Application of Ant Colony Optimization for the Traveling Salesman Problem

Bo Bleckel[1], Jasper Houston[1], Dylan Parsons[1]
[1]*Bowdoin College, Brunswick, Maine, USA*
{*lbleckel, jhouston, dparsons*}*@bowdoin.edu*

Keywords: ACO, ant colony optimization, optimization, EAS, elitist ant system, ACS, ant colony system, TSP, traveling salesman problem

Abstract: Ant Colony Optimization (ACO) is an iterative algorithm that seeks to mimic the behavior of ants in a colony. In ACO, each ant seeks to build the best possible path given a certain set of criteria. In this case, ACO is applied to the Traveling Salesman Problem (TSP), where the goal is to create the shortest route that visits each city once and returns to the starting city. The ants create potential paths and lay down pheromone on those paths, choosing each successive city based on the level of pheromone already there as well as the distance to the city. In this paper, we explore two variants of ACO: Ant Colony System (ACS) and Elitist Ant System (EAS). Our goal is to first determine the optimal parameters for each ACO algorithm and then use said optimal settings to explore which of the two algorithms performs better on TSP problems.

1 INTRODUCTION

Ant Colony Optimization (ACO) is a swarm intelligence algorithm inspired by the behavior of ants. The general idea for the algorithm is specifically based on the process of ants exploring a space, looking for food. As the ants explore the space, they leave pheromones based on how good the food source that they find is. Those pheromones will evaporate if the source is less used, therefore meaning that it is a lower quality source. The model for ant behavior that is used in ACO algorithms is derived from experiments and verified by simulation. We will use ACO to solve the Traveling Salesman Problem.

The Traveling Salesman Problem is a problem that is considered NP-complete, meaning that there is not a known algorithm that can solve the problem in better than super-polynomial time. ACO provides a way to solve the problem, on average, faster than other approaches and faster than super-polynomial time. In this paper we will explore the performance of two ACO variations (Elitist Ant System and Ant Colony System) in solving non-trivial instances of the Traveling Salesman Problem. Both of these approaches solve the problem by constructing candidate solutions using a pheromone model, (i.e., a probability distribution over the solution space) and then using the candidate solutions to modify the pheromone values in a way intended to bias future sampling toward higher quality solutions.

We extensively tested our two algorithms on the same TSP problem while varying parameters in order to find which values were optimal for EAS and ACS respectively. Once we found these optimal parameter values, they were used to run the algorithms side by side on multiple files. In each case, ACS marginally outperformed EAS, and also converged on an optimal result in fewer iterations.

Section 2 describes the Traveling Salesman Problem, and Section 3 describes the algorithm and the two variants of it that we use. Section 4 details our experimental methodology in comparing the two methods, Section 5 enumerates our results, Section 6 proposes potential further work that could be done, and, finally, Section 7 explains our conclusions.

2 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is set up as follows: given a set of cities and distances between each city, find the shortest path that visits each city once and returns to the origin city.

Effectively, the set of cities can be thought of as a weighted graph. In a weighted graph, there is a "cost" associated with moving from one node to an-

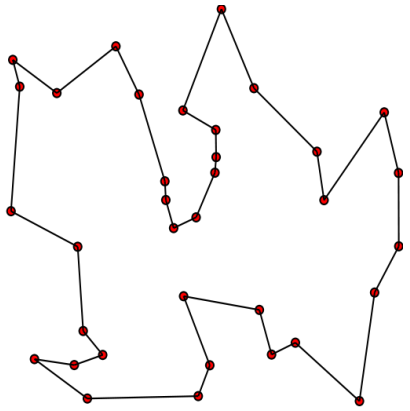

Figure 1: A sample solution to the Traveling Salesman Problem for the given set of cities.
Credit: https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/GLPK_solution_of_a_travelling_salesman_problem.svg/512px-GLPK_solution_of_a_travelling_salesman_problem.svg.png

other note. We can also think of TSP as a Hamilton Circuit – that is, a circuit that uses every vertex or node of a graph only once. Thus, the traveling salesman problem is finding a minimum weight Hamilton Circuit. Because we are looking to minimize cost, the Traveling Salesman Problem is an optimization problem that can be solved using optimization techniques such as Genetic Algorithms and Particle Swarm Optimization. In this case, however, we will be implementing an Ant Colony Optimization approach.

3 ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a swarm intelligence method for solving path-related problems, and is based upon the modeled behavior of ants. The key to the ants' behavior (and hence the algorithm) is the use of 'pheromones' a heuristic. Real ants release pheromones as they travel to find food, attracting other ants and eventually creating an effective path to the food source. Naturally, this pheromone trail evaporates without "foot traffic" from other ants. As the trail evaporates, it appeals less to other ants. ACO favors short paths by using pheromones: the longer a path is, the more time there is for the pheromone trail to evaporate as the ants traverse it, while shorter paths are more likely to grow in strength and appeal.

It is important to emphasize that the pheromone trail is stochastic; that is, if an ant encounters a pheromone trail (or several), it is not guaranteed to choose either the strongest pheromone trail or a pheromone trail at all (it could choose to continue along a random path). To achieve a balance between exploration and exploitation, an ant chooses a path based on the following probability:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{i \in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}, \tag{1}$$

where:

- τ_{ij} is the pheromone level on leg from i to j
- $\eta_{ij} = \frac{1}{d_{ij}}$, where $d_i j$ is the distance from i to j
- α, β are positive constants
- N_k is the set of cities not yet visited by ant k

In this way, each ant chooses a path based primarily on distance and available pheromones. Changing the factors α and β change the priorities of each ant; for example, choosing a very high α would reduce exploration by almost guaranteeing that ants will make their decisions based on pheromones.

3.1 Ant Colony System

Ant Colony System (ACS) was proposed in 1996 by Dorigo et al. and applies the solution construction described above. However, it doesn't always use this construction. With some probability Q_0, typically 0.9, ant k deterministically chooses the next city j to maximize $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$, the numerator in the calculation of p_{ij}^{k}.

The pheromone update in ACS is twofold. While the ants are building their tours, they wear away the pheromone on the legs they traverse as well as deposit a small amount. The new pheromone level on the leg is calculated as follows:

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

where:

- τ_{ij} is the pheromone level on leg from i to j
- $0.0 < \varepsilon < 1.0$
- N_k = cities not yet visited by ant k

In addition to the local pheromone update on legs being traversed by the ants, there is a global pheromone update each iteration after the ants have built their tours. The global pheromone update uses the following update rule for each leg:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bsf}$$

where:

$$\Delta\tau_{ij}^{bsf} = \begin{cases} \frac{1}{L^{bsf}} & \text{if } (i,j) \text{ is in the best so far} \\ 0 & \text{otherwise} \end{cases}$$

- τ_{ij} is the pheromone level on leg from i to j
- ρ is the evaporation rate, so $\rho\%$ of the pheromone on a leg evaporates leaving $(1-\rho)$ behind.

