John St. Belfield
4/21/2017

Ant Colony Optimization for the Traveling Salesman Problem

Abstract
This work implements two Ant Colony Optimization algorithms and investigates their performance on non-trivial instances of the Traveling Salesman Problem. The algorithms were tested with different parameter values to determine relative performance between both variants with regard to speed and accuracy. It was concluded that Ant Colony performed better than EAS when both were run with "optimal" parameters that we determine.

1 Introduction

Ant Colony Optimization (ACO) is an algorithmic method that imitates the way ants collect food and pass information to other ants on the best path to reach a food source. Given a specified number of ants, the ants in an ACO algorithm progressively build tours of cities, and each successive path to the next city is chosen probabilistically. The probability is adjusted based on the pheromones laid down by the ants, the amount of which is based on the tours. Higher levels of pheromone result in a higher likelihood that an ant will choose that path. Ants that select more efficient paths will ultimately build shorter, and thus better, tours. More efficient paths will receive more pheromone. The pheromone evaporates over time in most implementations, so if the optimal answer changes, or if a better solution is found, ants will eventually stop selecting less efficient paths as the pheromone disappears.

The Traveling Salesman Problem (TSP) consists of some number of cities and the distances between those cities. The solution is the shortest route between all the cities that includes every city only once, and returns to the original city. This problem works well with ACO because each city acts as a node that an ant stops at, and pheromones can be laid down on the paths between the cities.

In this work, we implemented two different types of ACO algorithms. We implemented Elitist Ant System and Ant Colony System. Through observation of the effect of modifying several key parameters while keeping some fixed, we attempted to discover the properties of those parameters that optimized each particular algorithm, and ultimately offer a judgement of the superior algorithm.

We found that ACS has two sets of optimal parameter values. One set generally conforms to rule-of-thumb settings, while the other has parameters that limit the explorative nature of ACS. Ultimately, both sets perform better than EAS, both in speed and in accuracy.

Section 2 describes the Traveling Salesman Problem. Section 3 offers a general overview of Swarm Intelligence and ACO algorithms, and their motivation. Section 4 provides an in depth analysis of the general structure of an ACO algorithm. Specific details regarding the Elitist Ant System and the Ant Colony System are provided in Sections 5 and 6, respectively. Experimental design is given in Section 7, and Section 8 describes the results. Section 9 gives possible further work and Section 10 gives the conclusion.

2 Traveling Salesman Problem

The Traveling Salesman Problem essentially asks the following question: “Given an assortment of nodes and their pairwise distances, what is the tour of the shortest length that includes all nodes?”

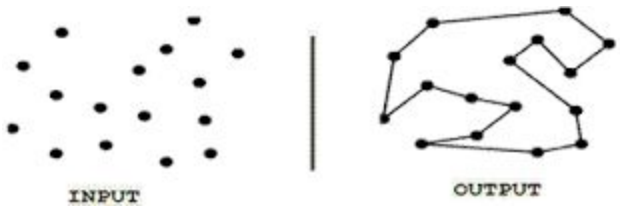


Figure 1: Example TSP. The dots represent “cities,” and the output illustrates a sample solution.

In our work, the ACO algorithms were tested on instances of the symmetric TSP problem. In the symmetric TSP problem, the distance between any two cities is the same in both directions. The TSP is a widely known and studied problem, and is thus used as a baseline of comparison for different optimization methods. The problem has also been applied in its form or to solve sub-problems in areas such as logistics and DNA sequencing.

In essence, the TSP problem is significant in that the methods used to find optimal solutions can be extrapolated to an incredible variety of problems. Many optimization problems can be represented in a form similar to that of the TSP, allowing successful TSP optimization algorithms to be adapted to many other applications.

3 Swarm Intelligence: Ant Colony Optimization

The principle motivation behind swarm intelligence is the idea that a group of individuals can achieve a collective performance which could not normally be achieved by an individual acting alone. This collective interaction via indirect communication can lead to good solutions; in fact, such a collective system is capable of accomplishing difficult tasks in dynamic and varied environments without any external guidance or control and with no central coordination. This results in a natural model particularly suited to distributed problem solving.

In Ant Colony Optimization algorithms, “ants” work concurrently and independently, imitating the real-world manner in which ants forage for food. There are many advantageous inherent features of ACO: inherent parallelism, stochastic nature, adaptivity, use of positive feedback, autocatalytic in nature.

A particularly unique aspect of ACO algorithms is *stigmergy*, a mechanism for binding task state information to local features of a task site. Stigmergy facilitates communication by modifying those features; thus, the environmental modification serves as an external memory, which allows work to be continued by any individual.

ACO has applications in a wide variety of problems, but it is especially suited to those that have a distinctly combinatorial quality, such as scheduling, routing, and set problems. Other applications include the quadratic assignment problem and protein folding.

The primary advantage of ACO is its suitability to dynamic applications. Such suitably arises from many qualities: a single ant does not need to find best solution, ACO is less affected by poor initial solutions, its greedy heuristic finds good solutions early, and it is less likely to converge prematurely.

