from our experiments as tables for ACS (Appendix C) and EAS (Appendix D) respectively. The table contains the average error measure (described above) for each set of parameters for each test problem over the three runs.

We also transform our data into a series of plots to demonstrate the impact of how the number of ants influences the performance of the two algorithms. The plots plot number of ants against the average error term for that number of ants. We present two graphs, one for ACS and one for EAS, displaying the above described results for all test problems. The graphs are presented in Appendix E and F respectively.

7.1.2 ACS Parameter Optimization

We present the best parameter combination for ACS below as a list of the best parameters.

7.2 ACS Parameter Optimization Results

The parameters that produced the best results for ACS when running on smaller problems include:

- Number of Ants = 30
- Number of iterations = 300
- $\alpha = 1$
- $\beta = 4$
- $\rho = 0.1$
- $\epsilon = 0.3$
- $q_0 = 0.7$

7.3 Discussion

7.3.1 Large Problem Testing

From our results and supporting graphs in Appendix E and F, we can conclude that a larger amount of ants generally produces better results for both ACS and EAS. From our results, we saw that 30 ants produced better results than both 20 and 10 ants, and that 20 ants produced better results that 10 ants. This makes sense, as having more ants explore the graph and form solutions increases the number of potential solutions that will be considered by the algorithm. As we can see form the graphs, there were instances when more ants produced worse results, however, considering the general downwards trend, we believe that this may be anomalies in the data that don't truly represent the true trends. Additionally, we also found that ACS produced better results compared to EAS when dealing with larger problems. Finally, we also saw that the larger the problem, the more error our algorithms produced. This also makes sense, as the larger the problem, the more potential candidate solutions can be created.

7.3.2 ACS Parameter Optimization

The goal of this experiment was to find the best combination of the seven parameters required by the ACS flavor of the ACO algorithm. While this experiment did not run on particularly large problems, it let us 'try' a large number (2187) combinations of parameters.

The results of this experiment agree with the ones from Large Problem Size Testing (Section 6.1) - the algorithm performs better with the higher number of ants. This makes sense because given a fixed amount of time, a large number of ants is more likely to find the solution, simply because more ants will try more possible paths from source to goal. The results of this experiment also favored a higher number of iterations. A reasoning similar to the number of ants can be applied in this case as well.

The optimal combination of parameters favored lower α and higher β, which means that distance was more important for picking the next city than the pheromones laid down by other ants. This is quite surprising, since we expected pheromones to play a larger role in the decision-making. Additionally, more local pheromone evaporation produced better results. This may be due to the fact that more local pheromone evaporation dissuaded following ants to follow the same route, thus encouraging more exploration, which may have resulted in the algorithm finding a better solution. However, less global pheromone evaporation, suggesting that large pheromone retention played a part in producing good results, although local pheromone evaporation was needed to encourage exploration. Finally, a lower q_0 value suggests that ACS's greedy heuristic did not lead to better results.

Something we would like to be able to determine in future testing would be if these good parameters determined here scaled well for large sized problems.

8 Further Work

The most significant improvement we want to make is to be able to run more tests on larger sized problems. While were able to successfully implement both ACO algorithms and investigate their performance on non-trivial instances of the Traveling Salesman Problem, the running time of our algorithms hindered our efforts of testing all the parameters that we would have liked. We were able to determine good parameters for ACS for smaller sized problems, however, we would have liked to be able to test these parameters on the larger problems to see if the parameters scaled. This additional testing would allow us to suggest the best parameter combination when running ACS and EAS on larger sized problems. In order to be able to run all of these additional tests, code optimization is critical.

Additionally, it would have been interesting to test if continually adding more ants would continue to improve the results.

We would also like to run the testing framework on the High Performance Computing (HPC) cluster. This would also improve the run time of our algorithms which in turn would lead to a larger number if experiments executed. In this way we would have a larger number of data points to better support the conclusions we presented in this report. For example, it would have been better to test our algorithms on different problems with a similar number of cities to see how the number of cities affects our results in reality. We did try to exploit the time we did have by running several experiments in parallel using our python script for auto-testing, but were limited in the amount of computing power available on our laptop computers.

Finally, an additional improvement for this project would be to add a feature to the visualizer that would let users manipulate the topology manually, which would let them see the effects of including or excluding certain points from the required route. This would be a dynamic feature that would allow users to see how the changes in the topology of the cities reflect on the performance of the algorithm immediately.

