Ant Colony Optimization for the Traveling Salesman Problem

Michael A. Webber, Andrew W. Stoneman, and Alexander L. Clark

Keywords: ACO, Traveling Salesman Problem (TSP), NP-hard, Tour, Optimal Tour

Abstract: The Traveling Salesman Problem (TSP) is a standard optimization problem that aims to minimize the path of a salesman traveling between a collection of cities. A completed TSP occurs when the salesman finds the optimal path to take or when they have gone through a certain number of iterations. In this experiment, we test one generic Ant Colony Optimization (ACO) algorithm along with two close variations to try and find a solution closest to the most optimal TSP. All three algorithms, although they are slightly different, operate in the same general form, using heuristics, and a digitized version of a chemical called pheromone that ants use to find food sources, to find the most optimal path. We compare the three ACO algorithms performance based on various factors such as the most optimal tour it can achieve as well as the number of iterations it takes to find the optimal solution. We will also determine the success of an algorithm by measuring which performed best on different population sizes. After comparing test problems, we recommend the Standard Ant System algorithm for TSP, as the other variants delivered both worse optimal tours and efficiency.

1 INTRODUCTION

In this study, we are implementing and comparing the Standard Ant System (SAS) algorithm and two other variations of the Ant Colony Optimization (ACO) algorithm for solving Traveling Salesman Problems (TSP). TSP is a common problem in computer programming where a salesman is trying to visit every “city” in a list of cities and return to the starting city in the shortest distance possible. The shortest distance that is possible for a salesman to travel in a given TSP is referred to as the optimal solution. All instances of TSP for this study are symmetric, meaning that the distance from City A to City B is equal to the distance from City B to City A.

In order to find the optimal tour, we are implementing three different variations of the Ant Colony Optimization (ACO) Algorithm. In SAS, the salesman travels city to city by choosing the next city of the tour probabilistically. This probability is based on both heuristic information, and a “pheromone” level, which is used to denote the paths from one city to another that have led to shorter tours. In the first variation (ACOV1), we make the evaporation rates nonuniform when updating pheromone levels during each tour. This is done by evaporating legs in tours that are worse (greater than) a value slightly larger than the tour length average by a small evaporation factor, and depositing pheromones like the standard algorithm if they are better (less than) the value slightly greater than the average tour length for the current iteration. In the second variation (ACOV2) that we implement, we increase the level of randomness when determining which city to travel to next instead of exclusively using the probability like the

standard version and ACOV1. The probability of constructing a random tour for an ant will become less likely as the number of iterations increases.

In order to compare the three variations of the ACO on the TSP, we looked first and foremost if any produced the optimal tour, but that seems to never be the case. Thus, we ran two types of experiments: First, we specified the number of iterations to see how close the algorithm gets to optimal, also noting what iteration it achieved this in; and second, measured the number of iterations it took to reach a specified percentage over the optimal tour length, allowing us to specify how close the algorithm should get before stopping.

Our results showed that SAS performed best between the three ACO implementations. It returned both the best results, and typically did it faster than the other variants. However, the ACOV1 was quite close to SAS in performance, typically returning results only marginally larger than SAS, although typically taking longer to reach its personal optimal. ACOV2 returned significantly worse results than the other two implementations, both in time and performance.

In Section 2, we will look deeper into the Traveling Salesman Problem and dissect the entirety of the problem. In section 3 will explore the meaning of Ant Colony Optimization and what is required to implement a successful and efficient algorithm. This leads into Section 4, where we will discuss our methods for comparing the various ACO algorithms. Section 5 will analyze the results of the experiments that we have run on 3 different algorithms. Section 6 will use what we have found in our results to show what could be done to improve our algorithm if we were given more time to optimize the code. Lastly, Section 7 will conclude the paper by summarizing our findings and determining the most efficient among the three ACO algorithms for solving TSP.

2 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is a common problem in computer programming that is NP-hard and was first contemplated mathematically in the 1930s. A NP-hard problem is one that is at least as hard to solve as a nondeterministic polynomial time problem. More often than not, NP-hard problems are actually much harder than NP-problems. The idea of the TSP is relatively straightforward. There is an input of a list of cities and their positions, thus a distance can be associated with the space between each city. The salesman must travel to every city and return to the city that he or she started at, the goal being to travel the shortest distance required to visit all cities. Constructing this path becomes obviously complicated quickly, as the number of potential tour subsets drastically increases every time a city is added to the problem.

This problem has gotten this specific name of TSP because of the origins of the problem, but it can be applied more generally to optimization problems. The broader scope of this problem is to minimize the distance of a path through multiple different points. Other examples could be minimizing the path a bus must travel to pick up students for school, or minimizing the distance the mailman must travel in order to deliver all their mail before returning to the post office. As we can see, we will be specifically addressing the TSP, but if thought about more generally, it can apply to a variety of different optimization problems that take place every day.

3 ANT COLONY OPTIMIZATION

In this experiment, we used three different variations of the Ant Colony Optimization (ACO) algorithm: the Standard Ant System (SAS) algorithm, the first variation (ACOV1), and the second variation (ACOV2).

ACO is, as the name suggests, inspired by how ant colonies function when scavenging for food sources. When an ant within the colony finds a food source, he brings it back to the colony, and leaves a trace of chemicals called "pheromones," which other ants can discover, and follow the path the chemicals create to the food source. These pheromones evaporate over time, but if more ants are continually visiting a food source, more pheromones will be laid down, creating an ideal path to follow for all other ants in the colony. If the food source were to run out, then less ants would visit it, and the pheromones would evaporate so that other ants do not continue to visit a poor food source.

In the context of the TSP problem, a path between two cities is analogous to a path between a potential food source and the colony. This path between two cities is referred to as a "leg" in a tour, and every leg in the TSP contains a certain amount of pheromone, which is roughly based on the length of a tour that an ant had in which that leg was a part. After every iteration, pheromone is updated, both being evaporated and deposited based on how desirable of a leg it was, which is based on if it is included in a good or bad tour or not utilized at all.

In the SAS, an ant starts at a random city, repeatedly chooses its next city based on the level of pheromones and distance on the leg from its current city to a potential next, and once it has visited all cities, it completes its tour. This is performed on a set number of ants, (20 ants were used in this experiment), and once an iteration completes, a best value is determined amongst all of the ants, referring to the best tour for that iteration. This is then run for either a set number of iterations in search of an increasingly better optimal performance. After all ants have constructed their tours for a given iteration, the pheromone levels for each leg are updated based on the performance (tour distance) of each ant that utilized that leg.