Unfortunately, ACO's main disadvantage is nontrivial: its problem representation is somewhat restrictive. As illustrated in the possible applications, the problems must generally be combinatorial in nature.

4 Ant System General Structure

Regarding the general structure of an ACO algorithm, problems are solved by repeating the following steps. First, candidate solutions are constructed using a pheromone model, i.e. a parameterized probability distribution over the solution space. Then, these candidate solutions are used to modify the pheromone values in a way that is intended to bias future sampling toward high quality solutions.

Specifically, after the initialization of pheromones, ACO algorithms repeat the following steps as desired: possible iteration stop conditions include a discrete number of maximum iterations, a stop after progress fails to continue for a certain number of iterations, or once a solution reaches a certain threshold of optimality/i.e. Gets within range of the optimal solution; however, this requires knowing the optimal solution beforehand. First, ants iteratively build solutions randomly based on heuristic information, but preferring legs with stronger pheromone concentrations. Thus, the pheromone levels are the principal guide for the building of tours. Constructed solutions are used to update the pheromones: when ant finishes building a tour, pheromones are deposited on the legs of the tour. The quantity of pheromone deposited depends on the length of the tour, and pheromones diffuse over time, so earlier "mistakes" can be forgotten.

Additional optional actions are then performed, depending on the specific choice of ACO algorithm. For example, some variants perform local search. Ultimately, legs that consistently appear in low-cost solutions are likely to be legs in the optimal solution.

Specifically, regarding the construction of solutions: for each individual, solutions are built probabilistically. Given a decision point i that presents a choice between paths a and a', the probability of choosing branch a is specified by:

$$p_{ia} = \frac{(k + \tau_{ia})^{\alpha}}{(k + \tau_{ia})^{\alpha} + (k+\tau_{ia'})^{\alpha}}$$

where τ_{ia} is the pheromone concentration on path *a* at decision point *i*. *k* and α are positive, real constants.

However, ants are generally presented with more than two options. Thus, the previous equation for the stochastic solution construction must be expanded as such. Note that a solution is constructed by starting at a random city, and adding cities until the tour is complete. Thus, given the current city *i*, ant *k* chooses the next city *j* from among those not yet visited with probability:

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

where τ is the pheromone concentration on the leg connecting cities *i* and *j*; $\eta_{ij} = \frac{1}{d_{ij}}$, where d_{ij} is the distance between cities *i* and *j*; α, β are positive, real constants that determine how heavily pheromone and city distance are weighted when determining probabilities; and N_k is the set of unvisited cities of ant *k*.

Tours are evaluated, and pheromone is deposited in the following manner:

If path (i,j) is not on the tour of ant *k*, $\Delta\tau_{ij}^{k} = 0$.
If $(i,j) \in$ tour of ant *k*, $\Delta\tau_{ij}^{k} = \frac{1}{L_k}$, where L_k is the length of ant *k*'s tour.
Note that legs in shorter tours get more pheromone.
It follows that $\Delta\tau_{ij}^{total} = \sum_{k=1}^{N} \Delta\tau_{ij}^{k}$, where *N* is the number of ants
Along with the addition based on legs of solutions proportional to the solution fitness, the pheromone update includes a phase to allow pheromones to evaporate. Given an evaporation factor ρ, where $0.0 < \rho < 1.0$:

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

Following the pheromone update, the current iteration is complete. The best tour is saved, and the process is repeated until stagnation, or after the threshold for maximum iterations is reached.

5 Elitist Ant System (EAS)

Developed in 1992, EAS introduces new pheromone update rules:

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

where $0.0 < \rho \leq 1.0$; *e* is the elitism factor, which is typically equal to the number of ants; *bsf* represents the best tour so far; and $\Delta\tau_{ij}^{k}$ is the same as before.

In the pheromone depositing phase:
$\Delta\tau_{ij}^{bsf} = 0$ if (i,j) not in *bsf*.
$\Delta\tau_{ij}^{bsf} = \frac{1}{L^{bsf}}$ if $(i,j) \in$ bsf, where L^{bsf} is the length of *bsf*.

The new update rules in Elitist ant system were intended to make the algorithm more greedy or exploitative. It would cause the best path so far to be more influential in drawing ants to choose that path by modifying the pheromone amounts.

For EAS, accepted rule-of-thumb parameters are α is 1, β is 2 to 5, ρ is 0.5, the number of ants is equal to the number of cities, and τ initial is m/C^{nn}, which is the heuristic found by taking the tour length found by having the ants go to the nearest city from the city they are at.

6 Ant Colony System (ACS)

Developed in 1996, this variant **e**xploits search experience more strongly. Pheromone changes only on legs of the "Best-So-Far" tour. Pheromones are then "worn away" when ants traverse legs during tour construction.