3.2 Elitist Ant System

Elitist Ant System (EAS) is a modified version of the original Ant System in which legs in the best route found so far are more likely to be chosen by the ant. In order to do this, a number of different tweaks can be made. One option is depositing more pheromone on the legs in the best-so-far route. Additionally, one could possibly increase α if $\tau > 1.0$. $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ can also be weighted to favor better paths. It is also a possibility to decrease evaporation rate on legs in the best-so-far path or to adjust the heuristic information to take into account the best-so-far. However, the only one of these options for elitism that we implemented was depositing more pheromone on the legs in the best-so-far path. The pheromone level for a leg is calculated as follows:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \Delta_{ij}^{total} + e\Delta\tau_{ij}^{best}$$

where:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L^{bsf}} & \text{if } (i,j) \text{ is in the best so far} \\ 0 & \text{otherwise} \end{cases}$$

- τ_{ij} is the pheromone level on leg from i to j
- L^{bsf} is the length of the best so far tour
- e is the elitism factor

With the elitism factor, it is a good rule of thumb to set e to be the number of ants. In our implementation, this is done automatically. All other parameters can be adjusted in the ACO.h file.

4 EXPERIMENTAL METHODOLOGY

When it comes to testing, ACO resides somewhere between the relatively low configurability of Particle Swarm Optimization and the stream of parameters available in Genetic Algorithms. Our testing focused on the configuration and combination of $\alpha, \beta, \varepsilon, \rho$, and Q_0. Each parameter plays a different role in making a given configuration unique. α and β balance the priority of each and between pheromones and distance, respectively. ε and ρ affect pheromone changes (deposit and evaporation) on legs. Q_0 determines the greediness of each ant: the higher it is, the more likely it is that each ant will choose a new leg based solely on the best combination of pheromones and distance. This process is described below. In order to test the effects of each parameter, we ran several tests on $d2103.tsp$, varying one parameter at a time. The default value of each parameter is:

- $\alpha = 1.6$
- $\beta = 5$
- $\varepsilon = 0.1$
- $\rho = 0.1$
- $Q_0 = 0.9$

These parameters were used to determine the optimal value for each parameter, which were then used when testing larger problem sizes. These optimal values are described in Section 5.

4.1 ACO Algorithms

There are many variations of algorithms for various ACO configurations; in this section, we describe, at a high level, the algorithms we used.

4.1.1 Choosing a City

A critical aspect of ACO is determining how an ant will choose which cities to place in its tour. We used two methods for choosing cities: greedily and probabilistically. The parameter Q_0 is responsible for determining the likelihood that an ant chooses one over the other. Higher Q_0 corresponds to more greedy choices, and lower Q_0 corresponds to more probabilistic choices.

4.1.2 Greedy City Selection

The greedy method for city selection chooses the city with the best 'value', determined by the formula below (note that this is the numerator from equation 1, which determines the probability of selecting a city).

```
BEGIN
    for each unvisited city
    city strength =
                τ_ij^α η_ij^β
        if strength > maxStrength:
             city = maxCity
   end for
   return maxCity
END
```

Figure 2: Pseudocode for greedy city selection.

4.1.3 Probabilistic City Selection

Alternatively, an ant may also choose a city probabilistically. In this way, an ant randomly chooses from unvisited cities, stopping once one has been chosen. The probability of choosing a city is based on equation 1.

```
BEGIN
  while true
    choose random city from unvisited cities
    probability =
```

$$\frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{i \in N_k} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}},$$

```
    p = random double, 0 <= p <= 1
    if p < probability :
      return city
  end while
END
```

Figure 3: Pseudocode for probabilistic city selection.

4.2 Solution

In order to 'solve' optimization problems, all the methods we've described come into play.

```
BEGIN
     while(!terminated):
          build new tour for each ant
          update best-so-far route
          update pheromones
          reset individual ant bests
     end while
END
```

Figure 4: Pseudocode for ACO solution.

Of course, the method of pheromone updating depends on whether we're solving ACS or EAS. The formulae for each of these are described in Sections 3.1 and 3.2. The termination condition can be one of many; while we always terminated upon reaching maximum iterations, we also used an optimality cutoff: terminate if the best-so-far achieves a certain percentage (we used 0.01%) of the known best solution.

5 RESULTS

As noted in Section 4, we first determined the optimal parameters for the two algorithms. These parameter values were all mostly in-line with the rule-of-thumb values. One note is that we found that α was more effective when set to 0.5, rather than the rule-of-thumb value of 1. Otherwise, the optimal values that we found reside nicely within the rule-of-thumb ranges. We decided to fix the number of ants in our tests to 20 ants, and to explore the progress of the algorithms over 300 iterations. In Section 6 we talk a little about how something we wished we had explored, but did not have the time for, was limiting our algorithm to a specific running time, rather than a number of iterations. The experimental results that allowed us to make a decision on the default values for each parameter are shown in Figures 5 through 12. It is important to note that these plots show the total best distance after each iteration. In order to compare with the known optimal length (which is 80,450 for the D2103.TSP file), one would simply divide our algorithm's optimal by the known optimal length. In general, we were within 150% of—or 1.5 times—the optimal length. We then tested our algorithms on two files using the previously found default parameter settings, and those results are shown in Figures 13 and 14.

Testing was performed on a Late 2013 MacBook Pro with a 2.4GHz Intel Core i5 Processor and 8GB of 1600MHz DDR3 RAM.