The formula seen in Figure 1 denotes the probability of ant k in city i going to city j based off of pheromone level, τ_{ij}, the inverse of the distance from i to j, $\eta_{ij} = 1/d_{ij}$, two positive constants, α and β (in this experiment, set to 1.5 and 3.5 respectively), which denote the importance of pheromones versus heuristics, and in the denominator, the sum of the cities that the ant has not yet visited, N_k, to normalize the probability.

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_k} \tau_{il}^{\alpha}\eta_{il}^{\beta}}$$

Figure 1: Probability that an ant k will move from city i to city j

After every tour, the pheromones on the legs are updated using the equation in Figure 2. Pheromone is evaporated uniformly on all legs by a factor of 1 – ρ (where ρ equals 0.5 in this experiment), and deposited on all legs that were used in ant's tour proportional to the distance of the ant's tour–the longer the tour, the less deposited on those legs, and the shorter, the more deposited. The changes in the pheromone levels alter the probabilities of selecting that leg in the next tour, ideally pointing an ant in to a leg included in the most optimal solution. The pheromones are updated based on the number of ants, m, the pheromone levels on a leg from i to j, τ_{ij}, and an evaporation factor, ρ.

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

Figure 2: Updating the pheromone on a leg $i\,j$

In ACOV1, everything is the same as the SAS while the ants construct tours. However, the difference lies in the way pheromone is updated. All legs on a given tour are still evaporated uniformly by a factor of (1 – ρ), (where ρ equals 0.5 in this experiment). However, when depositing pheromone, we calculate the average tour length for all ants in the current iteration, and then multiply the average by an average multiplier (set to 1.03), which increases the average slightly. If a given ant's tour is better (less distance) than the average multiplied by the average multiplier, then the ant has pheromone deposited on all of its legs the same way as SAS, which is proportional to the distance of its tour. On the other hand, if an ant's tour is worse (larger distance) than the average multiplied by the average multiplier, then the pheromones are evaporated on that leg by an additional evaporation factor (set to 0.9). Theoretically, this allows for some of the worse tour's legs in a given iteration to be less likely chosen in future iterations. The average multiplier is a very small number, allowing for a little bit less of the legs to evaporate, as overdoing it is not desirable. The evaporation factor is also small for the same reason, since a leg being on a bad tour for a given iteration does not mean it is not potentially a part of a more optimal solution. Note that these values were picked based on testing multiple combinations of numbers, these seeming to prove to work best, though more time would certainly be needed for confirmation of these values.

In ACOV2, we add a degree of randomness to the tour constructing portion of ACO, deviating slightly from the SAS. Based on a probability that decreases as the number of iterations increases, an ant either constructs a tour normally using the pheromone and distance equation, or generates a tour completely randomly. In theory, this allows for the ant to explore more paths in the beginning, and then hone in on better paths as the number of iterations increases. The pheromone updating process will work the same as the first variation, but it is probable that the legs will be updated differently due to the added random tours. The exact equation for determining whether a tour will be random or not is 1 divided by the number of iterations. This equation causes it to quickly become unlikely that a tour will happen randomly after a few iterations, which is intentional, as we do not want to construct tours randomly for a long time.

4 EXPERIMENTAL METHODOLOGY

After implementing the three algorithms, we tested them on four different files, ranging in city size from 2,103-5,915. We tested each algorithm five times per file, calculating averages across the five experiments. This allowed us to see how different algorithms performed across file size, as well as how long they took to run on average.

Next, we compared algorithms against each other in two different ways to make an accurate comparison and determine the most optimal and fastest algorithm(s). Part of the reason we decided to implement ACOV1 and ACOV2 the way that we did was in hopes that these variations would give additional insight on the impact of the city/leg selection process compared to the pheromone updating process.

The first test that we ran to compare the three ACO algorithms was by giving a specific number of iterations to run and measuring the distance of the best tour found divided by the optimal tour. This test was important to our experiment as it shows the best solution each algorithm computed in relation to the optimal while under iteration constraints, rather than letting it run an infinite number of times.

The second test we ran to compare the three ACO algorithms was measuring the number of iterations needed to find a specified fitness needed to end the program. By fitness, we refer to a user-specified percentage over the optimal solution (the best tour divided by the optimal solution). This test was important to include as it gives insight into which algorithm is most efficient, by demonstrating how long it takes to reach a particular point of accuracy.

5 RESULTS

Figures 3-6 depict the most optimal solution that each algorithm could come up with across all file sizes on average. This is calculated by dividing the best tour in fifty iterations by the optimal tour. Thus the smaller the number, the closer to the optimal, indicating better performance.

Figures 7-10 depict the iteration that each algorithm's most optimal solution occurred on average within a maximum range of 50 iterations across all files. This allows us to see when the most optimal solution is found for a particular implementation, comparing the personal time frames for each algorithm.

Figures 11-14 depict how quickly each of the algorithms can reach a specified range from the optimal distance across all files on average. The cap was set at 100 iterations to prevent the algorithm from running for an arbitrarily large amount of time. This demonstrates another way to measure the efficiency of each algorithm.

After performing these tests, it is clear that the SAS algorithm proves to generate answers that are closest to the optimal value. Figures 3-6 below demonstrate that across all file sizes, this algorithm consistently performs better than both of the other variants, achieving the closest optimal value on average.