Search experience is exploited more strongly using the randomized proportional action selection rule, resulting in a more deterministic algorithm. With probability q_0, typically equal to 0.9, ant *k* in city *i* chooses the next city *j* that it has not yet visited, where *j* maximizes $\tau_{ij}\eta_{ij}^{\beta}$. Otherwise, the next city is chosen according to the action choice rule of the general ant system.

The "Best-So-Far" is subsequently reinforced more strongly by increasing pheromone only on the *bsf* as such:

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

The pheromone deposit rules are identical to the EAS:
$\Delta\tau_{ij}^{bsf} = 0$ if (i,j) not in *bsf*.
$\Delta\tau_{ij}^{bsf} = \frac{1}{L^{bsf}}$ if $(i,j) \in$ bsf, where L^{bsf} is the length of *bsf*.

To increase exploration, ACS reduces the attraction of frequently used edges. Pheromone concentrated is limited by wearing away pheromone on edges crossed by ants:

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

where $0.0 < \varepsilon < 1.0$ (typically, $\varepsilon = 0.1$); τ_0 is some small constant.

For ACS, accepted rule-of-thumb parameters are β is 2 to 5, ρ is 0.1, m (number of ants) is 10, and τ initial is $1/nC^{nn}$, which is the heuristic found by taking the tour length found by having the ants go to the nearest city from the city they are at.

7 Experimental Design

We divided our work into three parts: in Part I, we examined EAS, investigating the role of the parameters $\alpha, \beta, \rho,$ and e. In particular, we examined the individual impact in performance of variations in alpha, beta, the evaporation factor, and the elitism factor. For each parameter, while

fixing all other parameters constant to the prescribed "rule-of-thumb" values found in prior research (Dorigo and Majercik), we performed several waves of trials that considered incremental differences of the isolated parameter in question. The increments for each progressive wave depended on the previous wave; specifically, the choice of values for subsequent waves were drawn from the interval of values that returned the best performance in the previous wave. In Part I, the performance of a value was dependent solely on the accuracy of the resulting solution.

Part II repeated the same process outlined in Part I but for ACS. We investigated $\alpha, \beta,$ and ρ, while also analyzing the impact of ε, the wear factor, and q, the action selection probability.

Part I and Part II provided us with a set of optimal values for the parameters we investigated for both EAS and ACS, respectively. In Part III, using these values for the parameters, we directly compared the performance of EAS and ACS. In Part III, performance not only included the accuracy of the solution, but also the speed of the algorithm. The consideration of speed can be interpreted in two manners: the first examines how quickly the algorithm converges to a final solution, and the second examines how quickly the algorithm actually runs through iterations.

For Parts I and II, we performed our analysis on d2103, a symmetric TSP problem with 2103 nodes. For Part III, we performed our analysis on pcb3038, a symmetric TSP problem with 3038 nodes. In essence, we used the d2103 problem to narrow down the specific parameter values that optimized EAS and ACS. After finding "optimal" parameter values, tests were run on pcb3038 to directly compare the two algorithm variants.

We also ran general tests using the "rule-of-thumb" parameter values on larger problems, fnl4461 and rl5915, which are of size 4461 and 5915 respectively.

We didn't test more problems because two thousand cities gave sufficient variation to see differences between the parameters. Also, testing a multitude of parameter configurations on the larger problems would have taken days or longer. By testing on a smaller problem, we believe we can safely extrapolate our results to larger ones. In the few tests we did do on larger problems, this hypothesis was supported.

Throughout our work, we fixed the number of ants to 20. The number of ants was not varied because they too heavily affected the running time to discover meaningful variations within the number of ants. Once the number of ants starts to approach the number cities, the run time is simply too long on the machines we were using to gather data.

We also fixed the number of iterations at 750: while debugging our code, this seemed like a safe number, as some of the algorithm variants still improved on iterations 600-700. It also offered reasonable testing times.

In Parts I and II, we ran one to three trials for each set of unique parameter values. While this may seem like a low number, we observed during our debugging of the actual code that the

solution scores returned were nearly identical across each trial. In addition to the long run time of every problem, it made it impractical and less necessary to run more trials.

