Ant Colony Optimization: An Advanced Approach to the Traveling Salesman Problem

Bronwen May, Elizabeth Greer, Griffin Holt, Kenneth Vargas

C S 312: Algorithm Design & Analysis - Sec. 001 - Team 17
Brigham Young University
Provo, UT, USA

Abstract

The Traveling Salesman Problem (TSP) is a widely studied computational problem. This paper walks through the process of designing and implementing an Ant Colony Optimization (ACO) algorithm to solve a TSP. We discuss the natural phenomenon behind ACO and how this is turned into an algorithm via pseudocode. The process of parameter selection, including a section in which each of the five variables α, β, ρ, k, and Q, is highlighted with tests showing that these variables will produce locally optimal results (as far as our limited testing proves) when the cost function $c = f(\alpha, \beta, \rho, k, Q)$ is set to $c = f(.8, 2, .2, 100, 1000)$. We compare ACO to both a Greedy and Branch-and-Bound approach to solving TSP and find that for problem size $n \leq 15$, Branch-and-Bound performed slightly better than our ACO algorithm (82.47% of Greedy vs 85.62%). However, ACO performs significantly better ($\geq$ 5% better) than Branch-and-Bound in problem sizes approximately 20-150. After 150, the two algorithms begin to converge to the Greedy algorithms results. Time and space complexity of both Greedy and ACO algorithms are discussed with the time complexity being $O(n^3)$ and $O(wkn^2)$ respectively, where *n = number of cities*, *k = number of ants*, and *w = the rate of the ACO implementation's convergence*.

1. Introduction

The Ant Colony Optimization Algorithm (ACO), first published in 1996 by Marco Dorigo, is a nature-inspired, probabilistic approach used to solve computational and optimization problems that can be reduced to finding a lowest cost path through a graph. One such problem is the well-known Traveling Salesman Problem (TSP). The purpose of this report is to describe, first, how ACO is used as a technique to solve TSP; second, the pros and cons of the approach; and third, the results we achieve on problems of size 15-250 cities. Specifically, the Ant System variation–the first of all ACO algorithms and a variation designed specifically for TSP [5]–is the algorithm that will be discussed. We have previously implemented both a Greedy and Branch-and-Bound approach to solve TSP and hypothesized that the ACO algorithm would produce more optimal results in a shorter amount of time than both of these approaches to the exact same problems.

While the Greedy approach creates routes purely by looking at the cost of each edge, our ACO algorithm uses edge cost as well as feedback from previous tours to determine which path to take. This led us to believe that we would produce better results with ACO than with the Greedy algorithm. We will first describe our greedy algorithm to give a basis for later comparison. Branch-and-Bound pseudocode will not be included in this report.

2. Algorithm Explanation

2.1 How the Greedy Algorithm Works

The greedy algorithm we constructed is defined by the pseudocode below:

```
Procedure GreedyTSP(G):
      Input: A graph G of n cities
      Output: A complete tour T of the graph G, if such a tour can be found by the greedy algorithm;
              otherwise, NIL

      found_tours = [ ]

      For each city u in G:
          partial_path = [u]
          Repeat until a complete tour is found or a dead end is reached:
                connected_cities = all cities v such that the edge (u, v) exists and v is not already in partial_path

                If connected_cities is empty:
                    If the partial_path is a complete tour:
                        found_tours.append(partial_path)
                    Exit the 'Repeat' loop (A complete tour has been found or a dead end has been reached)
                Else:
                    v = the city with the minimum edge cost in connected_cities
                    partial_path.append(v)
                    cost(partial_path) += edge_cost(u, v)

                u = v

      if found_tours is not empty:
          return the complete tour T in found_tours with the minimum cost
      else:
          return NIL
```

2.1.1 Theoretical Time Complexity

The greedy implementation will have a worst-case time *and* space complexity of $O(n^3)$. The greedy algorithm works by using every node as the starting city. Then we find the cheapest path to travel from the current city. Finding the minimum cost at each node is an O(n) operation. Then we will use the node we traveled to as the current node and repeat the process until we reach a dead end or a complete solution. This will be repeated a maximum of O(n) times. After we calculate all the possible solutions, we will select the one with the cheapest cost.

Time and Space Complexity: $O(n^3)$

2.2 Deciding which Advanced Algorithmic Approach to Use

As a team, we considered three different advanced approaches to the Traveling Salesman Problem: Ant Colony Optimization, the *k*-opt heuristic, and the recently published approximation algorithm [1] that beat the previous record for the metric Traveling Salesman Problem. Ultimately, we decided to implement the Ant Colony Optimization algorithm because we found it fascinating that the algorithm simulates a real-world biological process to approximate a solution to the TSP.

Our primary resource for learning about Ant Colony Optimization algorithms was Wikipedia's entry on the topic [5]. As we were studying the concept, we discovered that there are many different types of ACO algorithms, including Ant System (AS), Ant Colony System (ACS), Elitist Ant System, Max-min Ant System (MMAS), Rank-based Ant System (ASrank), Continuous Orthogonal Ant Colony (COAC), and Recursive Ant Colony Optimization. We opted to implement the Ant System (AS) variation [2], it being the most basic form upon which the others are predicated.

2.3 How the Ant Colony Optimization Algorithm (ACO) Works

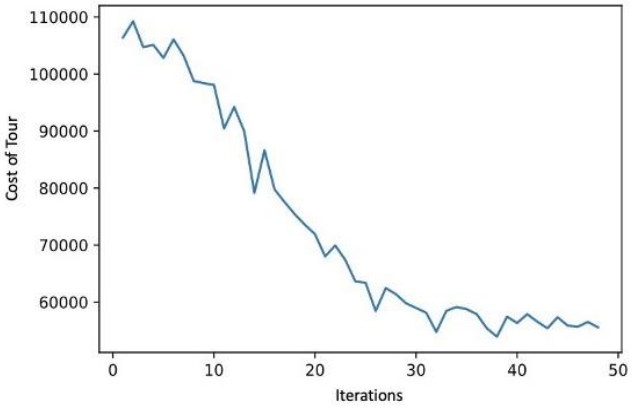
In the real-world, ant colonies send out forager ants [4] to travel to various locations for food, water, and other various resources. As each forager ant travels, it leaves behind pheromones that mark where it has traveled. In addition, forager ants tend to follow trails that already have some level of pheromone, since these paths are more likely to lead to the locations of already discovered resources. Over time, the shortest paths to resources acquire a greater level of pheromones and the ants collectively travel along those shortest paths. Ant Colony Optimization algorithms take advantage of this phenomenon by simulating the ant behavior of leaving behind pheromones on known paths.

The Ant System (AS) variation of the ACO algorithm is defined as follows [5]:

1. An "ant colony" of *k* ants is created and an arbitrary starting city is selected. (The starting city is arbitrary because a complete tour can start at any city.)
2. An initial best-solution-so-far is set to have a cost of infinity.
3. Until the algorithm terminates according to some *termination heuristic*:
    a. For each "ant" in the "ant colony":
        i. The ant starts at the starting city and begins to travel down a path, never visiting cities to which it has already been. When it comes to a city, it selects the next edge to travel down according to a probabilistic heuristic that is based on the amount of pheromone along that edge and the cost of the edge itself. This *edge selection heuristic* is defined further down. The ant stops traveling when it reaches a dead end (i.e., when there are no remaining edges ahead of the ant that lead to unvisited cities) or when a complete tour is found.
    b. For each complete tour that was found by the entire ant colony, pheromones are deposited on the edges of that tour according to a *pheromone deposit heuristic*, defined further down.
    c. Out of all the complete tours found by the ant colony in this iteration, the tour with the minimum cost is compared against the best-solution-so-far. If the minimum-cost tour has a lower cost than the best-solution-so-far, then the best-solution-so-far is updated to this minimum-cost tour.