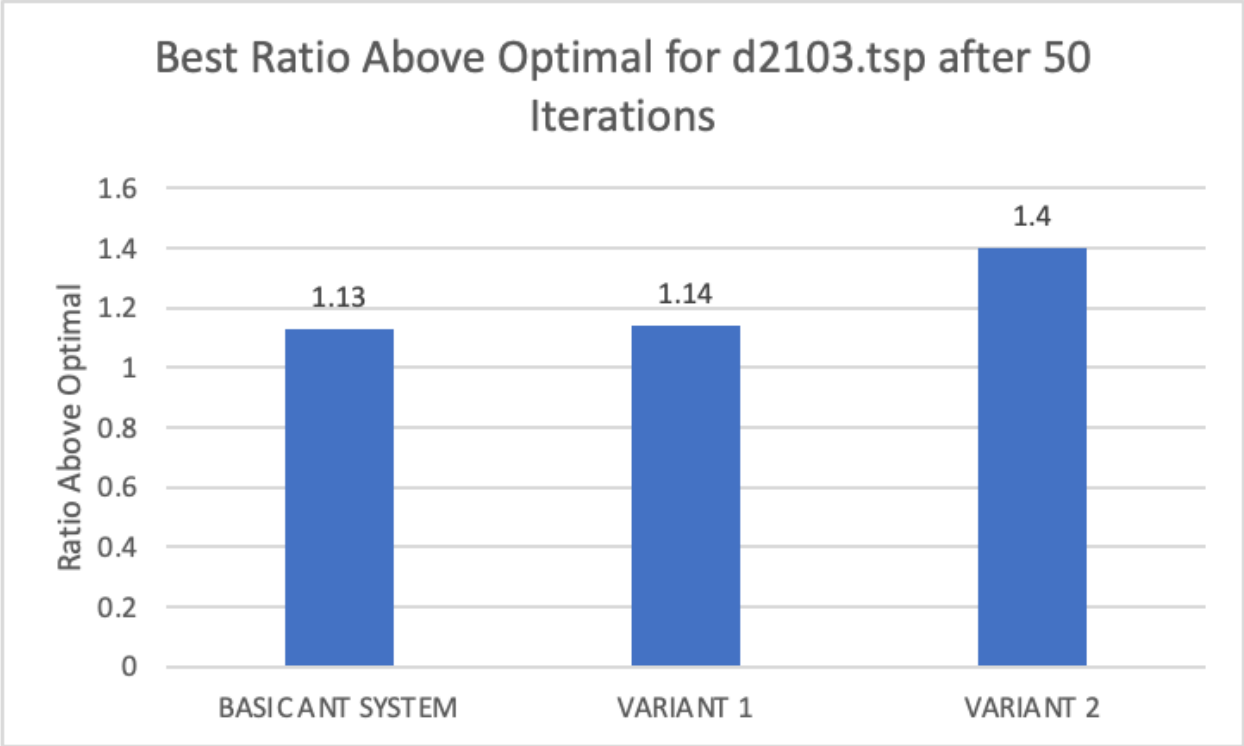

Figure 3: Best tour found divided by optimal solution for file d2103.tsp over 50 iterations

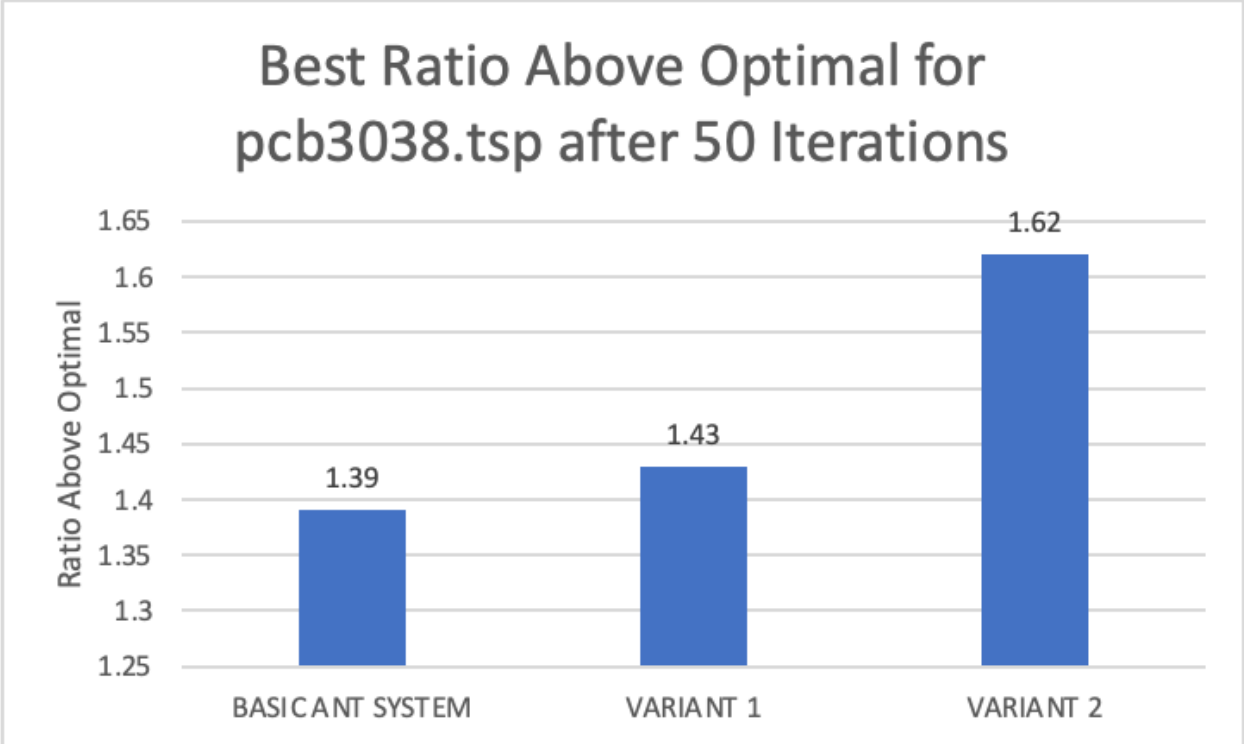

Figure 4: Best tour found divided by optimal solution for file pcb3038.tsp over 50 iterations