5.1 Default Parameter Values

In order to optimize our algorithms for the Traveling Salesman Problem, we tried to fine-tune our algorithm parameters by testing the various parameters values on a medium sized problem. Specifically we used the D2103.TSP file. As explained in Section 4 we found optimal values for all three general parameters and the two specific parameters for ACS. While testing for a certain parameter's optimal value, we fixed the value of the others to a reasonable value given by the rule-of-thumb values. For example, when trying to find the optimal value for α, we set $\beta = 5$ and $\rho = 0.1$. We also limited our testing to just one file for time's sake. Undoubtedly, this limited our understanding of how each of the parameters interacted with each other, but did allow us to feasibly run our tests; had we tried every possible parameter combination, we would have needed a much more powerful computer and weeks more for testing. As this was more of a comparison of the EAS and ACS algorithms, and not necessarily an exploration of their parameter space, we decided that this was an acceptable omission in our methodology. We did not run any tests on the number of ants, or e the elitism factor for the EAS algorithm. As mentioned in Section 6, we would include those tests in further iterations of these experiments. For Ant Colony System, we found that the following parameters gave the best results:

- $\alpha = 0.5$
- $\beta = 4$
- $\varepsilon = 0.01$

- $\rho = 0.25$
- $Q_0 = 0.9$

And for EAS:

- $\alpha = 0.5$
- $\beta = 4$
- $\rho = 0.1$
- note, ε and Q_0 are not parameters for EAS

The results plotted in Figures 5 through 7 show, among other things, ACS's responsiveness to parameter changes. Generally one parameter value seems to be dominant except in the case of β (Figure7 where it is clear that after 200 iterations all parameter values yield essentially the same result.

Figures 8 through 10 demonstrate EAS's responsiveness to parameter variation, and it is clear that EAS is more responsive than ACS. However, as shown in Figure 8, for β the story is generally as responsive as it was for ACS. Interestingly ACS and EAS behave very similarly when testing for β. That said, it is worth noting that the result is different: $\beta = 3$ is generally better for ACS, while $\beta = 5$ seems to outperform the others in the case of EAS.

Figures 11 and 12 show the effect of ε and Q_0 on ACS. We can conclude that ACS is more responsive to changes in ε and Q_0 than it was to changes in the other parameters. We would like to point out that we ran ACS with $Q_0 = 1.0$, and the results are shown in Figure 12. When $Q_0 = 1.0$ ACS no longer acts how it was intended to act. Thus we did not use that value as our optimal value for Q_0, and instead used the next best which is 0.9.

5.2 Performance

The results show that given optimal parameters, ACS performs better than EAS in most cases. That is to say that over 4 trials, on the median trial ACS reached it's optimal length earlier than EAS did on the median trial. We ran both algorithms on the D2103.TSP file. We used 20 ants, we again ran for 300 iterations, and used our optimal values found earlier for α, β, ρ, and q_0 and ε when running ACS (EAS only takes α, β, and ρ). The results show that ACS makes progress towards the optimal value much sooner than EAS does, at least in terms of iterations. Again, we will talk more about how timing the algorithm could provide insightful knowledge into which algorithm finds the better answer quicker in terms of time.

After 300 iterations, both algorithms came within approximately 110% of the known optimal length for the smaller D2103.TSP problem, as shown in Figure 13. This figure illustrates ACS's tendency to converge to the optimal length quicker than EAS by measure of iterations. On the larger PCB3038.TSP file, Figure 14 shows how again ACS reached the optimal much sooner, but was generally only a little better than EAS was over 300 iterations. In addition, the trend of EAS shown in Figure 14 may suggest that the algorithm still had a chance of getting a better answer if more iterations were allowed. On the larger problem EAS came within 125% of the known optimal answer, and ACS came within 124% of the optimal answer. It is worth noting that these results shown in Figure 13 and 14 are the median best values (from the last iteration) out of four independent trials of each algorithm on each file.

Interestingly, Figure 14 shows that both ACS and EAS made large initial steps then plateaued for a while. The difference between the two algorithms here seems to be that ACS plateaued for fewer iterations, thus enabling the algorithm to make greater progress earlier on. EAS appears to plateau for around 170 iterations (exactly 172 according to the data). This may be explained by the nature of Elitist ant system, in that it gives much more weight to the ants whom have found better tours on the previous iteration. If those tours are wrong, but better than every other ant, it might be that it takes a while for the algorithm to correct for that answer, thus allowing the algorithm to stagnate for many iterations. This point is also demonstrated in Figure 13, where EAS hits a plateau for 137 iterations before making very significant breakthroughs. It is also worth noting the general range of the algorithms' best tour lengths are relatively small. On the smaller file the algorithm has a range of approximately 1.2 times the optimal to approximately 1.08 times the optimal.

As shown in Figure 14, which shows performance on a larger problem, we see that an increased number of iterations generally gives better results. Had we limited the number of iterations to 100, only ACS would have found the result. Unfortunately, we were not able to test the effect of the number of ants on larger problems, but it is reasonable to think that increasing or decreasing the number of ants significantly might have significant effects on the performance of the algorithms, especially when there are so many cities involved.

5.2.1 Larger Files

We tested our algorithms on four files; however, we have only included the data from two files in this report. That is due to the fact that we were not able to get any results in a reasonable amount of time on the larger files. We tried on the FNL4461.TSP file (4,461