Over the course of the execution of the algorithm, the ants–prompted by the pheromone levels that have been placed on previously successful paths–get closer and closer to finding the optimal tour in the graph. This process is demonstrated by the graph below: as the number of iterations increases, the minimum tour found at each iteration approaches an asymptotic limit at the bottom, which we can suppose to be close to optimal–or at least *closer* to optimal than we started with. This graph reflects an execution of the algorithm on $n = 200$ cities. The reader will notice that oscillation still occurs from iteration to iteration; the pheromones do not guarantee that once a best-solution-so-far is found that it will be found every iteration, as each path is chosen probabilistically. After approximately 40 iterations, the colony begins to approach some bottom limit. Given even more iterations to run, the ant colony algorithm might have been able to find an even better solution.

Figure 1: Minimum Tours found in each ACO Iteration



As stated in the procedural description above, the AS variation, along with most ACO algorithms, depends on three heuristics: an edge selection heuristic, a pheromone deposit heuristic, and a termination heuristic. In order for the reader to correctly understand our implementation, we describe our definitions of these three heuristics in the subsections below. The edge selection heuristic and the pheromone deposit heuristic were both obtained from the definitions on Wikipedia [5]; the termination heuristic was an original creation by the members of our team.

2.3.1 Edge Selection Heuristic

Let us say that an ant has reached city u and there are cities v that the ant is allowed to travel to (i.e., the ant has not visited any of the cities v during this iteration and edges exist from u to v). Then, the probability that the ant travels down a particular edge (u, v_i) is determined by the equation

$$p_{u,v_i} = \frac{\left(\tau_{u,v_i}\right)^{\alpha} \cdot \left(\eta_{u,v_i}\right)^{\beta}}{\sum_{v_i \in v}\left(\left(\tau_{u,v_i}\right)^{\alpha} \cdot \left(\eta_{u,v_i}\right)^{\beta}\right)}$$

where $\tau_{u,vi}$ is the amount of pheromone on the edge from u to v_i ; $\eta_{u,vi}$ is equivalent[1] to $1 / (d_{u,vi} + 1)$ where $d_{u,vi}$ is the cost of the edge from u to v_i ; $\alpha \geq 0$ is a heuristic parameter that controls the

[1]In most implementations [5], $\eta_{u,vi}$ is equivalent to $1 / d_{u,vi}$ rather than $1 / (d_{u,vi} + 1)$. However, because edge costs of 0 are possible when the TSP GUI is set to "Hard" mode, we found it necessary to increment the value of the denominator by 1 to prevent divide-by-zero errors.

influence of the pheromones in the probability computation; and $\beta \geq 0$ is a heuristic parameter that controls the influence of edge costs in the probability computation.

Once the probability of each edge (u, v_i) is calculated, this probability distribution is then sampled to determine which city the ant "chose" to travel to next. Please note that probabilities are only calculated for the cities that this ant is allowed to travel to; this combination of allowed cities can differ from ant to ant–seeing as how each ant determines a different path probabilistically–and therefore, these probabilities must be computed each time an ant moves to a new city.

2.3.2 Pheromone Deposit Heuristic

After all ants in the ant colony have finished traveling (i.e., either found a complete tour or reached a dead end) in a single iteration, the pheromone levels for every edge (u_i,v_j) in the graph are updated according to the following equation:

$$\tau_{u,v_i} \leftarrow (1-\rho)\tau_{u,v_i} + \sum_k \Delta\tau_{k,u,v_i}$$

where $\tau_{u,vi}$ is the amount of pheromone on the edge from u to v_i , $\rho \in (0, 1)$ is a heuristic parameter that controls how much pheromone "evaporates"–as we might expect in a real-world situation–at each iteration, and $k \geq 0$ is the number of ants in the ant colony. The value $\Delta\tau_{k,u,vi}$ is determined by this secondary equation

$$\Delta\tau_{k,u,v_i} = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ uses edge } (u, v_i) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

where $Q \geq 0$ is a heuristic parameter that determines how much pheromone is deposited and L_k is the cost of the k^{th} ant's complete tour.

In essence, some percentage ρ of the pheromone amount evaporates from each edge and some proportion of pheromone is added to each edge according to the costs of the tours found by the ants that the edge was included in.

2.3.3 Termination Heuristic

In attempting to define a termination heuristic that would be useful for our implementation of the Ant Colony Optimization algorithm, our team brainstormed and produced the following three options:

1. Terminate the algorithm after a predetermined number of iterations.
2. Terminate the algorithm after the best-solution-so-far has been found X times in a row by the ant colony.

3. Terminate the algorithm after the cost of the minimum tour found by the ant colony in a single iteration has been within Y% of the cost of the best-solution-so-far, X times in a row.

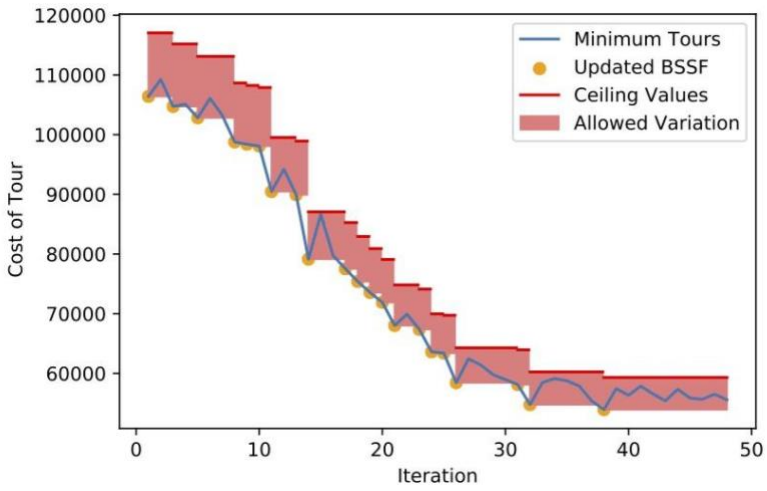
However, the first option does not guarantee that an optimal solution will be found or even approached asymptotically, nor is it dynamic. A predetermined number of iterations may work for some number of cities $n < z$, but when the number of cities crosses some threshold z, that predetermined number of iterations may not be sufficient to approach the optimal solution.

The second option does guarantee a sort of asymptotic behavior, but the probability of termination follows a Poisson distribution, meaning that the probability of termination is incredibly low and only increases once you reach a large number of iterations. In other words, finding the best-so-far-solution a certain number of times in a row is an incredibly rare event. We can see this by looking at the graph in *Figure 1* of the ACO simulation on $n = 200$ cities–even when the algorithm approached an asymptotic limit, there was still significant variation in the cost of the minimum tour from iteration to iteration. Therefore, because we wanted our algorithm to terminate in a reasonable time, we determined to not use this heuristic.

As a result, we decided to use the third option for our termination heuristic. This third option both a) guarantees that an asymptotic limit will be approached and b) that the algorithm will be able to terminate within a reasonable time once this asymptote is reached. We ran a few initial experiments with varying levels for the ceiling percentage Y and the number of iterations in a row X, but quickly narrowed in[2] on the following: our algorithm would terminate once the cost of the minimum tour had been within +10% of the best-solution-so-far 10 times in a row.

The graph at right is constructed from the same data as *Figure 1*–an ACO simulation on $n = 200$ cities–but a few extra details are included to demonstrate just how this termination heuristic operates: the (iteration, tour cost) points at which the best-solution-so-far was updated are marked in orange; the 110% ceiling value of the best-solution-so-far for each iteration is marked with a red line; and the region between the 110% ceiling values and the best-so-far-values is shaded in coral to illustrate the

Figure 2: Visualization of the regions of allowed variation over the course of ACO execution



[2] We designed a two-factor experiment testing three values of Y–1%, 5%, and 10%–against three values of X–5 iterations, 10 iterations, and 15 iterations. However, when we began the experiment with Y = 10%, we found that, even at this largest percentage, finding any number of iterations in a row still required a significant length of time. Narrowing that minimum window to +5% would take even longer and we wanted the algorithm to complete under 10 minutes. Therefore, we decided not to bother testing further and we settled on X = 10 iterations because it had produced the smallest tour costs out of the three iteration levels.

variation that is allowed by the algorithm when determining whether an asymptotic limit has been reached or not. It is not until the 38^{th} iteration that the best-solution-so-far is updated and the subsequent 10 iterations all return a minimum-cost tour with a cost that is less than 110% the value of the best-solution-so-far, at which point the algorithm then terminates after the 48^{th} iteration.