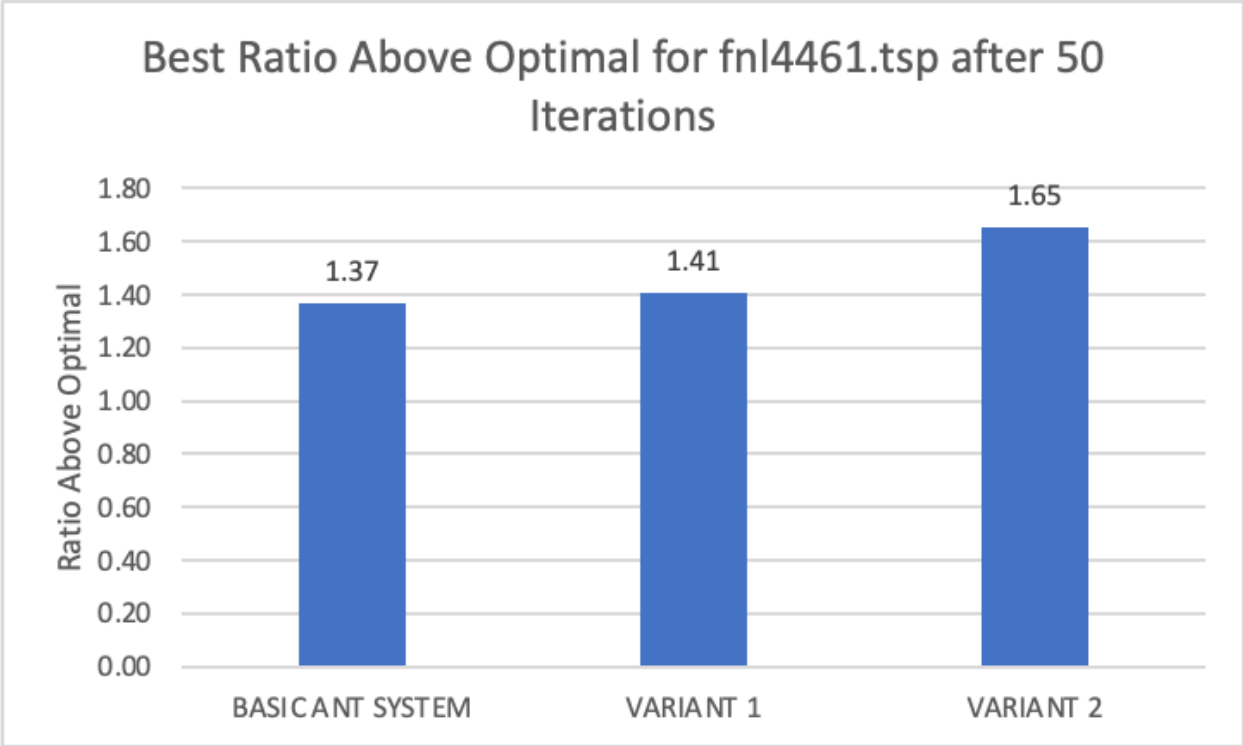

Figure 5: Best tour found divided by optimal solution for file fnl4461.tsp over 50 iterations

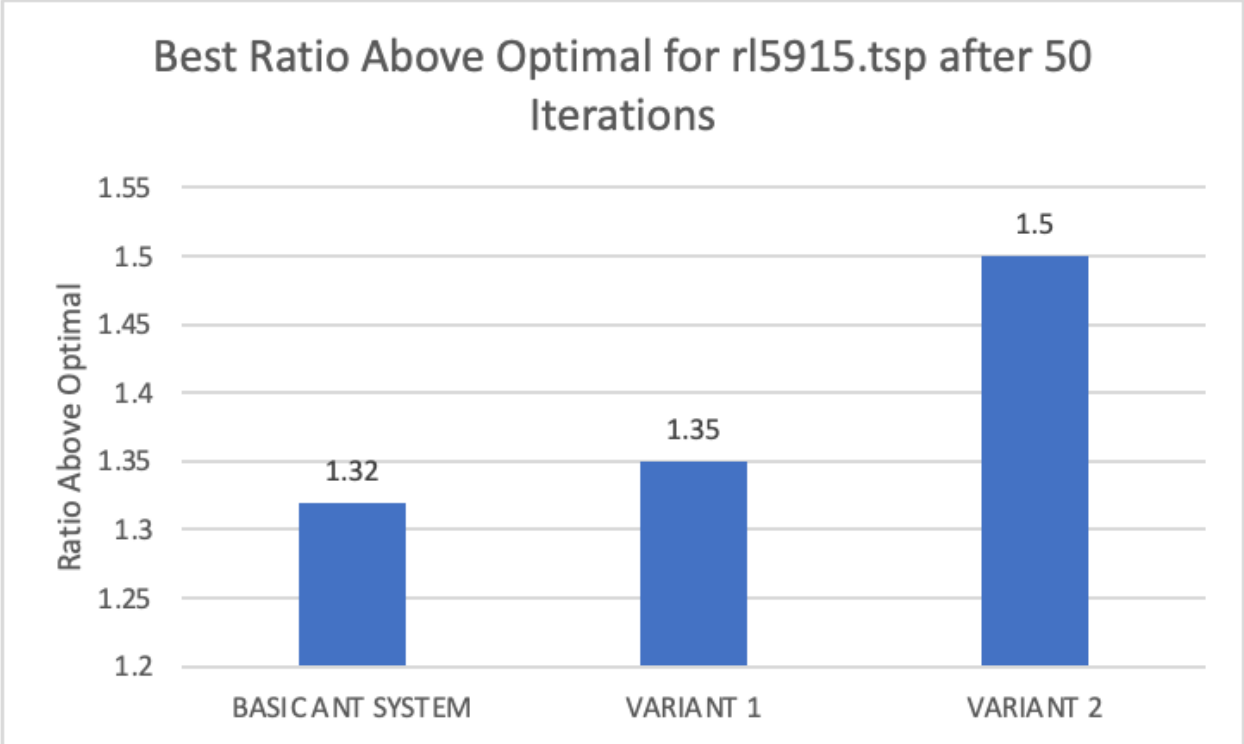

Figure 6: Best tour found divided by optimal solution for file rl5915 over 50 iterations

It is likely that SAS gives the most optimal solution due to the fact that it is a time tested algorithm, and it is difficult to implement something that returns better results. However, it should be noted that, as seen in figures 3-6, ACOV1 (variant 1), was often a close second in performance across all files, as its best tour was only, at worst, 0.04 times larger than that of the SAS. This could likely be due to the fact that it is quite similar in construction to SAS, other than the minor difference in pheromone updating. Perhaps this points to the possibility that there exists better values for an average multiplier and evaporation factor. However, this could also point to the simple fact that this methodology of updating pheromone does not increase performance, but rather only marginally decreases it. ACOV2 on the other hand, performed significantly worse than the other two variants across all file sizes, as its closest distance to ACOV1 occurred in rl5915.tsp, as seen in Figure 6, and was still 0.15 times further from the most optimal. This poor performance demonstrates that it is likely random city selection is undesirable in ACO algorithms, particularly for the TSP, as it clearly does not lead to optimal performance, even when random probability is decreased across iterations.

While SAS did consistently outperform the other algorithms in finding the most optimal tour across 50 iterations, it often found its best tour later than the other algorithms found their own best tours. In Figure 8 and Figure 9 below, we see ACVO1 actually finds its most optimal tour more quickly than SAS does, and below in Figure 10, ACVO2 finds its most optimal tour at the same time as SAS. While it is obvious that this comparison matters significantly less than the actual distance of the most optimal tour from the real optimal solution, it is still interesting to compare the iteration or timeframe for when an algorithm is able to compute its performance. It should also be noted that across all files, SAS, the best performing ACO algorithm, took a minimum of 33 iterations to find its most optimal tour on average, which occurs in d2103.tsp, the smallest file, denoted in Figure 7.