9 Conclusions

In this section we present the conclusions that we were able to draw from our testing described in Section 6 as well as the general lessons learned over the project.

According to our results from the Large Problem Testing (Section 6.1) we determined that both the ACS and EAS flavors of ACO performed better with the higher number of ants. Thus, when running both ACS and EAS, to achieve the best results, we recommend running with more ants. Additionally, we also found that ACS performed much better than EAS on larger problems. Thus, if running on larger problems, we would also recommend using ACS to achieve better results.

We were also able to draw some conclusions about Ant Colony Optimization by running this algorithm on smaller problems. We ultimately found an optimal set of parameters that produced the smallest error and these are presented in section 7.2. In this case 30 ants was sufficient to find the most optimal solution which is not the case for our other experiments for larger problems. This leads us to believe that for TSP problems with a smaller number of cities may require a smaller ant colony to reach the optimal solution. Additionally we found that 1 was our best value for α, which implies that though pheromones play an important role in remembering which edges are useful, they should not be the deciding factor for choosing the next leg on an ants tour. β on the other hand proves to be an important which implies that the distance between a city is more useful for finding the optimal solution in smaller problems. Additionally, q_0 works best with he smallest value we tested it on which means that it is better to choose the next leg on our tour probabilistically than always exploiting the "best" next city. When we combine this with the fact that ϵ,the local evaporation factor for pheromones after each ant builds it's tour, works best with the highest value, we see it is actually best to leave choosing the next city to probability instead of continuing to exploit solutions that have been found to be "optimal" in the past iterations.

We once again came to appreciate the power of writing re-usable, testable code; as well as the capability of automated testing. Many of the results presented in this paper would require hundreds of hours of manual labor to derive. We were able to automate most of our data collection and processing to minimize the time taken by menial tasks, which let us focus our efforts on performance and correctness of the algorithms presented in the paper.

10 Appendices

A Ant Colony System Pseudocode

Algorithm 1 Ant Colony System

```
Input: Ants$_{size}$, Iterations$_{size}$, $\alpha$, $\beta$, $\rho$, $\epsilon$, $q_0$
Output: tour$_{best}$
$\tau_0 \leftarrow$ 1 / computeCost(tour$_{NearestNeigbor}$)*Ants$_{size}$
tour$_{best} \leftarrow$ initializeRandomTour(Nodes)
cost$_{best} \leftarrow$ computeTourCost(tour$_{best}$)
initialPheromone $\leftarrow$ 1 / numCitiescost$_{best}$
PheromoneMatrix $\leftarrow$ initializePheromoneMatrix(initialPheromone)
while !StopCondition(currIteration , currTimeElapsed , currError) do
    for Ant$_k \in$ Ants$_{size}$ do
        while length(tour$_{Ant_k}$) != length(Cities) do
            if randDouble $\leq q_0$ then
                choose city$_i$ with max($P_{lastCity,i} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}$)
                tour$_{Ant_k} \leftarrow$ city$_i$
            else if randDouble $> q_0$ then
                choose city$_i \in$ Cities with $P = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}$
                tour$_{Ant_k} \leftarrow$ city$_i$
        cost$_{Ant_k} \leftarrow$ computeTourCost(tour$_{Ant_k}$)
        if cost$_{Ant_k} < cost_{best}$ then
            tour$_{best} \leftarrow$ tour$_{Ant_k}$
            cost$_{best} \leftarrow$ cost$_{Ant_k}$
        for City$^i \in$ tour$_{Ant_k}$ do
            for City$^j \in$ tour$_{Ant_k}$ do
                $\tau_{ij}$ = (1-$\epsilon$)*$\tau_{ij}$ + $\epsilon\tau_0$
                PheromoneMatrix $\leftarrow \tau_{ij}$
    for City$^i \in$ Cities do
        for City$^j \in$ Cities do
            $\tau_{ij}$ =(1-$\rho$)$\tau_{ij}$
            PheromoneMatrix $\leftarrow \tau_{ij}$

    for City $\in$tour$_{bsf}$ do
        $\tau_{ij}$ += $\rho$ (1/computeCost(tour$_{bsf}$))
        PheromoneMatrix $\leftarrow \tau_{ij}$
return tour$_{best}$
```