2.3.4 Theoretical Time Complexity

We start our algorithm by initializing the distance and the pheromone arrays. Both of these arrays are two-dimensional. The size of these arrays depends on the number of cities in the problem, having a $O(n^2)$ time and space complexity.

The time complexity to initialize the ant colony will depend on the number of ants. In our code, the number of ants is a constant with a value of 100. The time complexity will be O(k) where k is the number of ants.

When initialized, the ants will receive a reference to the distance and pheromone arrays. However, each ant does hold a list and a set. These will store the partial path and the cities that the ant has visited. The max size of each will depend on the number of cities in the problem, for a space complexity of O(nk).

The core part of the algorithm is to send the ants to explore the problem. The time complexity of this travel function has a worst case of $O(kn^2)$ where n is the number of cities and k the number of ants. The other functions called inside the loop also have time complexities of $O(n^2)$ or less.

If all the complete tours found in the iteration are within ten percent of the best-solution-so-far then we increase the number of iterations where the solution has been unchanged. Because of the inherited non-deterministic nature of the termination heuristic, the processing time is unpredictable.

Thus, the time complexity will depend on a value w, where w is an unknown rate at which the solutions found plateau to a local or global minimum as demonstrated in *Figure 1* and *Figure 2*. The value of w depends on the constants used on our code: α, β, ρ, k, and Q. All of these values play a part in the value of w. Is this complexity, that makes determining w only feasible empirically.

Time complexity: $O(wkn^2)$
Space complexity: $O(n^2 + nk)$

2.4 Determining the Parameters of the ACO

The Ant Colony Optimization algorithm also requires that we define the constant parameters α, β, ρ, k, and Q that were included in the definitions of the edge selection and pheromone deposit heuristics. To determine which values of these five parameters would minimize the solutions, we

conducted a grand total of 195 trials across five separate factorial experiments[3]. The designs of the experiments were original creations by our team, independent of influence from external sources. Each of these trials was conducted with the following configurations: the difficulty level in the GUI was set to "Hard (Deterministic)", the seed was set to 20, and the time limit was set to 60 seconds.

When designing the order and structure of these experiments, we first realized that the pheromone and edge-cost influence coefficients, α and β, were very likely to interact as they are both included–nonlinearly–in the probability computation of the edge selection heuristic. Therefore, we decided that our first task would be to carry out a two-factor experiment with varying levels of α and β.

The subsequent experiments grew out of this first experiment. After determining appropriate values of α and β, we used those α and β values in the experiment to determine ρ. We then used the determined values of α, β, and ρ in the experiment for k, and then the determined values for α, β, ρ, and k in the experiment for Q.

Before continuing, we would like to add a disclaimer on the effectiveness of our experimental process: It is entirely possible that all five coefficients interact with each other, which means that there is a possibility–likely a high one–that the values we determined from our experiments only *locally* minimize the solutions found by our ACO implementation. A better combination of α, β, ρ, k, and Q likely exists. However, to determine how the coefficients interact would require a 5-factorial experiment, with many levels for each of the five factors and many replicates conducted for each treatment, resulting in an incredibly large grand total of required trials that we did not have the time to conduct[4]. Therefore, we settled for the possibility that this process may have only found a local minimum of the cost function $c = f(\alpha, \beta, \rho, k, Q)$.

2.4.1 Pheromone and Edge-Cost Influence Coefficients, α and β

To determine values of α and β that minimized the cost of the tour returned by our ACO algorithm, we designed two experiments: an initial Low-Medium-High two-factor experiment to learn something about the basic interaction between α and β and a more complex two-factor experiment that we designed from the knowledge gained from the initial experiment.

The initial Low-Medium-High experiment was structured as follows:

1. The effect of two factors were evaluated, α and β, with three levels for each factor: Low (0.3), Medium (0.6), and High (0.9). Three replicates were conducted for each treatment, resulting in a grand total of 27 trials.

---

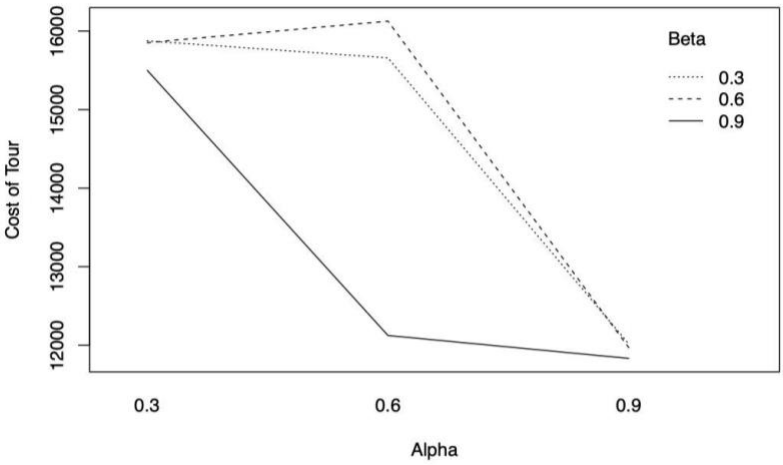
[3] The complete data tables for each of these experiments can be found in Appendix A.

[4] Even with only 5 levels for each of the 5 factors–a low number of levels per factor that wouldn't provide us with sufficient data to determine global optima–and 3 replicates per treatment–a much higher number of replicates would be needed per treatment to truly decrease the variance in the results–a grand total of $5^5 \cdot 3 = 9375$ trials would need to be conducted; with the length of each trial being 60 seconds, such an experiment would require more than 6 days to complete.

2. The number of cities *n* was set to 20. The values of ρ, *k*, and *Q* were arbitrarily set to 0.4, 50, and 1000, respectively.

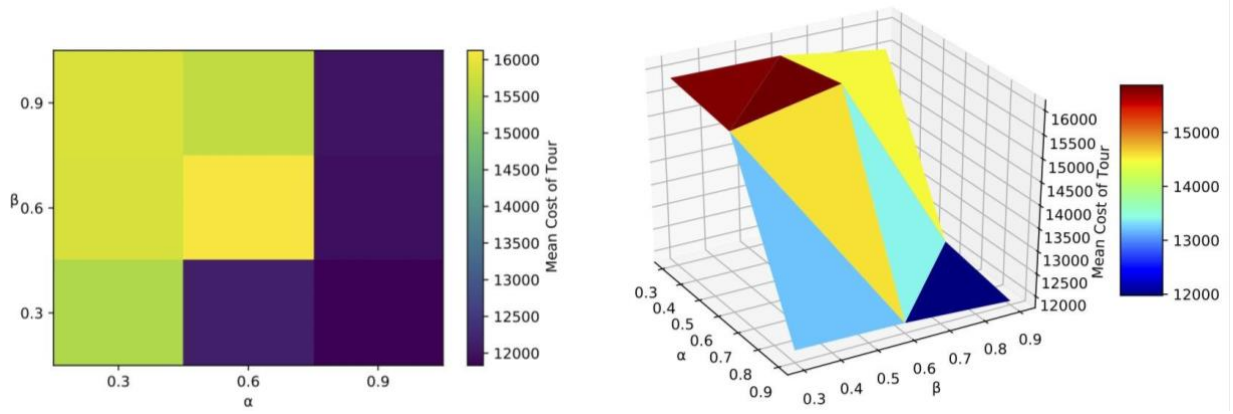
A two-way analysis of variance (two-way ANOVA) on the results of the experiment demonstrated that our initial understanding was correct: the interaction between α and β is significant at a 0.001% level. The interaction plot of the experiment–featured in *Figure 3* at right–confirms this fact, as the lines of the plot are not parallel.

Figure 3: Interaction Plot for Low-Medium-High α-β Experiment



As can be seen in the interaction plot, low values of α produced higher tour costs whereas higher values of α produced smaller tour costs. The cost of the tour did not seem to be as sensitive to low values of β. These two facts can also be seen by the color plot and the 3D surface plot in *Figure 4* below.