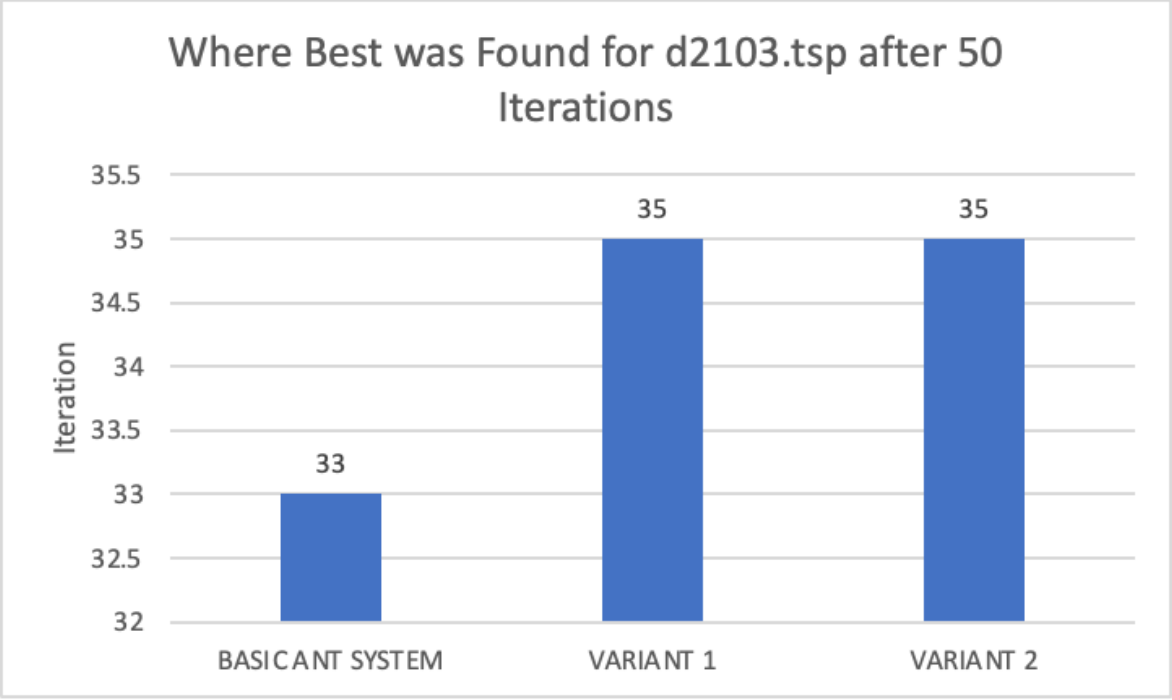

Figure 7: Number iteration that best tour was found for file d2103.tsp. Capped at 50 iterations.

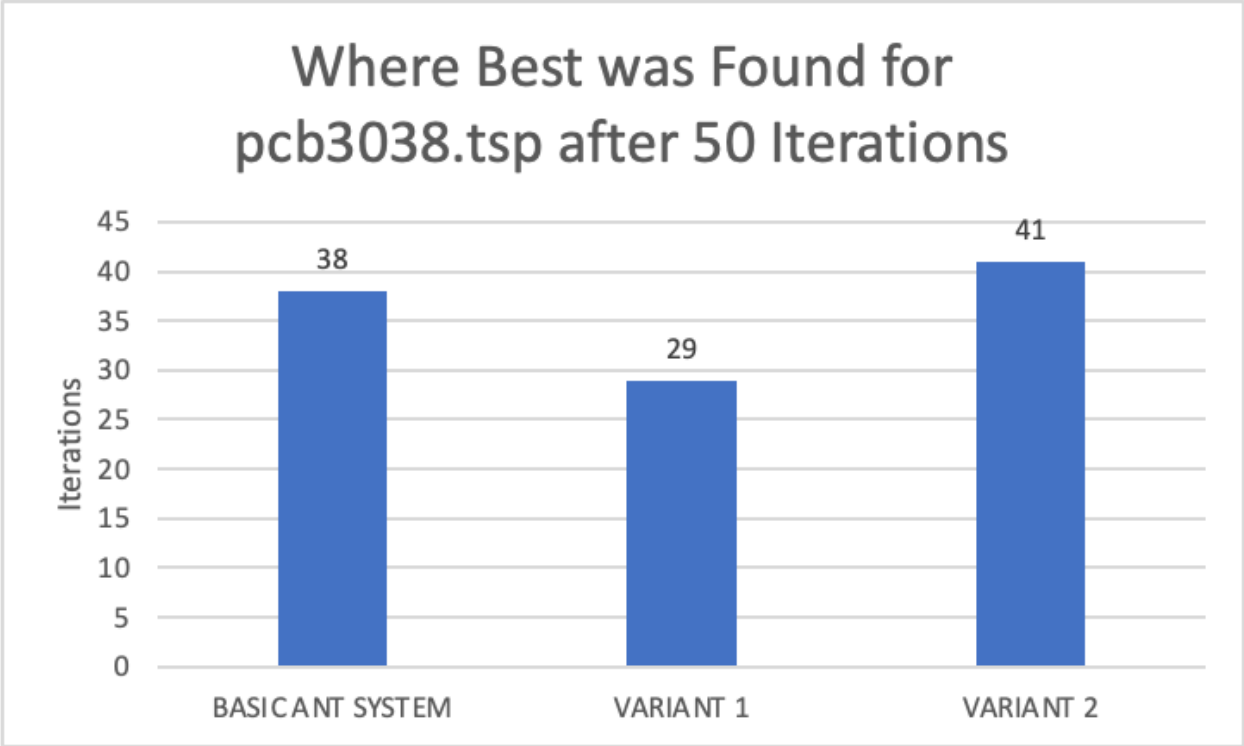

Figure 8: Number iteration that best tour was found for file pcb3038.tsp. Capped at 50 iterations.