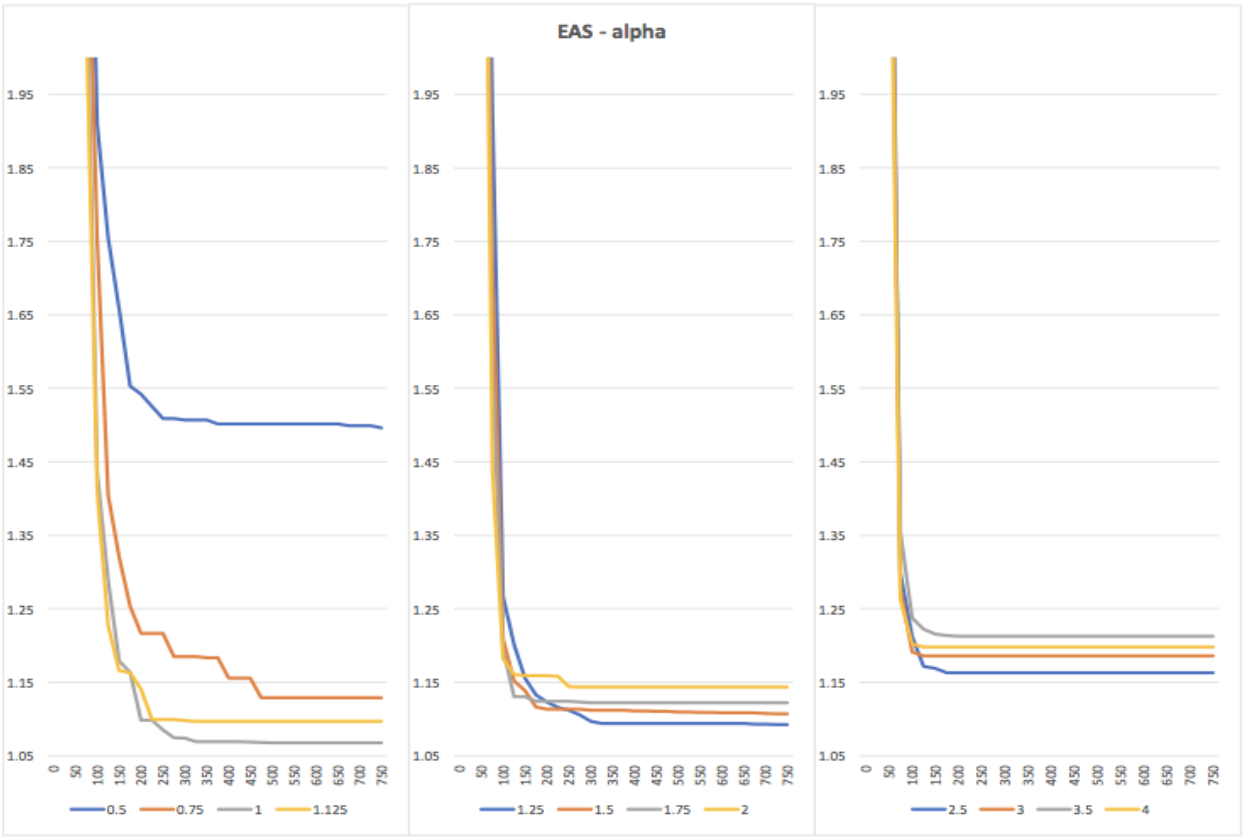
We only recorded times for the larger problems, since the larger problems were where we compared EAS to ACS; within each variant, the parameters we changed had little effect on the running time (none at all, or by a couple seconds).

Throughout our experiments, we kept the number of ants constant at 20. For EAS, the literature recommends the number of ants to be the same as the number of cities--however, in practice, such a value results in an algorithm unsuitable for testing, because of the incredibly long run time.

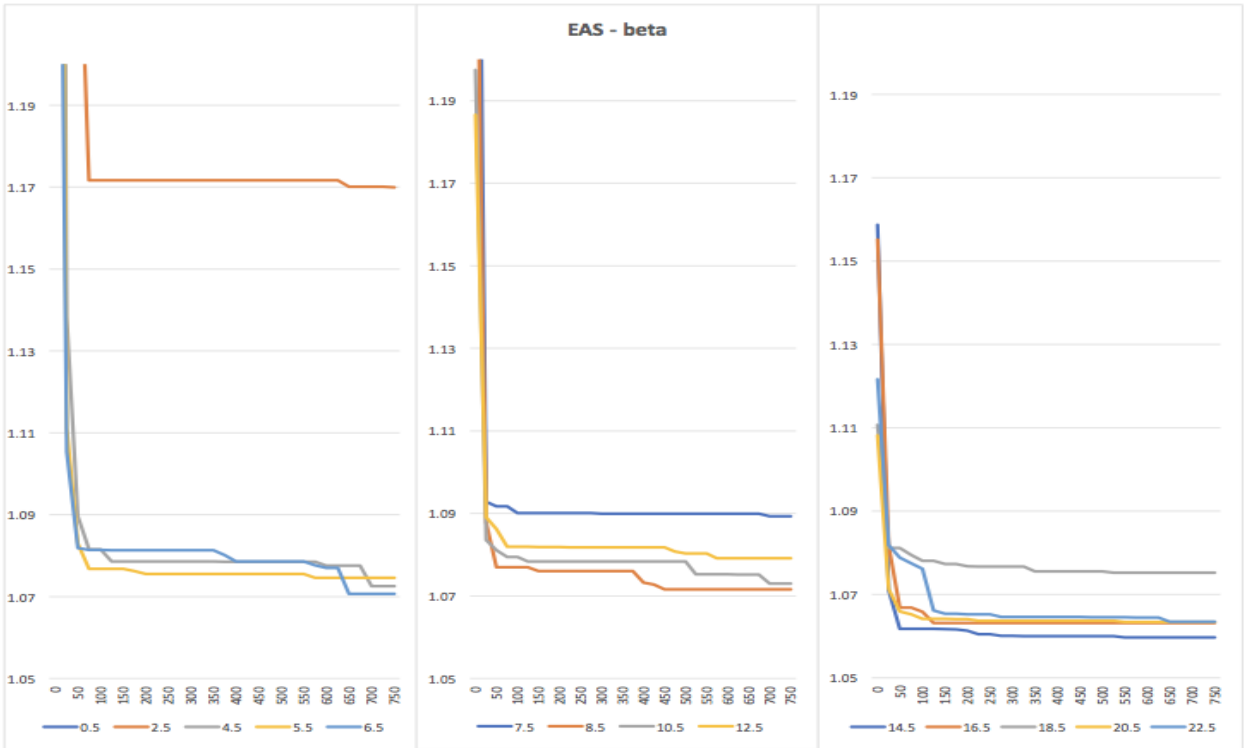
The tests were ran on Multiple computers with the same specifications. They were imacs with a 4 GHz Intel Core i7 processor that were running OS X EL Capitan version 10.11.6. The code was written in Java. The tests were run on multiple computers due to the extensive run times of each test.