B Elitist Ant System Pseudocode

Algorithm 2 Elitist Ant System

```
Input: Nodes, Ants$_{size}$, Iterations$_{size}$, $\alpha$, $\beta$, $\rho$, $e$
Output: tour$_{best}$
tour$_{best}$ $\leftarrow$ initializeRandomTour(Cities)
cost$_{best}$ $\leftarrow$ computeTourCost(tour$_{best}$)
initialPheromone $\leftarrow$ 1 / numCities * cost$_{best}$
PheromoneMatrix $\leftarrow$ initializePheromoneMatrix(initialPheromone)
while !StopCondition(currIteration , currTimeElapsed , currError) do
    for Ant$_k$ $\in$ Ants$_{size}$ do
        while length(tour$_{Ant_k}$) != length(Cities) do
            choose city$_i$ $\in$ Cities with $P = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}$
            tour$_{Ant_k}$ $\leftarrow$ city$_i$
        cost$_{Ant_k}$ $\leftarrow$ computeTourCost(tour$_{Ant_k}$)
        if cost$_{Ant_k}$ < cost$_{best}$ then
            tour$_{best}$ $\leftarrow$ tour$_{Ant_k}$
            cost$_{best}$ $\leftarrow$ cost$_{Ant_k}$
        for city$_i$ $\in$ tour$_{Ant_k}$ do
            secondCity $\leftarrow$ city$_i$
            firstCity $\leftarrow$ city$_{i+1}$
            $\tau_{i(i+1)}$ += 1 / distance(secondCity , firstCity)
            legMatrix $\leftarrow$ $\tau_{ij}$
    for x , y $\in$ pheromoneMatrix do
        $\tau_{xy}$ += (1-$\rho$) * $\tau_{xy}$ + legMatrix[x][y]
elitePheromone $\leftarrow$ $e$*(1 / computeCost(tour$_{best}$))
    for city$_i$ $\in$ tour$_{best}$ do
        secondCity $\leftarrow$ city$_i$
        firstCity $\leftarrow$ city$_{i+1}$
        $\tau_{i(i+1)}$ += elitePheromone
        pheromoneMatrix $\leftarrow$ $\tau_{ij}$
return tour$_{best}$
```

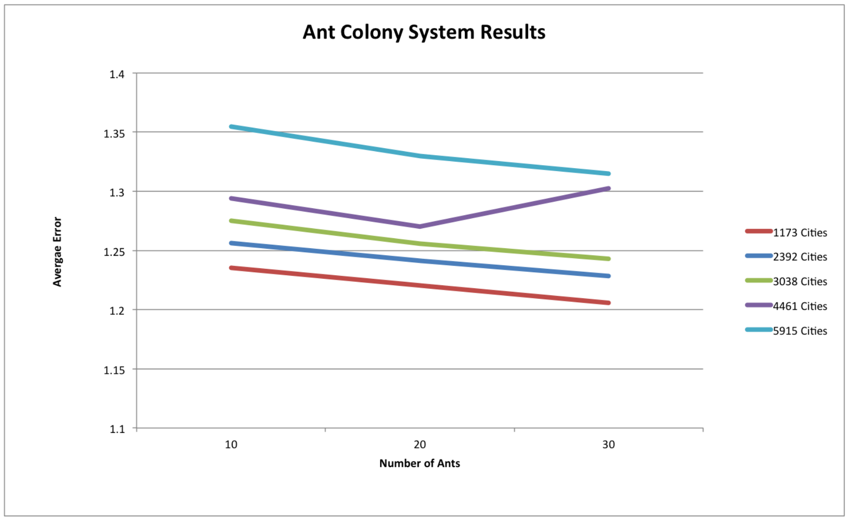
C Ant Colony System Data

Problem Size	Parameter	Test Number	Average
1173 Cities	10 Ants	1.23518245	1.23548993
		1.22343759	
		1.24784976	
	20 Ants	1.22094494	1.22068903
		1.21983472	
		1.22128745	
	30 Ants	1.20461576	1.20573134
		1.20214537	
		1.21043289	
2392 Cities	10 Ants	1.25639468	1.25976936
		1.26347556	
		1.25943786	
	20 Ants	1.23636785	1.24119347
		1.23938678	
		1.24782579	
	30Ants	1.22368753	1.22817981
		1.23139557	
		1.22945634	
3038 Cities	10 Ants	1.27497284	1.27508573
		1.27192763	
		1.27835672	
	20 Ants	1.25358724	1.25575218
		1.26347543	
		1.25019387	
	30 Ants	1.24087837	1.24297095
		1.24829628	
		1.23973821	
4461 Cities	10 Ants	1.29182731	1.29410128
		1.28918271	
		1.30129384	
	20 Ants	1.27126731	1.2702142
		1.27019283	
		1.26918271	
	30 Ants	1.30271628	1.3024536
		1.31281738	
		1.29182731	
5915 Cities	10 Ants	1.35827392	1.35458283
		1.36253728	
		1.34293729	
	20 Ants	1.34524624	1.3296802
		1.32456321	
		1.31923143	
	30 Ants	1.31678543	1.31491542
		1.30916547	
		1.31879538	