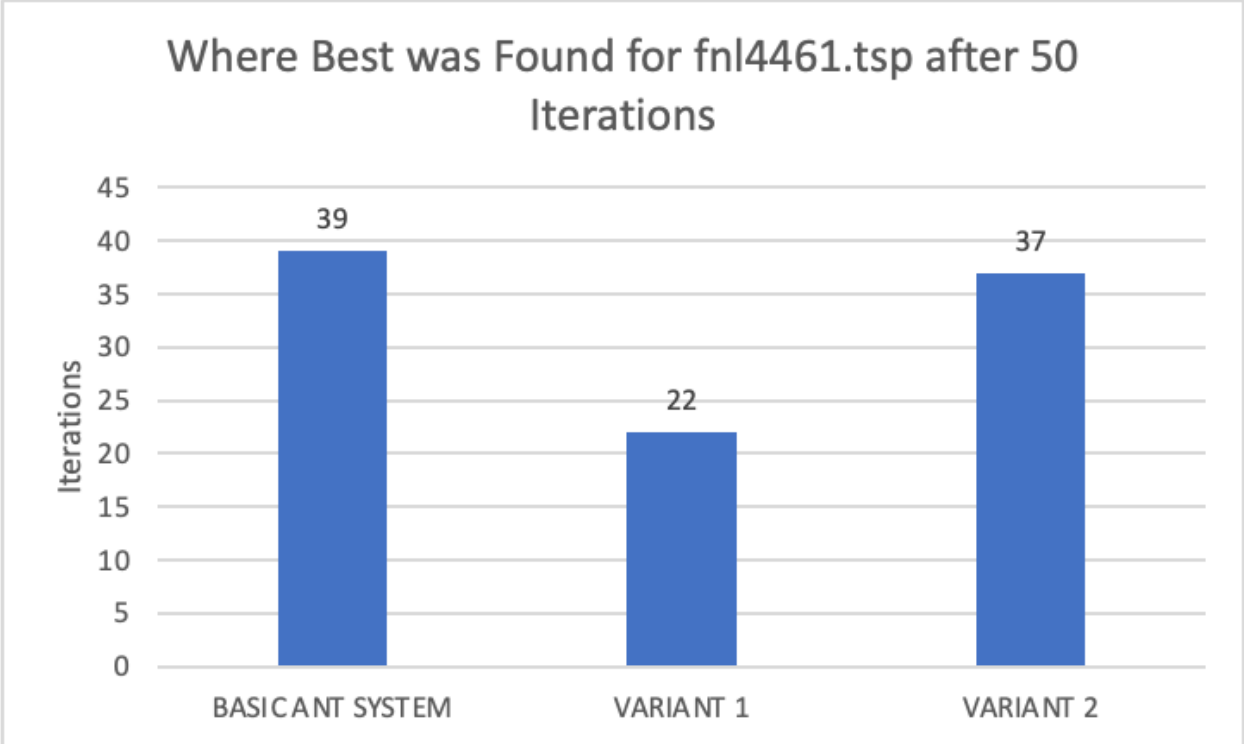

Figure 9: Number iteration that best tour was found for file fnl4461.tsp. Capped at 50 iterations.

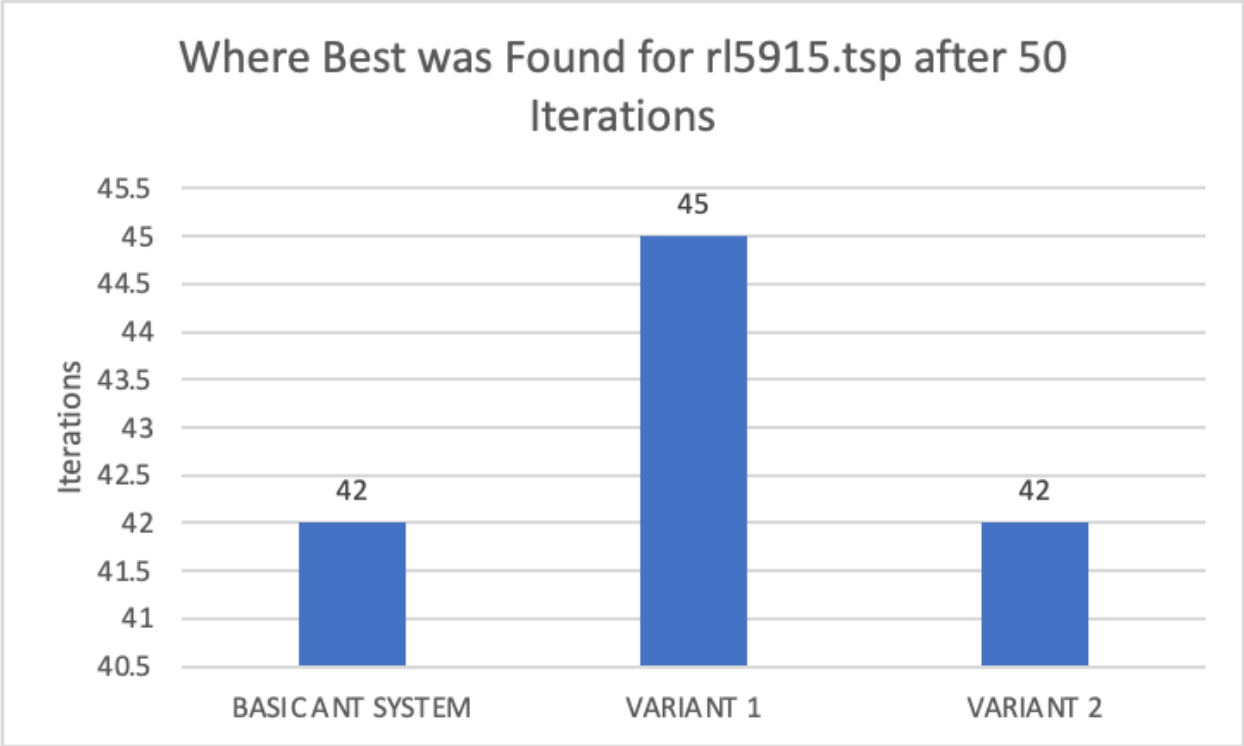

Figure 10: Number iteration that best tour was found for file rl5915.tsp. Capped at 50 iterations.

We are unsure why the iteration the optimal tour is found in varies a fair amount, but it seems like there is a general trend for the algorithm to perform increasingly better for at least the first 25 iterations across all file sizes. However, the differences in optimal tour location could possibly just be due to randomness and variance across file size. There seems to be no clearly defined reason for this to occur.

Figures 11-14 depict the speed at which each algorithm could reach a certain level of optimality. First, when picking the optimality level, we wanted to keep it constant across all algorithms for a particular file, making it easy to compare. We also wanted to have the optimal be a sufficient enough distance that none of the algorithms would reach it within the first few iterations. However, as a side effect to this, ACOV2 never reached the specified optimal range, as seen in Figures 11-14 below. This continues to demonstrate its overall poor performance.

We can also see in the Figures below that, not only did SAS find the most optimal tour, but it also always found it the fastest, outperforming ACVO1 by a significant margin across all file sizes. This demonstrates that while ACVO1 may have been somewhat close in performance to the SAS, it took significantly longer to reach a specified optimal range. Thus we can conclude that the efficiency of SAS is quite a bit higher than that of ACOV1.