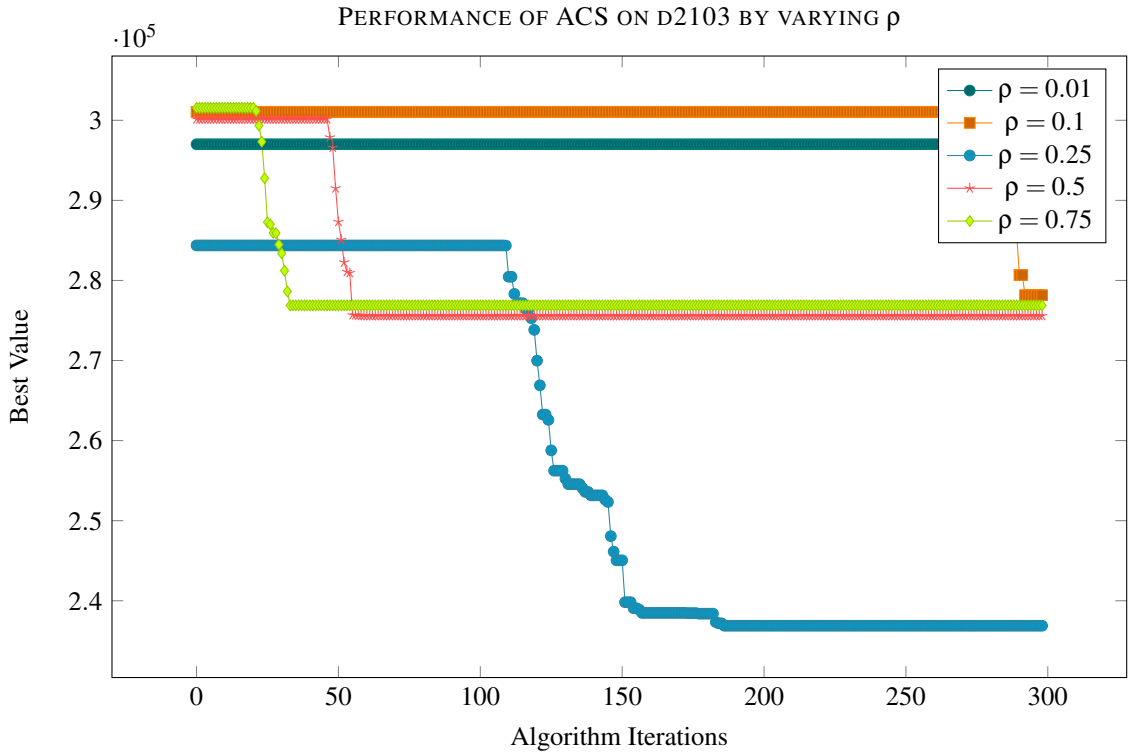

Figure 5: ACS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.6$, $\beta = 5$ and varying values for ρ. This file's optimal distance is 80,450.

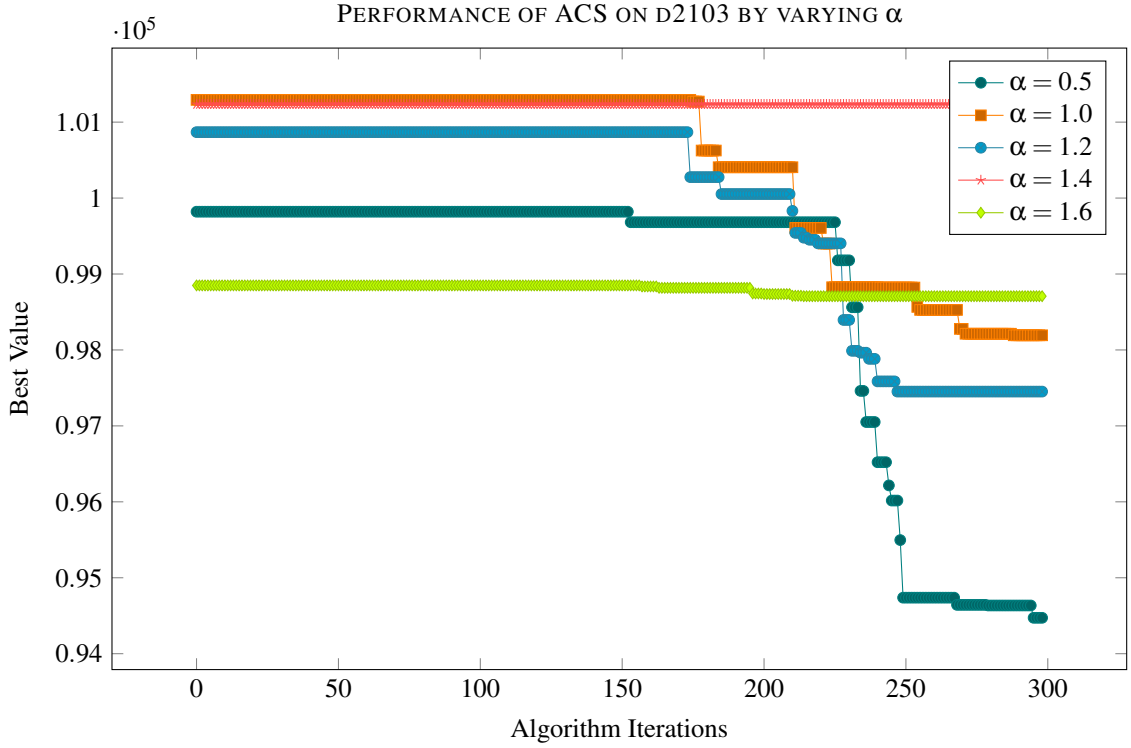

Figure 6: ACS run on the d2103.tsp file with 20 ants, 300 iterations, $\rho = 0.1$, $\beta = 5$ and varying values for α. This file's optimal distance is 80,450.

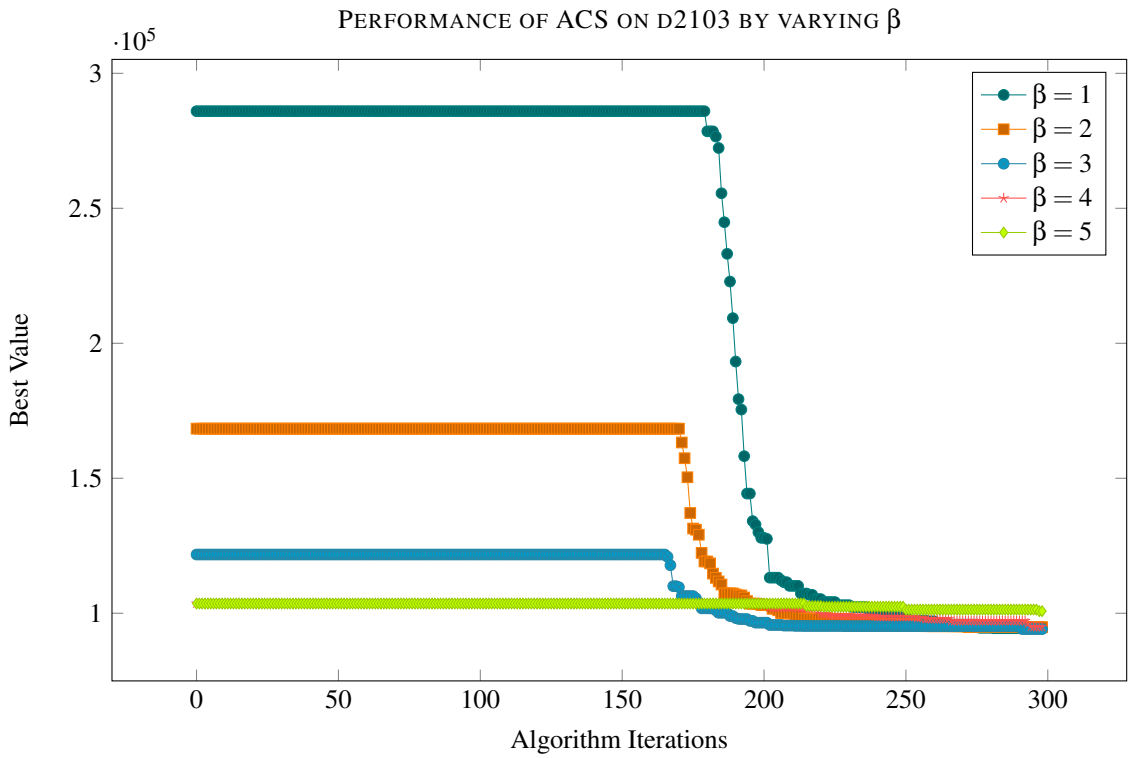

Figure 7: ACS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.6$, $\rho = 0.1$ and varying values for β. This file's optimal distance is 80,450.