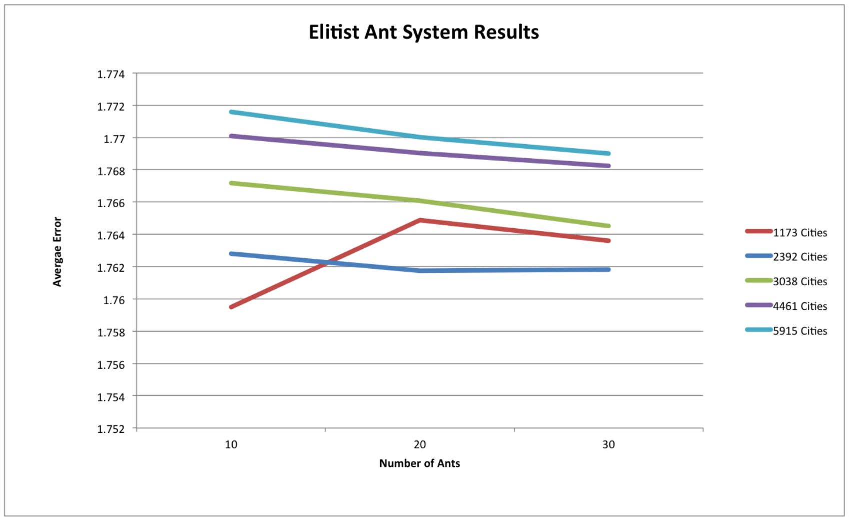
D Elitist Ant System

Problem Size	Parameter	Test Number	Average
1173 Cities	10 Ants	1.75838431	1.75949531
		1.75918271	
		1.76091891	
	20 Ants	1.76437811	1.7648712
		1.76531827	
		1.76491726	
	30 Ants	1.76371018	1.7635996
		1.76437263	
		1.76271627	
2392 Cities	10 Ants	1.76251623	1.7628179
		1.76291827	
		1.76301928	
	20 Ants	1.76128172	1.7617397
		1.76191827	
		1.76201921	
	30 Ants	1.76200192	1.7618154
		1.76151627	
		1.76192825	
3038 Cities	10 Ants	1.76918273	1.767169
		1.76516272	
		1.76716251	
	20 Ants	1.76516253	1.7660699
		1.76591726	
		1.76712817	
	30 Ants	1.76517261	1.7645025
		1.76416253	
		1.76417267	
4461 Cities	10 Ants	1.77019285	1.7700983
		1.77019182	
		1.76991029	
	20 Ants	1.76918273	1.7690275
		1.76891822	
		1.76898172	
	30 Ants	1.76890193	1.7682407
		1.76790192	
		1.76791827	
5915 Cities	10 Ants	1.77191827	1.7715882
		1.77091821	
		1.77192813	
	20 Ants	1.77019282	1.7700039
		1.77000182	
		1.76981723	
	30 Ants	1.76901928	1.7690071
		1.76899817	
		1.76900381	

E Ant Colony System Data Graphical Representation



F Elitist Ant System Data Graphical Representation



References

[1] Dorigo, Marco, and Luca Maria Gambardella. ”Ant colony system: a cooperative learning approach to the traveling salesman problem.” IEEE Transactions on evolutionary computation 1.1 (1997): 53-66.

[2] Hoffman, Karla L., Manfred Padberg, and Giovanni Rinaldi. ”Traveling salesman problem.” Encyclopedia of operations research and management science. Springer US, 2013. 1573-1578.

[3] Jason Brownlee. Clever Algorithms: Nature-Inspired Programming Recipes. January, 2011.

[4] Majercik, Stephen. “Ant Colony Optimization” Lecture, Bowdoin College, March 27 and 29, 2016