Figure 4: Color Plot and 3D Surface Plot of Mean Cost of Tour vs. α and β for Low-Medium-High Experiment



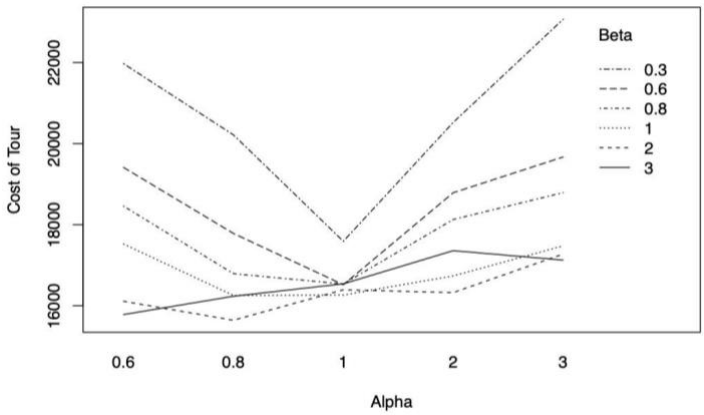
As a result of these discoveries, we designed and conducted a more complex two-factor experiment. In this second experiment, we only tested for higher values of α, since we determined in the first experiment that low values of α tended to produce higher tour costs. We also expanded the number of levels to be tested for both α and β in order to see if we could home in on a better local optimum than the combination $\alpha = 0.9$ and $\beta = 0.3$ had provided in the first experiment.

This second experiment was structured as follows:

1. The effect of two factors were evaluated, α and β. The α factor had five levels–0.6, 0.8, 1.0, 2.0, and 3.0–and the β factor had six levels–0.3, 0.6, 1.0, 2.0, 3.0. Three replicates were conducted for each treatment, resulting in a grand total of 90 trials.
2. The number of cities[5] n was set to 30. The values of ρ, k, and Q were arbitrarily set to 0.4, 50, and 1000, respectively.

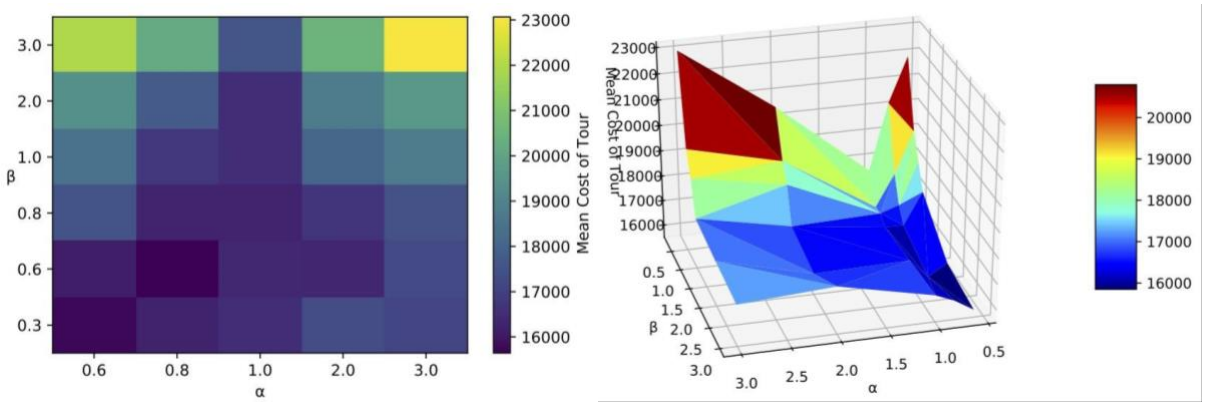
Two-way ANOVA on the results of this second experiment once again demonstrated that the interaction between α and β is significant at a 0.001% level. In addition to confirming this fact, the interaction plot of this second experiment–featured in *Figure 5* at right–reveals some interesting behavior of the system. When α = 1.0, the tour cost produced seems to converge[6] around a central–although not minimum–value.

Figure 5: Interaction Plot for Complex α-β Experiment



To understand where the local minima in this experiment is located, we refer to the color plot and the 3D surface plot in *Figure 6* below. The darkest section of the color plot and the lowest point on the surface plot are characterized by the point α = *0.8* and β = 2.

Figure 6: Color Plot and 3D Surface Plot of Mean Cost of Tour vs. α and β for Complex Experiment



Thus, for our implementation, we decided to use the values α = *0.8* and β = 2 for the pheromone and edge-cost influence coefficients found in the edge selection heuristic.

---

[5] At n = *20*, the algorithm was approaching a limit–the reader can see this reflected in the fact that the initial experiment returned very similar results for all three levels of β when α = *0.9*. We wanted to produce greater variation in the second experiment–we wanted to challenge the algorithm–so we increased the number of cities to 30.

[6] This convergence is also visible on the 3D surface plot, as a valley occurs at α = 1.0.

2.4.2 Pheromone Evaporation Coefficient, ρ

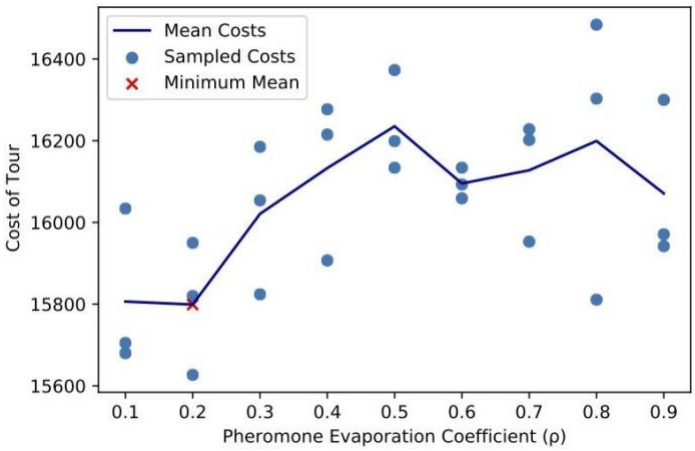
To determine a value of ρ that minimized the cost of the tour returned by our ACO algorithm–with all other parameters held constant–we designed the following single-factor experiment:

1. The effect of one factor, ρ, was evaluated. This factor had nine levels[7]–0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9. Three replicates were conducted for each treatment, resulting in a grand total of 27 trials.
2. The number of cities n was set to 30. The values of α and β–determined by the previous experiment–were set to *0.8* and 2, respectively. The values of k and Q were arbitrarily set to 50 and 1000, respectively.

The results of this experiment are plotted in the graph below-right.

Interestingly enough, a one-way analysis of variance (one-way ANOVA) on the results of this experiment showed that the pheromone evaporation coefficient, ρ, did not have a significant effect on the cost of the tour, even at a 10% significance level. It is clear from the graph that there was a high amount of variance within treatments as well. The graph also shows a seeming convergence around a central–not minimum–cost at $\rho = 0.6$, and a general upward trend as the value of ρ increases.

Figure 7: Minimum Tour Costs vs. Pheromone Evaporation Coefficient ρ



Despite the fact that one-way ANOVA reported the value of ρ to be insignificant, we decided to use the value $\rho = 0.2$ for our implementation, as it produced both the minimum actual cost and the minimum mean cost across the entire experiment.