8 Results

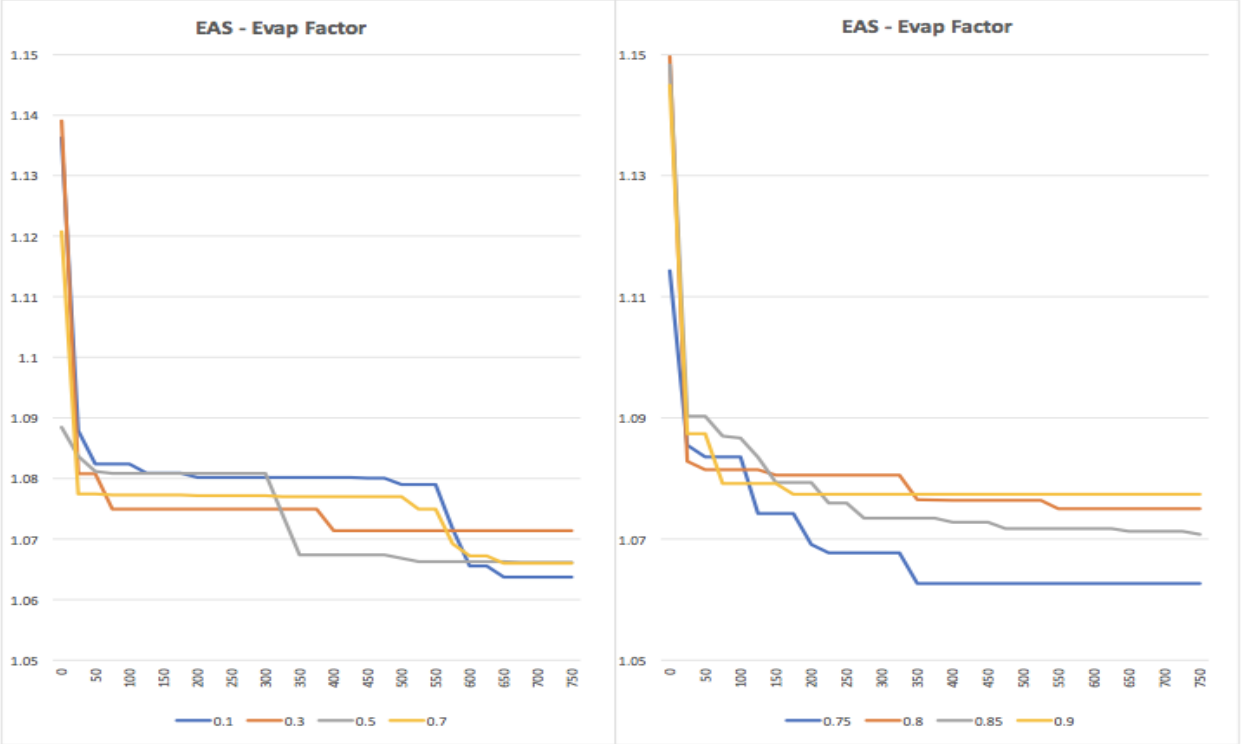
Note that all of the solution scores reported in our work are in terms of a solution ratio: $\frac{solution\ score}{optimal\ score}$. Thus, the lower the score, the better the performance, the lowest score being a one. In the following graphs, the y-axis is the solution ratio, and the x-axis is the number of iterations.