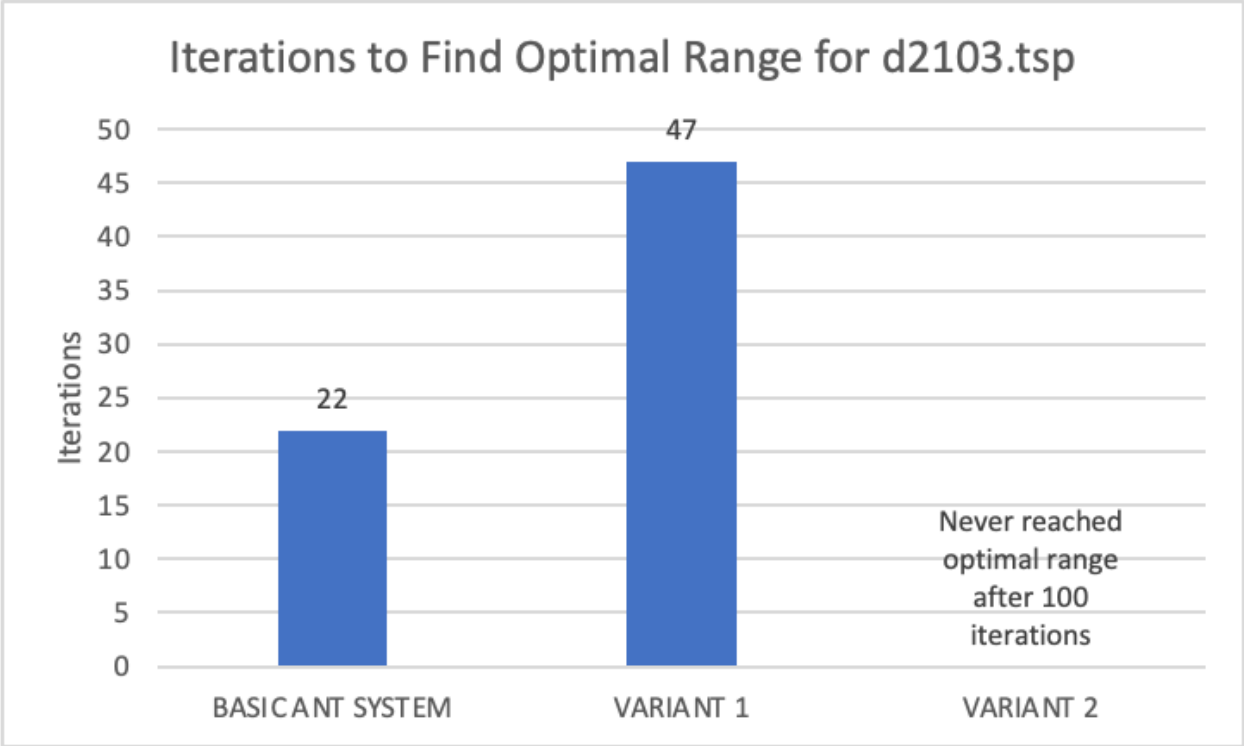

Figure 11: The number of iterations to find specified optimal range (1.15) for file d2103.tsp. Capped at 100 iterations.

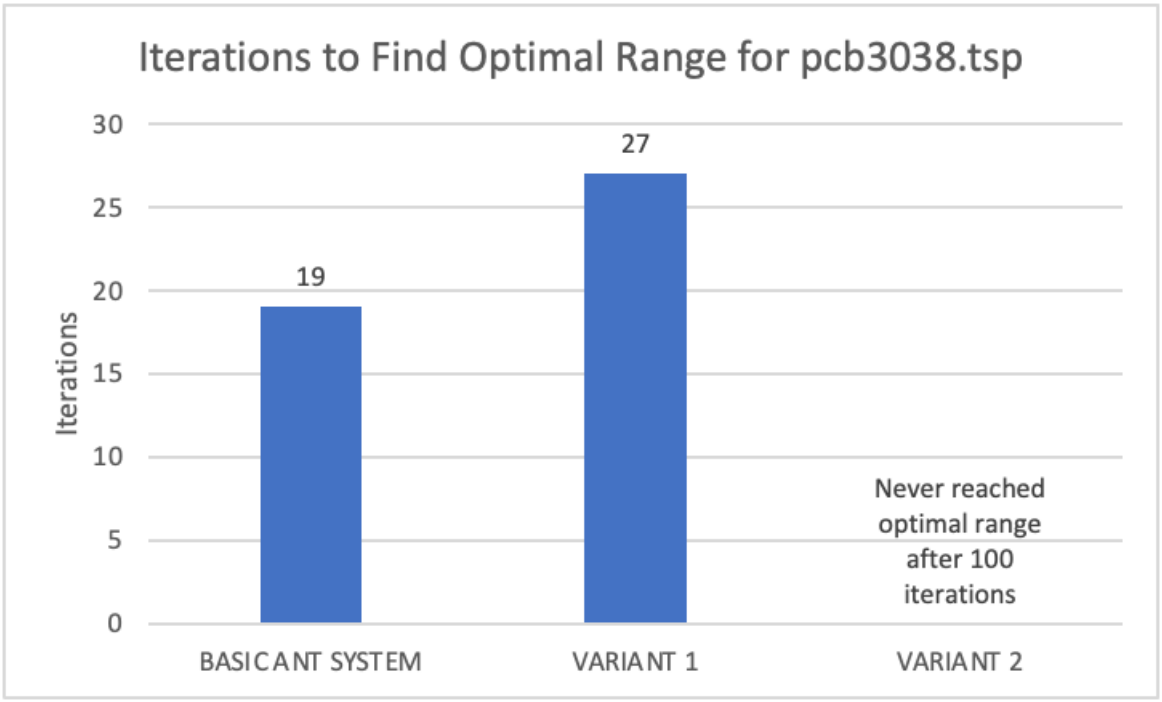

Figure 12: The number of iterations to find specified optimal range (1.43) for file pcb3038.tsp. Capped at 100 iterations.