2.4.3 Size of the Ant Colony, k

To determine a value of k that minimized the cost of the tour returned by our ACO algorithm–with all other parameters held constant–we designed the following single-factor experiment:

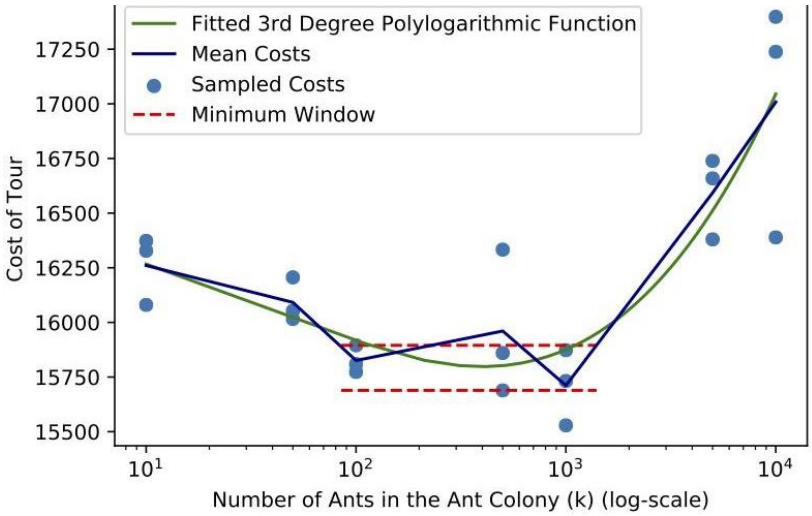
1. The effect of one factor, k, was evaluated. This factor had seven levels– 10, 50, 100, 500, 1000, 5000, 10000. Three replicates were conducted for each treatment, resulting in a grand total of 21 trials.

[7] These nine levels represent a very complete picture of the spectrum of possible values of ρ, since the value ρ must be between 0 and 1, as defined in the pheromone deposit heuristic.

2. The number of cities n was set to 30. The values of α, β, and ρ–determined by the previous experiment–were set to 0.8, 2, and 0.2, respectively. The value of Q was arbitrarily set to 1000.

A one-way ANOVA analysis on the results confirmed that the size of the ant colony k has an effect on the cost of the tour produced by the ACO algorithm, at a 0.1% significance level.

Figure 8: Minimum Tour Costs vs. Size of the Ant Colony k



The results of this experiment also showed a much clearer pattern than the previous experiment, so much so that we decided to run a regression analysis on the sampled data to extrapolate a model that could help us to understand the behavior of the ACO system as k changes. The regression analysis produced a 3rd-degree polylogarithmic function $\hat{c}(k)$ that fit the sampled data with an error $e \approx 1.1117$. This 3rd-degree polylogarithmic function $\hat{c}(k)$ is displayed in green on the graph at right of the experiment data and is defined by the equation below:

$$\hat{c}(k) = 150.87452446 \log^3 k - 751.92025297 \log^2 k + 850.59418128 \log k + 16017.35071693$$

Upon further inspection of the data and the model, we realized there was not much difference between the costs of $k = 100$, 500, and 1000; seven out of nine of the points for those values are all located within a small "minimum window" that only spans a cost value of 206. Essentially, we could choose any of those three k values and we would likely get similar results. However, we chose $k = 100$ because a smaller value of k would decrease the amount of time required for each iteration of the ACO algorithm; this would allow us to fit more iterations within a set period of time and, therefore, to approach an asymptotic limit much faster for values of n higher than 30. If we had chosen $k = 1000$, the time required per iteration when the number of cities n reached 200 would have been costly in the time efficiency of the algorithm.

2.4.4 Pheromone Deposit Coefficient, Q

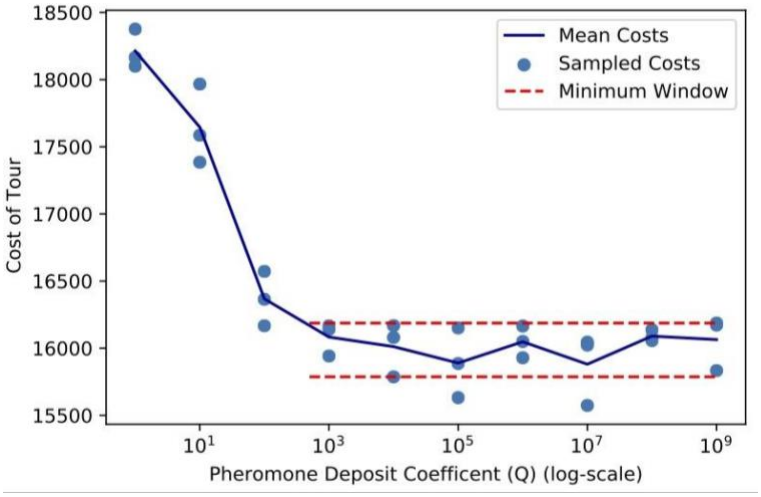
To determine a value of Q that minimized the cost of the tour returned by our ACO algorithm–with all other parameters held constant–we designed the following single-factor experiment:

1. The effect of one factor, Q, was evaluated. This factor had ten levels–10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, and 10^9. Three replicates were conducted for each treatment, resulting in a grand total of 30 trials.

2. The number of cities n was set to 30. The values of α, β, ρ, and k–determined by the previous experiment–were set to 0.8, 2, 0.2, and 1000, respectively.

A one-way ANOVA analysis on the results confirmed that Q has an effect on the cost of the tour produced by the ACO algorithm, at a 1^{-11}% significance level (practically nothing).

Figure 9: Minimum Tour Costs vs. Pheromone Deposit Coefficient Q



Similar to the last experiment, a small "minimum window" is clearly present– spanning a cost value of 400 this time–wherein nineteen out of twenty-one of the data points from $Q = 10^3$ to $Q = 10^9$ are located; in addition, there is a significant level of variation within this window. This asymptotic behavior, visually evident in the graph, seems to imply that an increase in the value of Q can only improve the algorithm to a certain point and no further.

Out of the seven values of Q that were within this minimum window, we decided to use the value $Q = 1000$ for our algorithm implementation. Although $Q = 1000$ did not produce the absolute minimum mean cost across the data, it was the smallest of the Q values inside the "minimum window". All other values of Q in the minimum window seemed to simply be an excessive waste of computer memory and arithmetic complexity when compared to $Q = 1000$. The smaller the value of Q, the simpler our implementation would be.

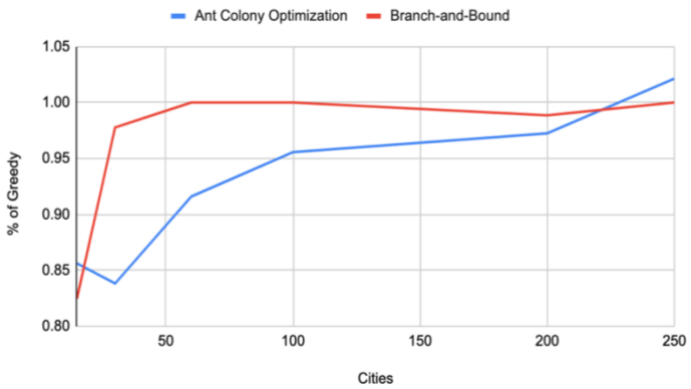
3. Results

Table 1: TSP Algorithm Results

	Random		Greedy			Branch and Bound			Ant Colony Optimization		
Cities	Time (sec)	Path Length	Time (sec)	Path Length	% of Random	Time (sec)	Path Length	% of Greedy	Time (sec)	Path Length	% of Greedy
15	0	2	0	11820	54.34	6.9	9748	82.47	2.07	10119	85.62
30	0.07	41819	0.03	17683	42.29	600	17288	97.77	22.27	14823	83.83
60	161.86	82884	0.29	26254	31.68	600	26254	100	148.79	24049	91.6
100	600	TB	1.31	36854	TB	600	36854	100	88.48	35223	95.57
200	600	TB	13.67	54745	TB	600	54119	98.86	509.4	53241	97.25
250	600	TB	29.88	62263	TB	600	62263	100	600	63593	102.14

For smaller cities (15 cities), our algorithm performed slightly worse than Branch-and-Bound (85.62% of Greedy vs 82.47%). However, for larger numbers of cities, it did better. This may be because the Branch-and-Bound algorithm can give priority to states with low cost and that are deeper in the state creation process, but when expanded, are followed by states that greatly increase the path length and thus do not end up being good solutions. On the other hand, in the Ant Colony Optimization, at each step of the path, probability and weights are used to determine the next city traveled to. This means that on occasion new paths that may initially seem costly but end up being shorter, may be explored. As the graph below shows, the path length produced by the Branch-and-Bound algorithm converges to that of the Greedy algorithm as the number of cities increases keeping the max runtime constant at 600 seconds. This is because the Branch-and-Bound uses the Greedy algorithm as its initial best solution so far and in most cases, doesn't have enough time to find a better one. The path lengths produced by the Ant Colony Optimization also increase as the number of cities increases, taking the max runtime allotted in all of the tests run with 250 cities. In addition, it also begins to perform worse on average than the Greedy algorithm, this is likely due to being limited by the time allotted similarly to the Branch-and-Bound algorithm. Thus, it is likely that the paths produced by the Branch-and-Bound algorithm would have been shorter if it was allowed to run for longer. Similarly, if more time was allotted for the Ant Colonization Optimization, or changes were made to the constants used to determine how close a path must be to the best-solution-so-far for 10 iterations until it terminates, the cost of the paths could decrease.