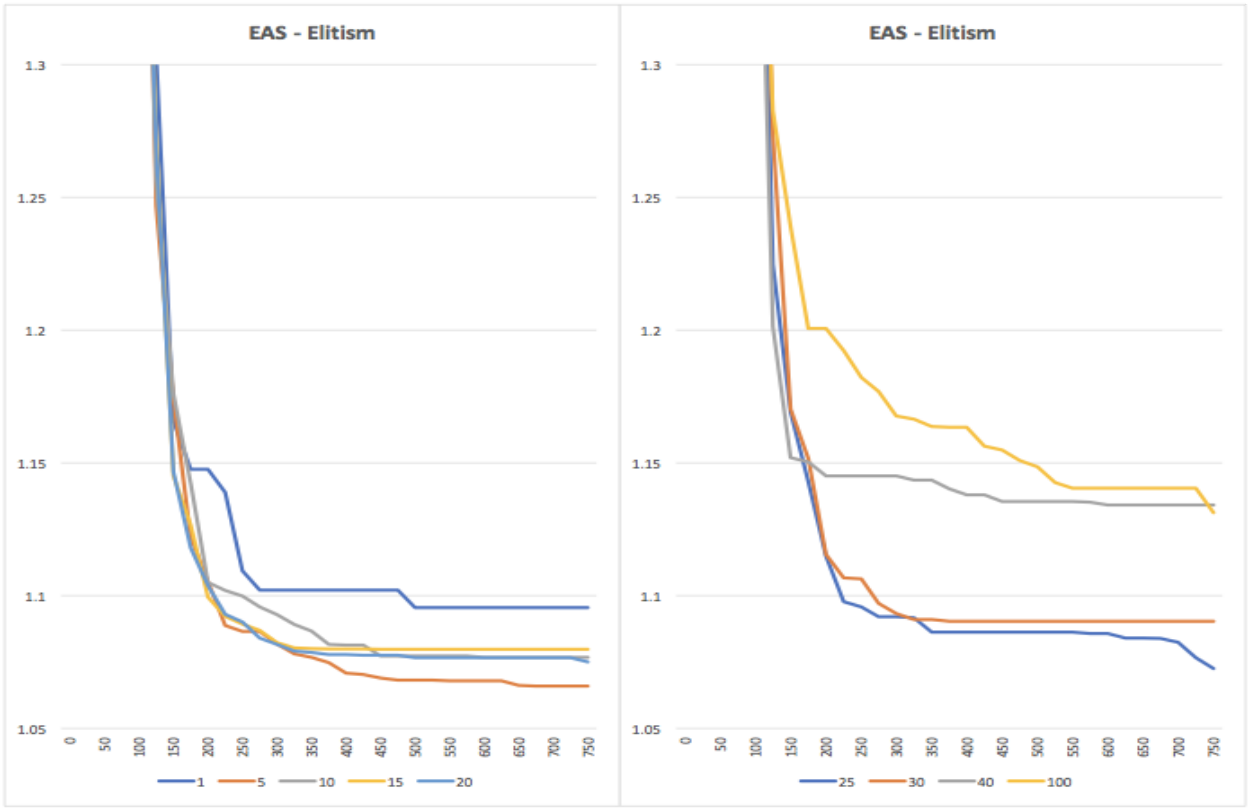
Graph 1: EAS, comparison of ratio of best found solution to optimal solution versus iteration number as alpha varies (d2103.tsp)



Graph 2: EAS, comparison of ratio of best found solution to optimal solution versus iteration number as beta varies (d2103.tsp)



Graph 3: EAS, comparison of ratio of best found solution to optimal solution versus iteration number as the evaporation factor varies (d2103.tsp)



Graph 4: EAS, comparison of ratio of best found solution to optimal solution versus iteration number as elitism factor varies (d2103.tsp)