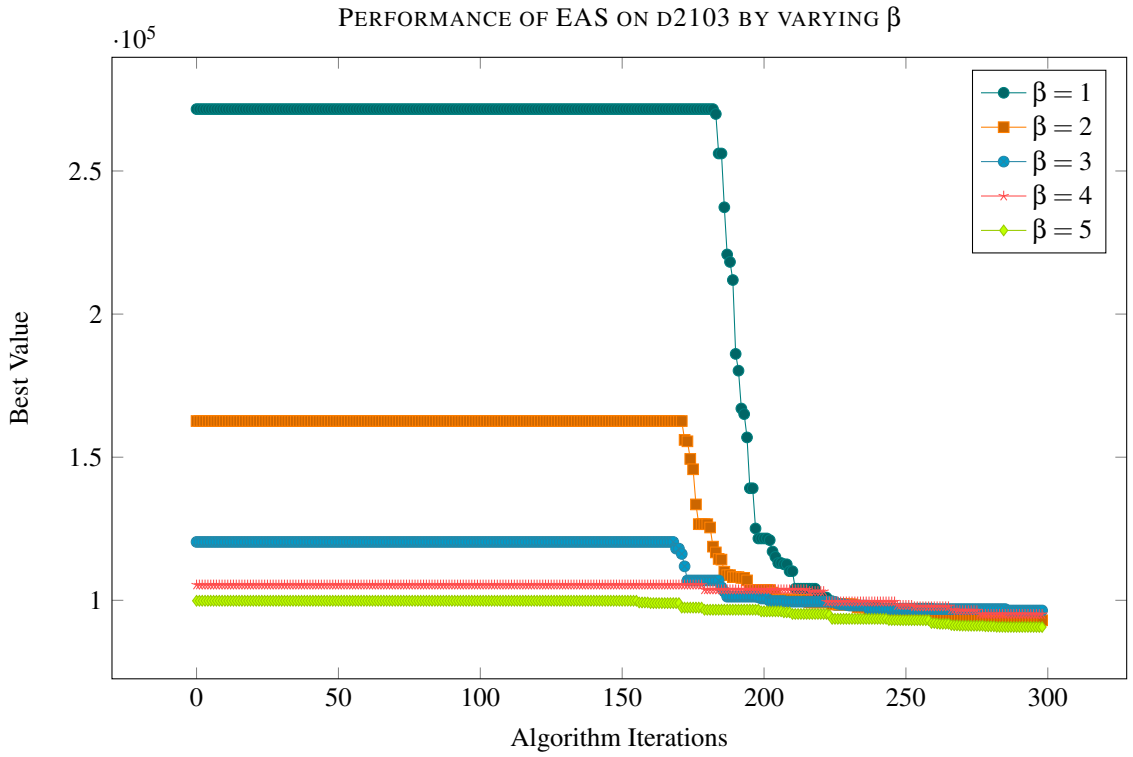

Figure 8: EAS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.6$, $\rho = 0.1$ and varying values for β. This file's optimal distance is 80,450.

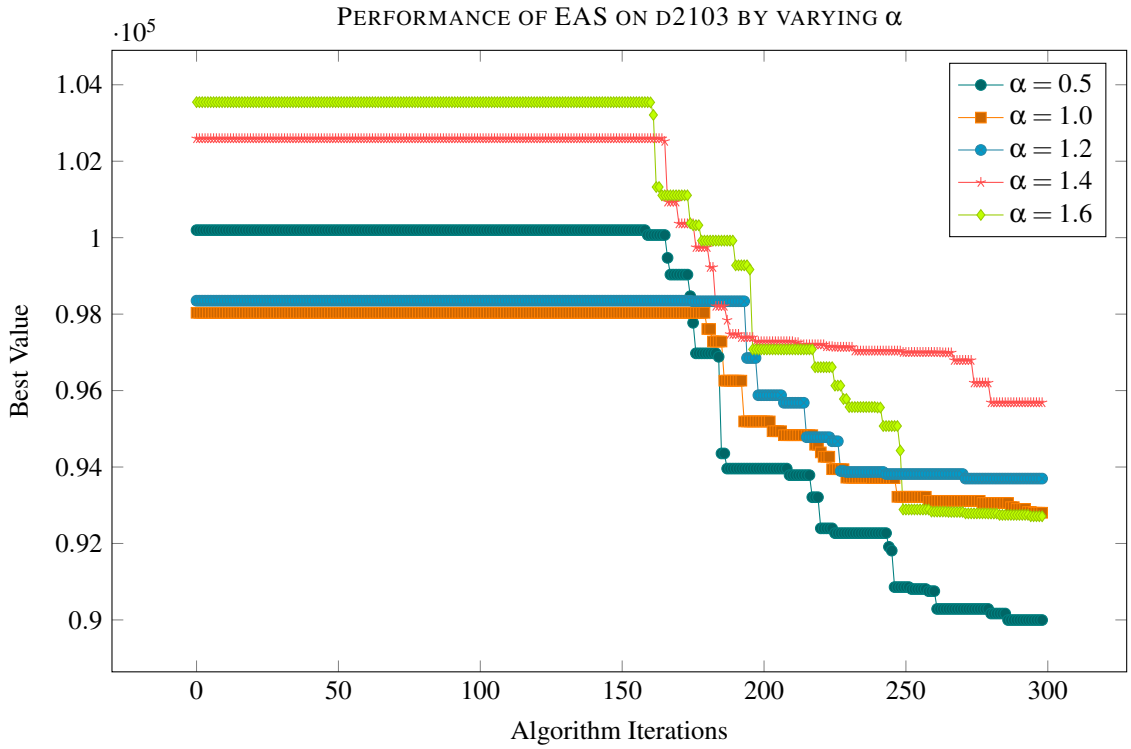

Figure 9: EAS run on the d2103.tsp file with 20 ants, 300 iterations, $\rho = 0.1$, $\beta = 5$ and varying values for α. This file's optimal distance is 80,450.