Figure 10: Performance of Ant Colony Optimization vs. Branch-and-Bound



3.1 Screenshots of Typical Examples

Figure 12: Greedy Algorithm - n = 60, Difficulty = Hard, Seed = 20

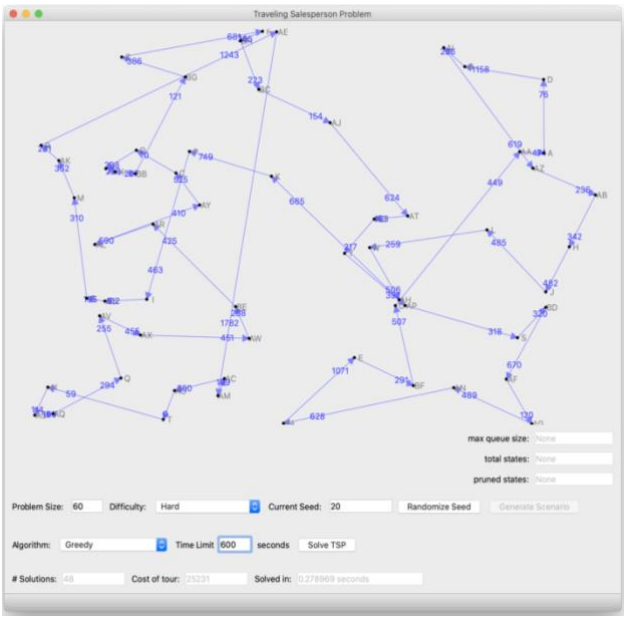


Figure 11: ACO Algorithm - n = 60, Difficulty = Hard, Seed = 20

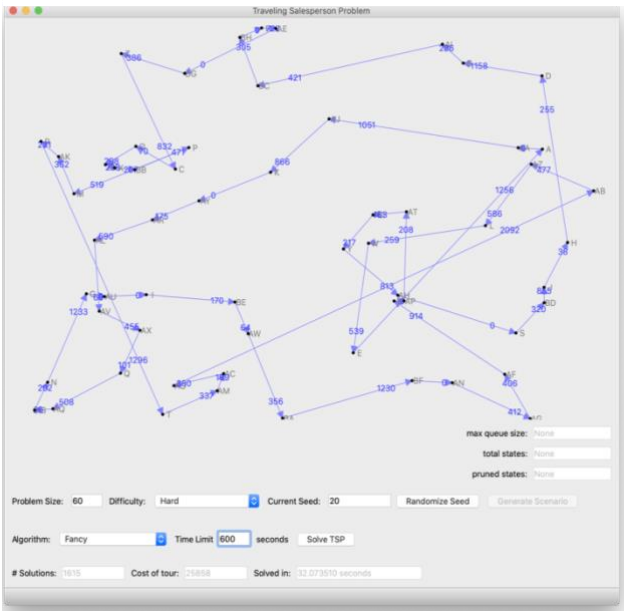


Figure 14: Greedy Algorithm - n = 200, Difficulty = Hard, Seed = 20

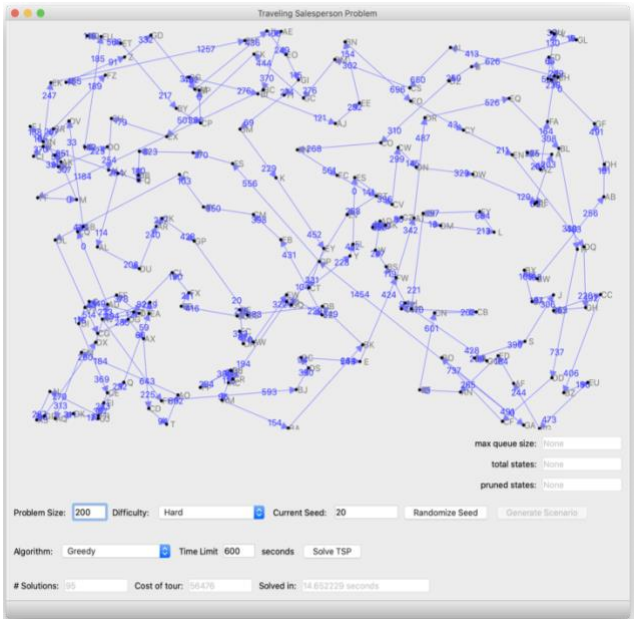
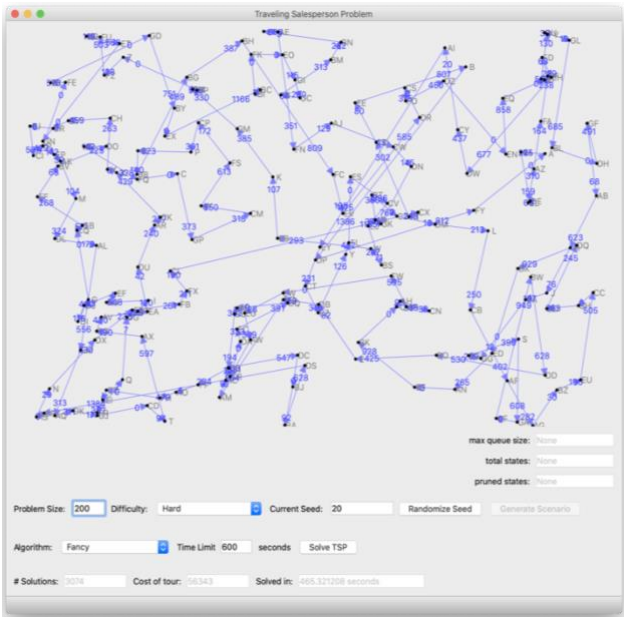


Figure 13: ACO Algorithm - n = 200, Difficulty = Hard, Seed = 20

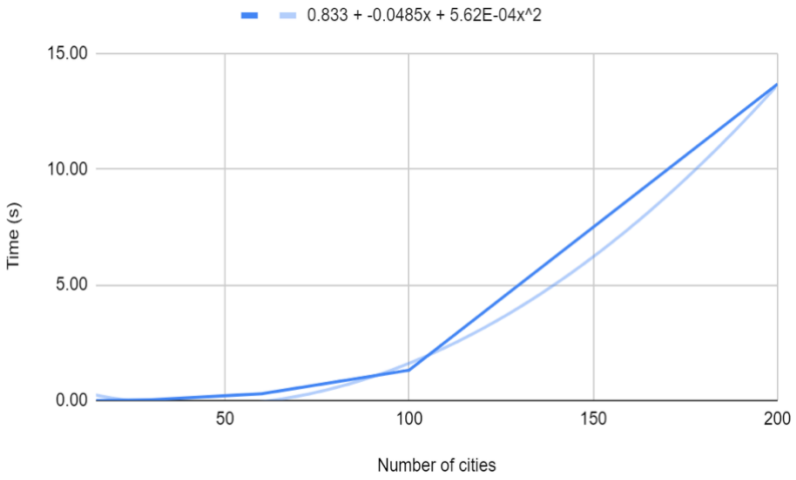


4. Discussion

4.1 Empirical Complexity of Greedy Algorithm

The expected run time was $O(n^3)$. This reflects the worst-case scenario where for every city you will do $O(n^2)$ work to find the cheapest city you can travel to. In reality you will not do that much work for every node. Some nodes might not be connected to each other, and an incomplete path will be returned. The empirical run time is $O(0.833 - 0.0485 + 5.52 * E\text{-}04 * n^2)$ where n is the number of cities in the problem. The empirical run time is faster than the expected value.