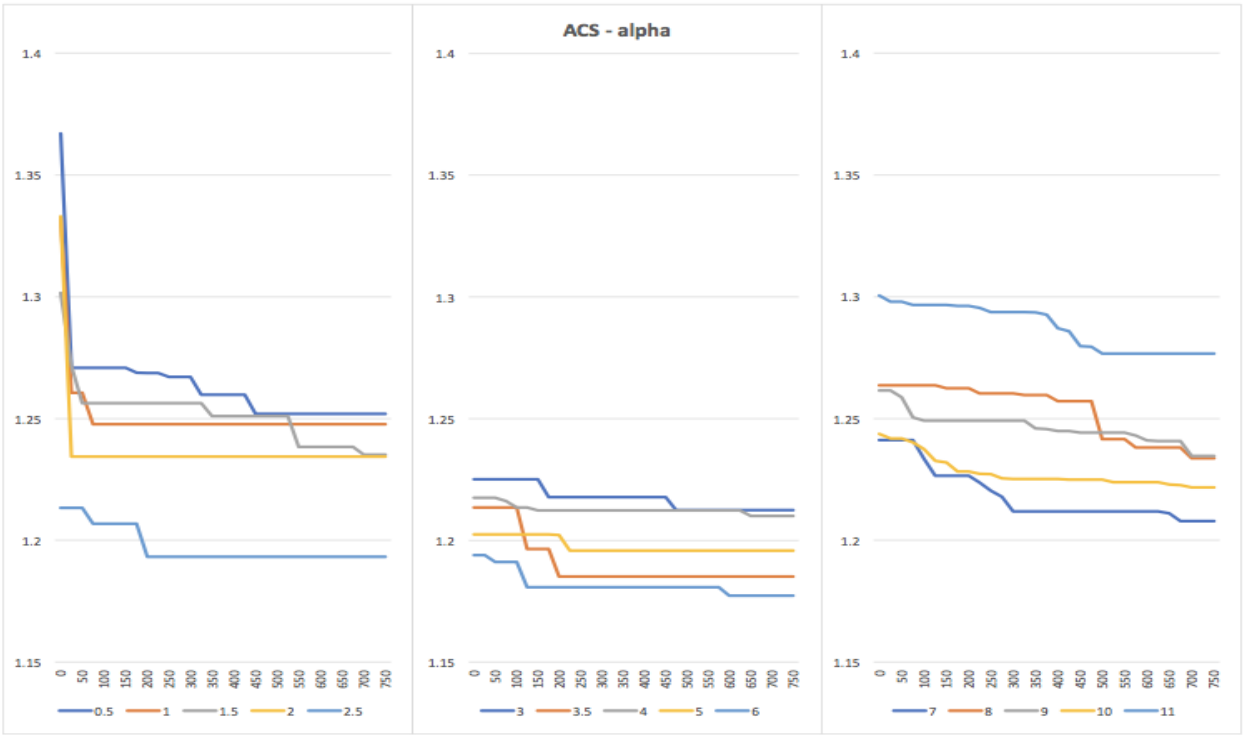
Ultimately, the optimal parameters we discovered for EAS generally adhere to the parameter range recommended by the literature. We found optimum values to include the following (as evidenced by Graphs 1, 2, 3, and 4 above): an alpha α of 1, an evaporation factor ρ between .1 and .5, and an elitism factor e equal to the number of ants. However, optimal beta β values had to at least be 4. These graphs also all showed a pattern of rapid improvement over the early number of iterations and a leveling off as the calculated solution approached the optimum. This result makes sense, as the number of solution available to increase accuracy decreases as the optimal solution is approached, but the rapid rate at which the solution is approached at the after not very many iterations was unexpected.

For smaller problems, the range of optimal values for the parameters α, β, ρ, and e was fairly broad. Essentially, the choice of values did not matter too much as much as long as they lied within a specific range; the values wouldn't produce the best results, but the solutions would still fall very close to the optimum values. For example, on d2103, β values of nearly 20 worked fine; in fact, they often gave better results than a β of 4.5. However, β values that were too high (i.e. 50) were unsuitable, and broke the algorithm; in fact, the larger the problem, the smaller the range of optimal β value.

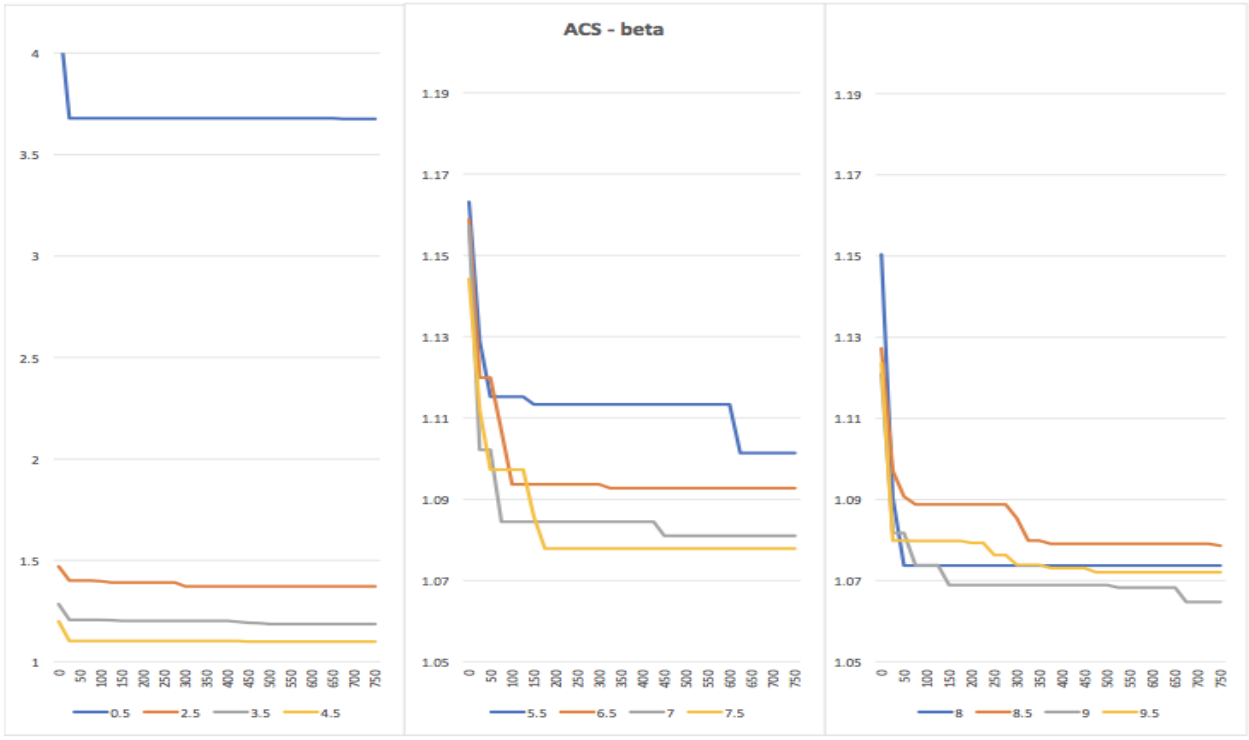
Note that beta β is only accessed during the algorithm once, during the initialization of the heuristic matrix. Specifically, β is the power to which the inverse of the nearest neighbor heuristic is raised, to generate the replicated entry of heuristic matrix. Note that the heuristic is generally some factor of $\frac{1}{C^{nn}}$, where C^{nn} is the length of a nearest neighbor tour beginning at an arbitrary node, and the factor can be as high as the number of cities plus the number of ants. Thus, larger problems will usually have larger distances, meaning that the generated heuristic is smaller. Note that the heuristic is also less than 1, and that powers of numbers less than one are less than the number itself. Thus, high beta values on large problems break the algorithm: because a double can generally only represent 16 decimal places, when the values of the pheromones sometimes drop below $1*10^{-16,}$ they are unfortunately rounded to zero.