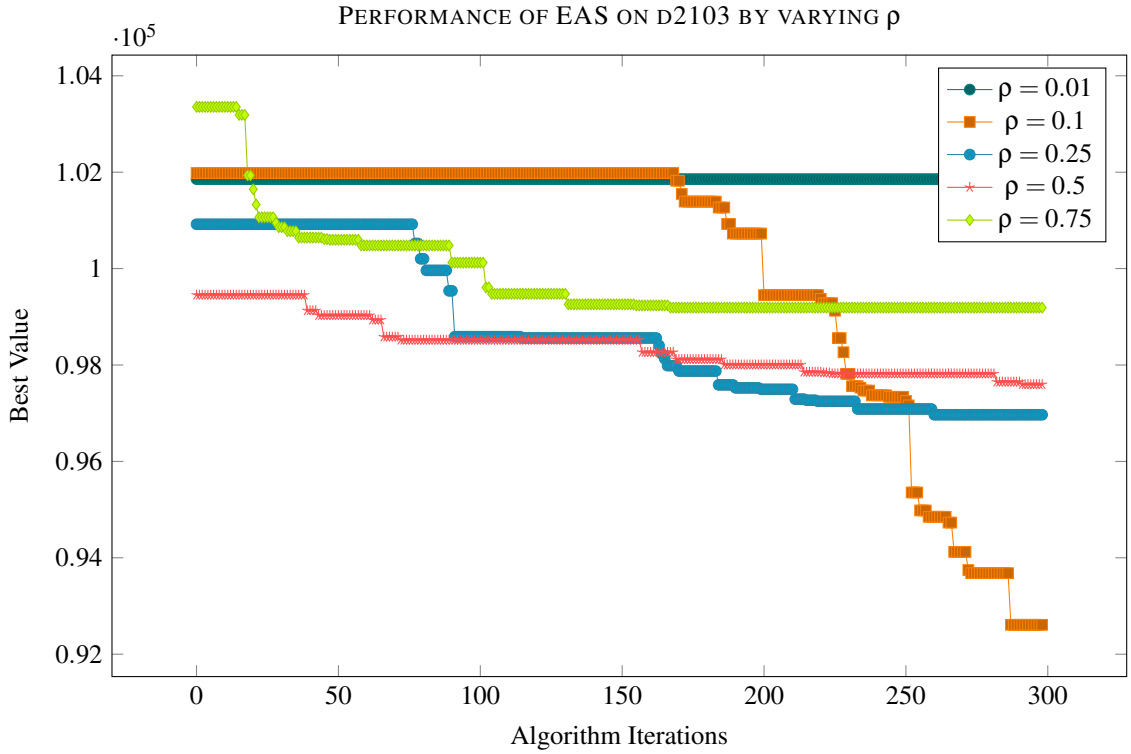

Figure 10: EAS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.6$, $\beta = 5$ and varying values for ρ. This file's optimal distance is 80,450.

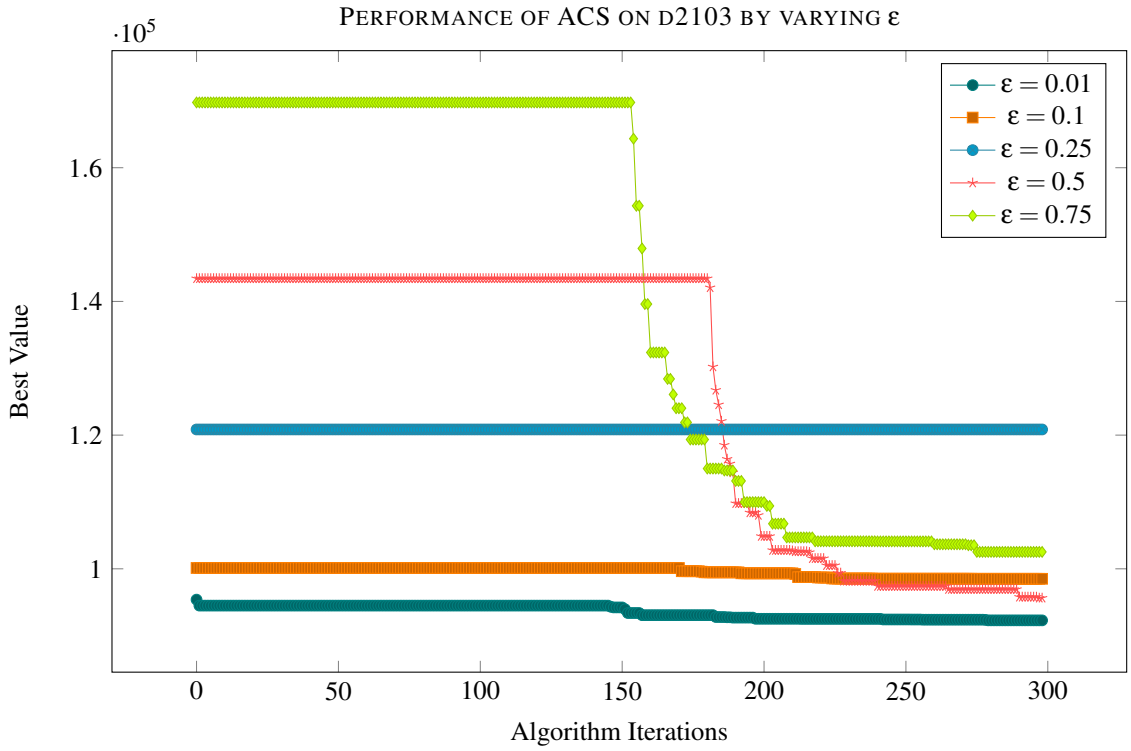

Figure 11: ACS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.2$, $\beta = 5$, $\rho = 0.1$, and varying values for ε. This file's optimal distance is 80,450.