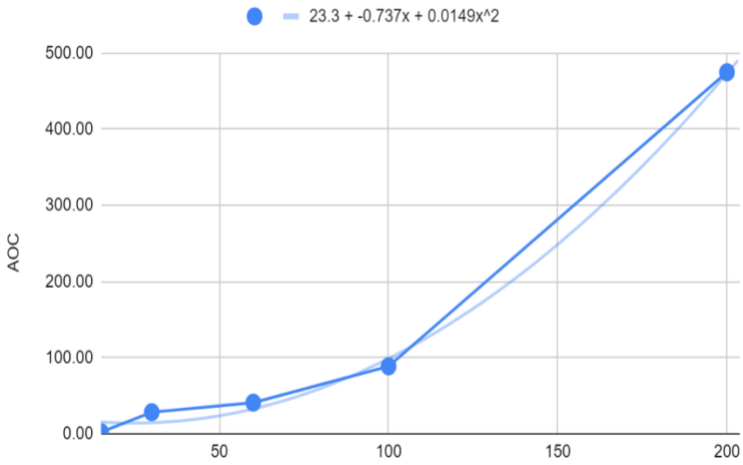
Figure 15: Run Time of the Greedy Algorithm vs. Number of Cities n



4.2 Empirical Complexity of the Ant Colony Optimization Algorithm

The theoretical run time for the AOC algorithm is $O(wkn^2)$ Where *w* is the rate at which the algorithm plateaus to a stable solution, *k* is the number of ants and *n* is the number of cities. We fitted a polynomial line to describe the behavior of the code. The data is described by the line $23.3 - 0.737n + 0.0149 n^2$. The polynomial element of the solution is of the same degree as expected and the rate at which the solution converges would be 0.0149. Although the time complexity from other elements is included in that function, the specific value for *w* can only be calculated once other factors are removed. As mentioned before, the non-deterministic side of AOC makes it impossible to predict w. But for our model the number of cities will be the largest factor for predicting the run time. For large problems our solution will take much longer than for smaller problems.

Figure 16: Run Time of the ACO Algorithm vs. Number of Cities n



4.3 Pros and Cons of the Ant Colony Optimization Algorithm

Our Ant Colony Optimization algorithm consistently found better solutions than both the Greedy and Branch-and-Bound approaches when the number of cities was between 30 and 200. ACO has a probabilistic nature. This allows it to explore paths that might not seem optimal at that point but can produce a good solution and exit to a local minimum. Another benefit of this algorithm is that it is easy to understand and implement.

One of the cons we found while implementing the Ant Colony Optimization algorithm is that it took longer than the Random and Greedy algorithms on small problems. This is due to the parameters used to determine whether we have obtained a good solution or not. This brings another possible problem; the constants used in the code might be optimal for a certain number of cities but perform poorly on different numbers. A possible solution to this problem could be to introduce dynamic parameters that depend on the size of the problem. While this could be beneficial to the overall runtime, it also introduces more complexity to the value of rate at which the algorithm converges to a solution (w). Additionally, finding good parameters to use as constants or designing a good equation to produce the values are time consuming tasks.

The Ant Colony Optimization algorithm had a good overall performance for the given constraints of this project. While it's probabilistic side can bring a lot of complexity to the code, the easy implementation, and possible improvements make it a great tool for finding good estimates of complex problems.

5. Future Work

Additional work could be done to further optimize the results achieved by an ACO approach to solving TSP. One idea that was pursued while developing our algorithm was to create a multi-threading-based implementation in order to speed up the rate at which the entire ant colony can traverse the graph. After running analysis on this multi-threaded approach, it was discovered that the overhead for creating and using threads in Python was greater than the benefits we saw up to a problem size of 60. However, were this algorithm to be implemented using a different programming language (such as C or C++, which both include threading libraries), taking a multi-threaded approach could yield more optimal results.

As mentioned previously, other ACO algorithms have been created, each branching from the original Ant System approach that we implemented in this project. These algorithms each have a unique spin off that gives weight to different elements of the original Ant System (AS) algorithm with the main variation being in how pheromone levels are updated. For example, the Max-Min (MMAS) varies from the original AS approach in that rather than having every ant which completes a tour update the pheromone levels of the graph, only the global best tour or iteration best tour are used in updating the pheromones. Additionally, max and min pheromone levels are set and each edge is initialized with the maximum pheromone level which leads to more solutions being explored from the beginning. The Ant Colony System (ACS) approach has also been shown to be better than the AS approach when solving TSP. ACS allows only the ant that returns with the best solution to be used in updating the pheromones after each iteration. It also favors the lower-cost edges and uses a local pheromone updating rule during each iteration [5].

Using a combination of an Ant Colony Optimization algorithm combined with a Local Search Algorithm has been shown to yield better results when tackling TSP [3]. If we were to continue working on our algorithm to optimize our results we would explore and analyze using a hybrid algorithm.

Lastly, to improve upon our current algorithm and design we would ideally run a factorial experiment with all five variables to determine the best values for our cost function $c = f(\alpha, \beta, \rho, k, Q)$. This would allow us to study the effect of each factor on the response factor as well as the effect of the interactions between factors. With the current level of testing we have performed, we were only able to confidently assume we have found a local minimum for c but after running a 5-factorial experiment we would gain more conclusive results as to what the global minimum for c is.

6. References

[1] Anna R. Karlin, Nathan Klein, & Shayan Oveis Gharan. (2020). A (Slightly) Improved Approximation Algorithm for Metric TSP. Retrieved December 10, 2020, from https://arxiv.org/pdf/2007.01409.pdf

[2] M. Dorigo, V. Maniezzo and A. Colorni, "Ant system: optimization by a colony of cooperating agents," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 26, no. 1, pp. 29-41, Feb. 1996, doi: 10.1109/3477.484436. Retrieved December 10, 2020, from http://www.cs.unibo.it/babaoglu/courses/cas05-06/tutorials/Ant_Colony_Optimization.pdf

[3] Stützle, T., & Dorigo, M. (1999). ACO Algorithms for the Traveling Salesman Problem. In K. Miettinen, P. Neittaanmäki, J. Periaux, & M. M. Mäkelä (Authors), *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications* (pp. i-xxiii). Chichester, England: John Wiley & Sons. Retrieved December 10, 2020, from http://staff.washington.edu/paymana/swarm/stutzle99-eaecs.pdf

[4] Western Exterminator Company. (2020). The ant colony: Structure and roles. Retrieved December 10, 2020, from https://www.westernexterminator.com/ants/the-ant-colony-structure-and-roles/

[5] Wikipedia. (2020, December 07). Ant colony optimization algorithms. Retrieved December 10, 2020, from https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

Appendix A: Data Tables from Experiments and Algorithm Tests

Table A-1: Data from Low-Medium-High Two-Factor α-β Experiment

α	β	Cost of Best Tour	Mean Cost of Best Tour
0.3	0.3	16047	**15875.66667**
0.3	0.3	15221	
0.3	0.3	16359	
0.3	0.6	15138	**15852.66667**
0.3	0.6	16487	
0.3	0.6	15933	
0.3	0.9	15726	**15499.66667**
0.3	0.9	15835	
0.3	0.9	14938	
0.6	0.3	14902	**15659**
0.6	0.3	16185	
0.6	0.3	15890	
0.6	0.6	16377	**16126.66667**
0.6	0.6	15789	
0.6	0.6	16214	
0.6	0.9	12183	**12124.33333**
0.6	0.9	12105	
0.6	0.9	12085	
0.9	0.3	12085	**12016.66667**
0.9	0.3	12085	
0.9	0.3	11880	
0.9	0.6	12062	**11970.66667**
0.9	0.6	12269	
0.9	0.6	11581	
0.9	0.9	11581	**11832.33333**
0.9	0.9	11647	
0.9	0.9	12269	