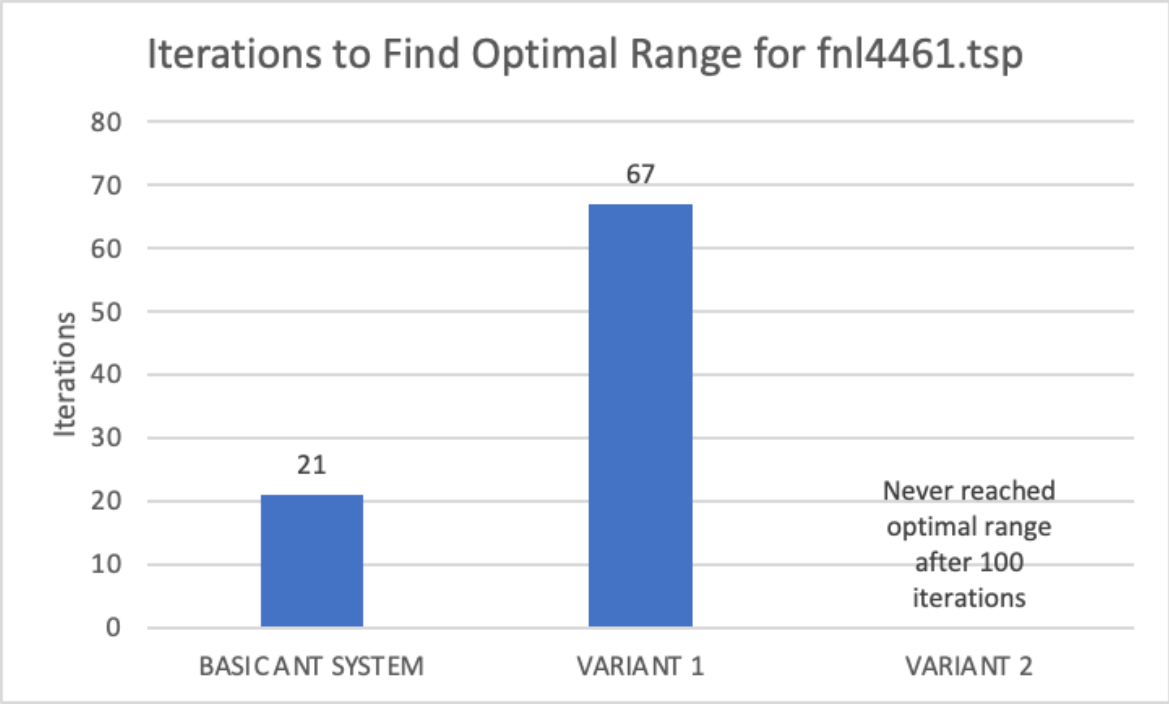

Figure 13: The number of iterations to find specified optimal range (1.40) for file fnl4461. Capped at 100 iterations.

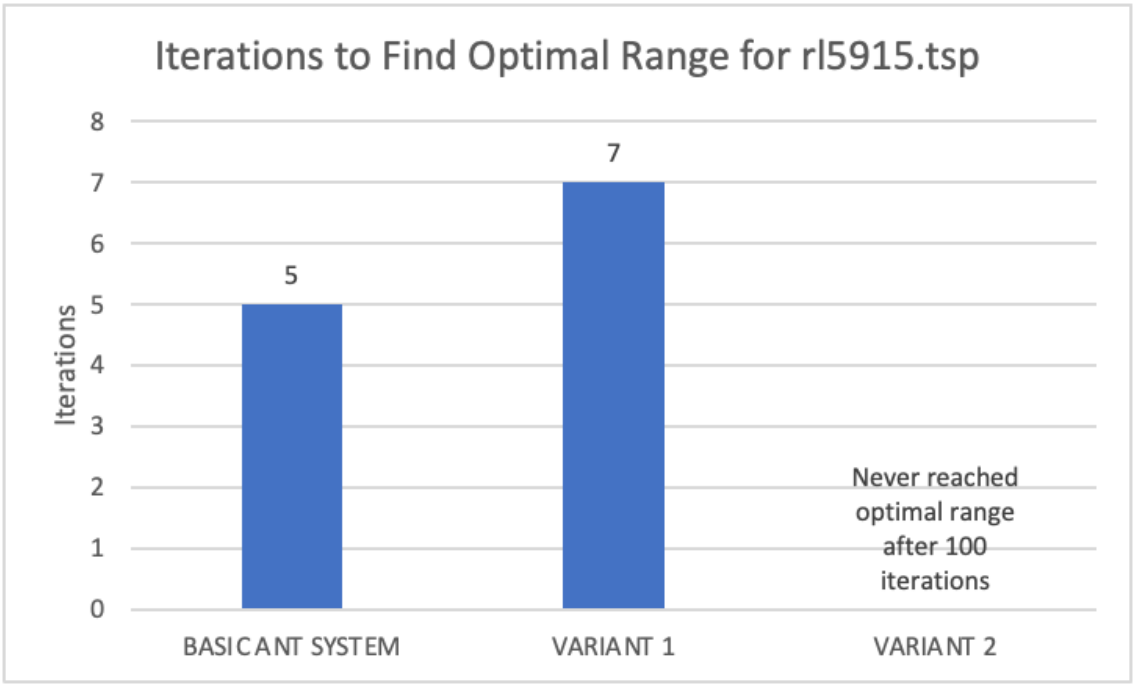

Figure 14: The number of iterations to find specified optimal range (1.43) for file rl5915. Capped at 100 iterations.

We believe this difference in efficiency between SAS and ACVO1 can likely be attributed to the evaporating of pheromones in legs that may be reasonable to work with in ACVO1, thus taking longer for the algorithm to utilize legs that may actually prove useful.

ACVO2 likely performed poorly in this realm because of previously mentioned issues with randomization, and its optimal tour was unable to reach a threshold that was instead reasonable for the other ACO variants.

6 FURTHER WORK

Given more time to work on ACO variants for TSP, we would likely modify the two variants by using different number configurations (i.e. average multiplier, evaporation rate, random probability, number of ants, beta and alpha, etc…). This would allow us to try configurations that might prove to be more optimal than the original ways they are set, and perhaps even more optimal than SAS.

In addition, we would likely develop new variants that could both change other aspects of the ACO process, but also change the same aspects as our current variants but in different ways of course. This would create potential for more optimality and better results. Examples could include implementing randomness but in a different portion of the ACO, depositing or evaporating pheromone in a different manner, or only using best and worst tours to update pheromone.

Finally, we would also take the time to run ACO on more TSP files that vary in size. This could be useful to extract more data and subtle differences between different variants.

7 CONCLUSION

We have presented three ACO algorithms for solving TSP: the Standard Ant System, the first ACO variant that updates pheromone levels every iteration in a slightly different manner, and the second ACO variant that puts an ant through a random tour based on a decreasing probability.

Neither of the variants we introduced outperformed the SAS, as it proved to return the most optimal tour in the fastest speed across all file sizes. However, ACOV1 found a most optimal tour that was only marginally worse than SAS, though this consistently took longer to happen. ACVO2 performed poorly in both optimal tour level and speed in comparison to both SAS and ACVO1.

These findings demonstrate that SAS is a well tested and useful ACO for solving optimization problems, particularly TSP. It also likely shows that variants using random tours result in poor performance, and evaporating particular pheromones on particular legs may not increase efficiency. However, more testing is needed to know this for certain. Overall, we certainly recommend using the SAS amongst these three variants for solving TSP with any number of cities.