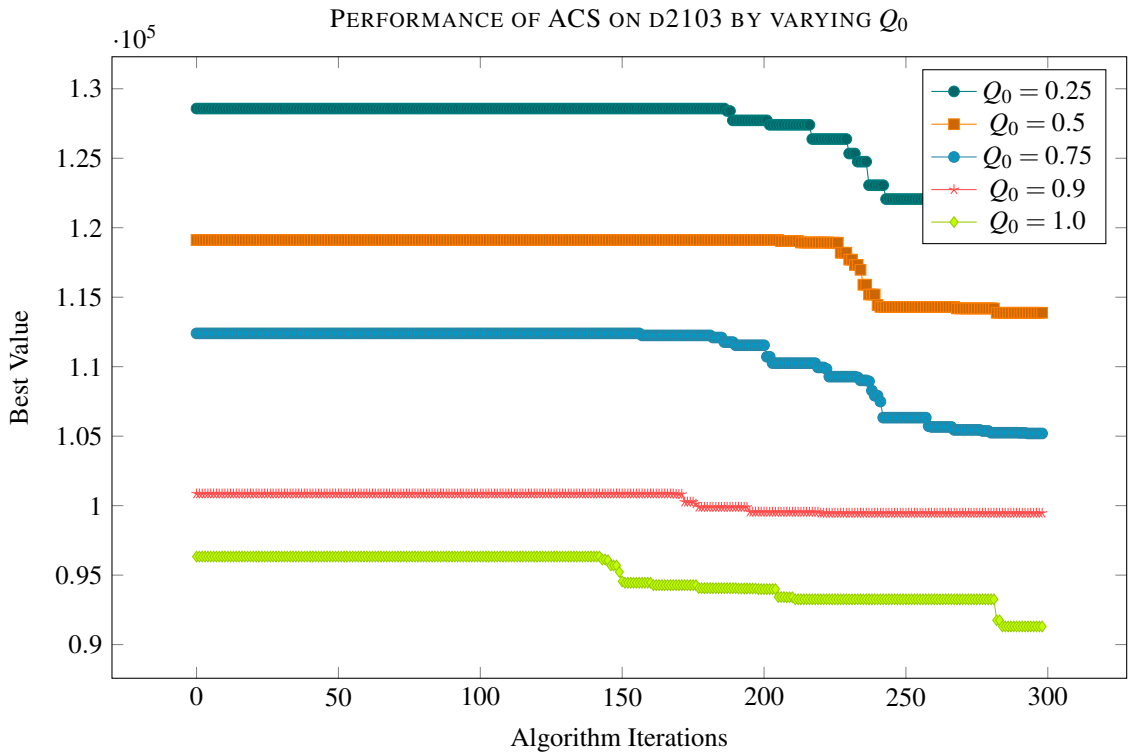

Figure 12: ACS run on the d2103.tsp file with 20 ants, 300 iterations, $\alpha = 1.2$, $\beta = 5$, $\rho = 0.1$, $\varepsilon = 0.1$, and varying values for Q_0. This file's optimal distance is 80,450.

cities) and the RL5915.TSP file (5,915 cities). We ran them for hours, but were unable to get past a few iterations for both EAS and ACS. It is not clear whether this is an issue with our code in handling large problems, or simply a fact of the matter for this size problem. We will note that our code does seem to be running well on the slightly smaller problems, and only ran into issues with over 4,000 cities. This leads us to believe that it is simply the nature of these larger problems that is limiting our results, not any imperfections in our code.

6 FURTHER WORK

Throughout our exploration of Ant Colony Optimization and its application to the Traveling Salesman Problem, there were a number of items that we wished we had more time to explore.

6.1 Further Testing

In our testing process, there are a number of things we wished we had more of an opportunity to test.

6.1.1 Larger TSP Files

Unfortunately, due to lack of time in our testing process, we were unable to test on some of the larger files. This would simply be resolved by leaving tests running for a few more days (or weeks).

6.1.2 Multiple Trials

In addition to not being able to test on the larger files, we were unfortunately unable to run multiple trials for the same file. This meant we were unable to take a median over all the trials to show in our results. Taking a median is typically a good idea because it eliminates the impact of outliers. There is a possibility that the tests that we ran were outliers. However, we do not suspect that this is the case, as results were very consistent on smaller files with less than 500 cities, and as such we have no reason to suspect that the trials run on the larger files in the 2000-3000 city range would be outliers.

6.1.3 Additional Parameters

There are a few parameters that we did not change through all our testing. We left the number of ants constant at 20, the number of iterations constant at 300, and the elitism factor for EAS constant as the same as the number of ants. Given opportunity for additional work in this subject, we would like to truly find an optimal number of ants for each of the algorithms as well as determine whether having the elitism factor be the same as the number of ants is truly optimal. With the number of iterations, results would likely not change, as it seemed in many of our tests that the most change happened by 200 or 250 iterations, but it would nevertheless be good to be sure that 300 is sufficient through testing.

6.1.4 Timing

We tested solely based off iterations. However, we noted during our testing that ACS was reliably faster than EAS. This made us wonder how significant the speed difference might be. There are two elements of timing that would be interesting to implement. First, we would compare how long it takes to do a given number of iterations on the same file for EAS and ACS. It would then be interesting to set each algorithm to time how long it takes to find a solution within a certain percent of the optimal solution and compare in this way.

6.2 Other ACO Variations

In addition to EAS and ACS, there are other variations of ACO that have been conceived over time that we did not look at. In particular, we are very interested in the original Ant System. We have been led to believe that the newer ACO algorithms such as the ones we tested are significantly faster and better at finding a shorter path, but we are curious about how significant this difference is. In addition to this, we discussed MIN-MAX Ant System, which strongly manipulates the best tour so far and manipulates pheromones in order to do so. Other potential ACO algorithms to implement and test would be Rank Based Ant System, Continuous Orthogonal Ant Colony, and Recursive Ant Colony Optimization.

6.3 Explore Q_0

In our testing, we found that setting $Q_0 = 1$ was optimal for ACS. However, this meant that the ants would always choose the next city using the greedy method of maximizing the numerator in the selection equation. Had we used $Q_0 = 1$ as the optimal value for ACS, it would have effectively made it not be ACS any longer because there would be no chance of probabilistic picking of a next city, just the greedy method. Thus, we used the next best value, $Q_0 = 0.9$. However, we were certainly intrigued by this finding and are interested in exploring why it might be the case.