Table A-2: Data from Complex Two-Factor α-β Experiment

α	β	Cost of Best Tour	Mean Cost of Best Tour
0.6	0.3	20857	**21971.33333**
0.6	0.3	22338	
0.6	0.3	22719	
0.6	0.6	20005	**19412**
0.6	0.6	19001	
0.6	0.6	19230	
0.6	0.8	18424	**18459.33333**
0.6	0.8	18308	
0.6	0.8	18646	
0.6	1	17592	**17524.33333**
0.6	1	17689	
0.6	1	17292	
0.6	2	16302	**16111.66667**
0.6	2	16501	
0.6	2	15532	
0.6	3	15687	**15784.33333**
0.6	3	15532	
0.6	3	16134	
0.8	0.3	20614	**20215**
0.8	0.3	20333	
0.8	0.3	19698	
0.8	0.6	17181	**17784.66667**
0.8	0.6	18485	
0.8	0.6	17688	
0.8	0.8	16257	**16792.33333**
0.8	0.8	16908	
0.8	0.8	17212	
0.8	1	16125	**16255**
0.8	1	16453	
0.8	1	16187	

Table A-2: Data from Complex Two-Factor α-β Experiment (cont'd)

α	β	Cost of Best Tour	Mean Cost of Best Tour
0.8	2	15652	**15644.33333**
0.8	2	15675	
0.8	2	15606	
0.8	3	16406	**16236.66667**
0.8	3	16018	
0.8	3	16286	
1	0.3	17994	**17591.66667**
1	0.3	17263	
1	0.3	17518	
1	0.6	16505	**16520.33333**
1	0.6	16731	
1	0.6	16325	
1	0.8	17017	**16538.33333**
1	0.8	16408	
1	0.8	16190	
1	1	16379	**16266**
1	1	15999	
1	1	16420	
1	2	16335	**16395**
1	2	16206	
1	2	16644	
1	3	16331	**16543**
1	3	16708	
1	3	16590	
2	0.3	19670	**20521.66667**
2	0.3	20149	
2	0.3	21746	
2	0.6	18231	**18788.33333**
2	0.6	18841	
2	0.6	19293	

Table A-2: Data from Complex Two-Factor α-β Experiment (cont'd)

α	β	Cost of Best Tour	Mean Cost of Best Tour
2	0.8	19015	**18129**
2	0.8	17265	
2	0.8	18107	
2	1	16792	**16738**
2	1	16420	
2	1	17002	
2	2	16115	**16326.33333**
2	2	16444	
2	2	16420	
2	3	17298	**17360.66667**
2	3	18064	
2	3	16720	
3	0.3	21807	**23065.66667**
3	0.3	22044	
3	0.3	25346	
3	0.6	20182	**19669.66667**
3	0.6	19123	
3	0.6	19704	
3	0.8	17595	**18791.33333**
3	0.8	18802	
3	0.8	19977	
3	1	17344	**17478.33333**
3	1	17070	
3	1	18021	
3	2	17226	**17276.66667**
3	2	18003	
3	2	16601	
3	3	17805	**17128**
3	3	17001	
3	3	16578	

Table A-3: Data from Single-Factor ρ Experiment

ρ	Cost of Best Tour	Mean Cost of Best Tour
0.1	16034	**15806.33333**
0.1	15705	
0.1	15680	
0.2	15627	**15799**
0.2	15950	
0.2	15820	
0.3	16185	**16021**
0.3	15824	
0.3	16054	
0.4	15907	**16133**
0.4	16215	
0.4	16277	
0.5	16134	**16235.33333**
0.5	16199	
0.5	16373	
0.6	16093	**16095.33333**
0.6	16134	
0.6	16059	
0.7	16228	**16127.66667**
0.7	16202	
0.7	15953	
0.8	16484	**16199.33333**
0.8	15811	
0.8	16303	
0.9	16300	**16071**
0.9	15971	
0.9	15942	

Table A-4: Data from Single-Factor k Experiment

k	**Cost of Best Tour**	**Mean Cost of Best Tour**
10	16328	**16260.33333**
10	16080	
10	16373	
50	16206	**16091.66667**
50	16016	
50	16053	
100	15809	**15825.66667**
100	15773	
100	15895	
500	15689	**15960.66667**
500	16333	
500	15860	
1000	15873	**15711.33333**
1000	15732	
1000	15529	
5000	16739	**16592.66667**
5000	16380	
5000	16659	
10000	16389	**17008.33333**
10000	17398	
10000	17238	

Table A-5: Data from Single-Factor Q Experiment

Q	**Cost of Best Tour**	**Mean Cost of Best Tour**
1	18165	**18213.66667**
1	18376	
1	18100	
10	17385	**17646**
10	17967	
10	17586	
100	16572	**16367.66667**
100	16364	
100	16167	
1000	16166	**16083**
1000	16141	
1000	15942	
10000	16080	**16011.66667**
10000	16168	
10000	15787	
100000	15632	**15889.66667**
100000	15886	
100000	16151	
1000000	16050	**16048**
1000000	15929	
1000000	16165	
10000000	15574	**15880.66667**
10000000	16045	
10000000	16023	
100000000	16137	**16090**
100000000	16057	
100000000	16076	
1000000000	15833	**16064**
1000000000	16172	
1000000000	16187	

Table A-6: Data from TSP Algorithm Tests

# of Cities	Seed	Random		Greedy		Branch-and-Bound		ACO	
		Time (sec)	*Path Length*	*Time (sec)*	*Path Length*	*Time (sec)*	*Path Length*	*Time (sec)*	*Path Length*
15	20	0.00	23393	0.00	11072	0.56	9687	2.50	10211
15	30	0.00	20392	0.00	15704	6.21	10913	2.16	11586
15	40	0.00	26230	0.00	11331	3.30	9928	1.66	10200
15	50	0.00	19942	0.00	11800	1.83	10405	2.38	10795
15	60	0.00	18808	0.00	9191	22.59	7805	1.64	7805
30	20	0.04	44057	0.03	20395	600.00	19809	18.43	15143
30	30	0.06	42818	0.03	17367	600.00	17319	13.32	14644
30	40	0.11	42818	0.03	16128	600.00	16128	15.66	14796
30	50	0.12	41147	0.04	16760	600.00	16455	22.71	14656
30	60	0.01	38256	0.03	17767	600.00	16730	41.24	14878
60	20	600.00	inf	0.42	24622	600.01	24622	41.82	24149
60	30	148.49	80638	0.26	25123	600.01	25123	40.93	22992
60	40	6.01	83678	0.26	28400	600.00	28400	38.62	25383
60	50	38.29	85765	0.27	27024	600.00	27024	22.58	25983
60	60	16.50	81455	0.26	26103	600.00	26103	600.01	21739
100	20	600.00	inf	1.30	38093	600.01	38093	91.36	34305
100	30	600.00	inf	1.31	35565	600.00	35565	86.66	34364
100	40	600.00	inf	1.33	36569	600.00	36569	108.66	34709
100	50	600.00	inf	1.29	38915	600.00	38915	77.90	37092
100	60	600.00	inf	1.31	35130	600.00	35130	77.84	35643
200	20	600.00	inf	13.37	56227	600.00	53100	607.24	52217
200	30	600.00	inf	13.69	55583	600.00	55583	582.46	52158
200	40	600.00	inf	14.25	54090	600.00	54090	474.76	54161
200	50	600.00	inf	13.46	55158	600.00	55158	424.15	55006
200	60	600.00	inf	13.58	52665	600.00	52665	458.38	52661
250	20	600.00	inf	30.12	62517	600.01	62517	601.11	63716
250	30	600.00	inf	30.59	64090	600.00	64090	600.09	64617
250	40	600.00	inf	28.97	60787	600.00	60787	600.05	63758
250	50	600.00	inf	30.27	62132	600.04	62132	600.03	62739
250	60	600.00	inf	29.45	61790	600.01	61790	600.01	63136