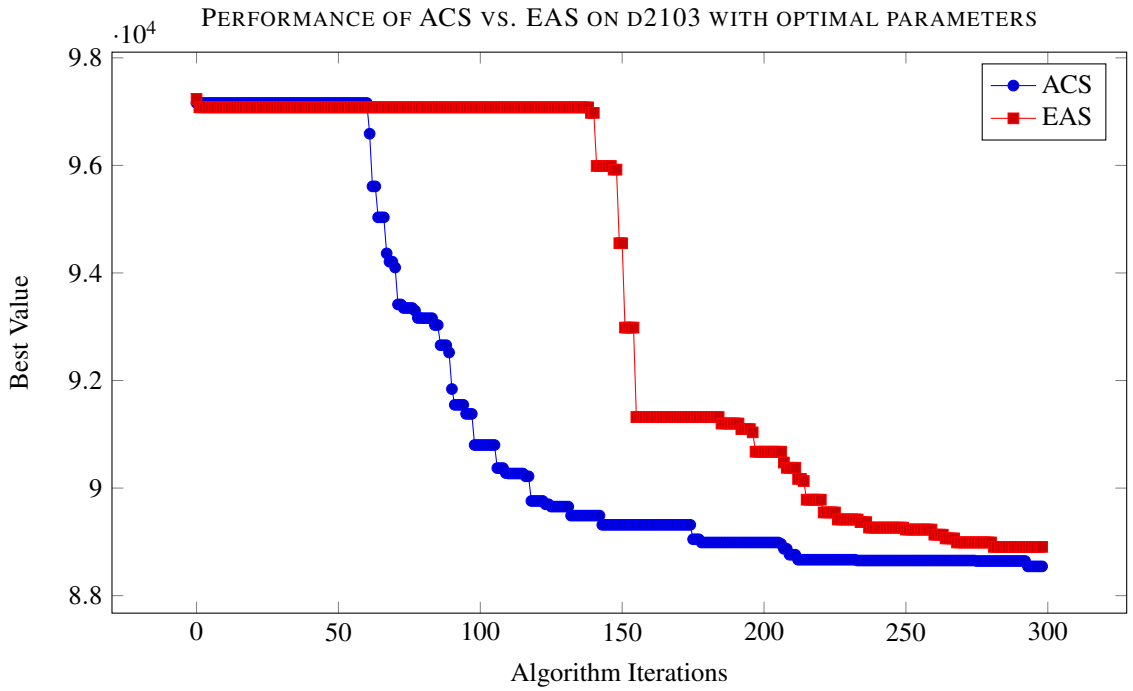

Figure 13: ACS run on the d2103.tsp file with optimal parameters of 20 ants, 300 iterations, $\alpha = 0.5$, $\beta = 4$, $\rho = 0.1$ for EAS and $\rho = 0.25$ for ACS, $\varepsilon = 0.01$, and $Q_0 = 0.9$. This file's optimal distance is 80,450. EAS found a path of distance 88,907.3 (110.5% of the optimal) and ACS found a path of distance 88,543.9 (110% of the optimal).

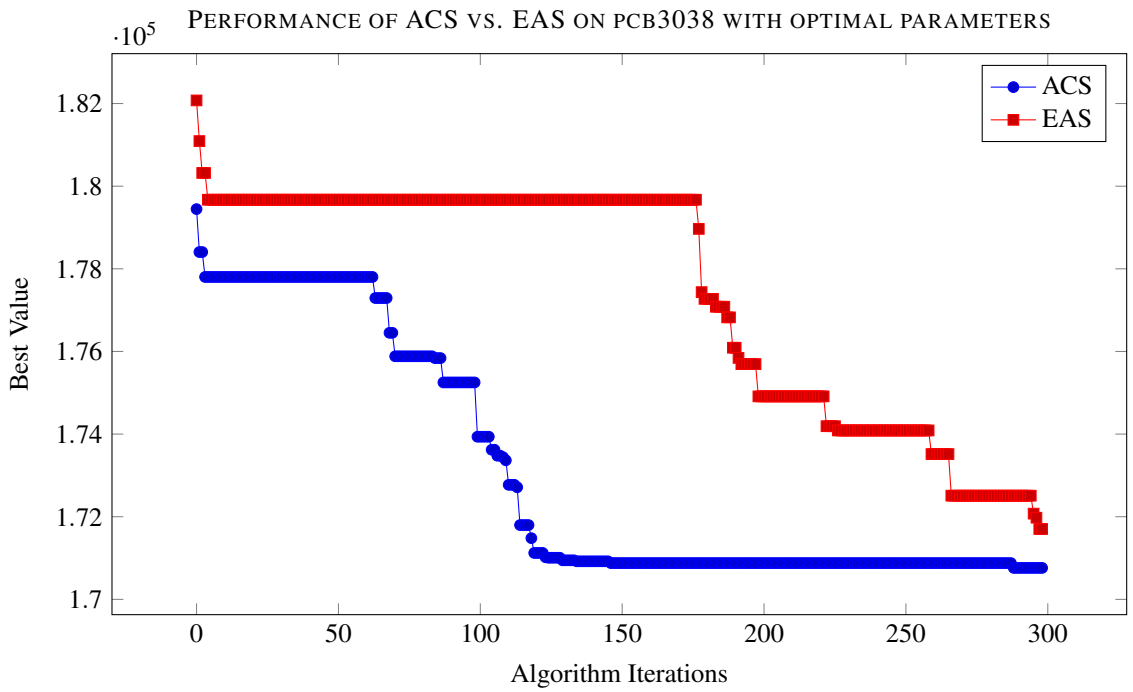

Figure 14: ACS run on the pcb3038.tsp file with optimal parameters of 20 ants, 300 iterations, $\alpha = 0.5$, $\beta = 4$, $\rho = 0.1$ for EAS and $\rho = 0.25$ for ACS, $\varepsilon = 0.01$, and $Q_0 = 0.9$. This file's optimal distance is 137,694. EAS found a path of distance 171,703 (124.7% of the optimal) and ACS found a path of distance 170,761 (124% of the optimal).

6.4 Parallel Programming

A possible extension of ACO could be exploring the possibility of running it in parallel using a master-slave implementation. Each slave thread would be a single ant, thus splitting up the route calculations for each ant among a number of different cores. The master thread would perform the global updates at the end of each iteration and manage the slave threads. This could be implemented on the Bowdoin High Performance Cluster and would likely result in significant performance benefits.

7 CONCLUSIONS

By extensively testing parameters and selecting those that functioned best for our two ACO algorithms, we were able to test the effectiveness of EAS as compared to ACS. Between the two algorithms, we found Ant Colony System to consistently outperform Elitist Ant System, but not by a significant margin. In each of our two tests we were able to complete with optimal parameters, there was a difference of less than one percent of the optimal path length. However, by qualitative analysis of the results graphs, we are able to see that ACS took fewer iterations to find a better response. Had we limited the testing to 150 or 200 iterations, there would have been a significant difference favoring ACS. This being said, ACS and EAS have both proven themselves to be highly effective in finding a good (albeit not perfect) solution for examples of the traveling salesman problem, and looking into how it might stand up to other optimization techniques such as Particle Swarm Optimization, for this application or others, could be very interesting.