Thus, for larger problems, the α, β, and ρ have upper bounds; too high a value, and they risk breaking the algorithm. The argument for why alpha and the evaporation factor have upper bounds is the same as above.

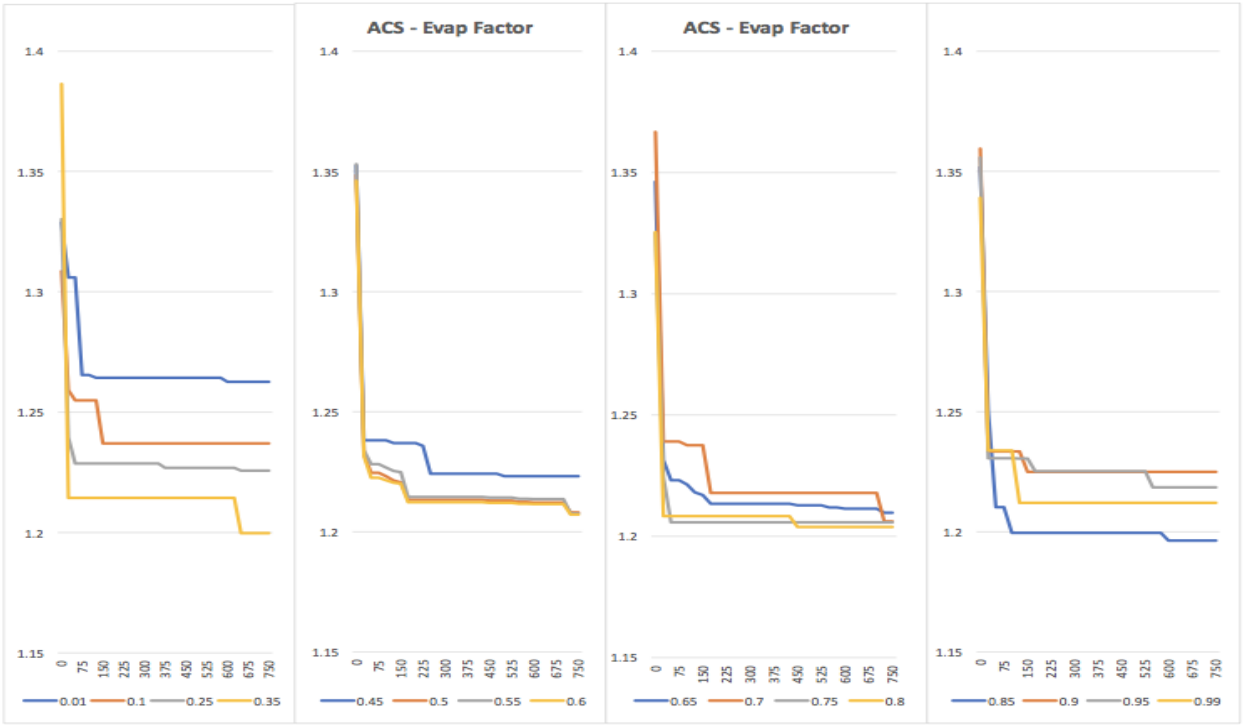
In summary, the ultimate combination of optimal values for EAS were as follows: alpha of 1, beta of 6, elitism of 5, and an evaporation factor of .1. Note that in our initial experiments, evaporation factors of .5 and .75 generated optimal results as well; however, in combination with the beta of 6, .1 provided the best results, while .75 resulted in a failed algorithm. We define an optimal value as the one that approaches the closest to the optimal solution when comparing the values as a ratio to of the optimal solution to the best found solution, because all the answers approached the answers after relatively the same number of iterations.



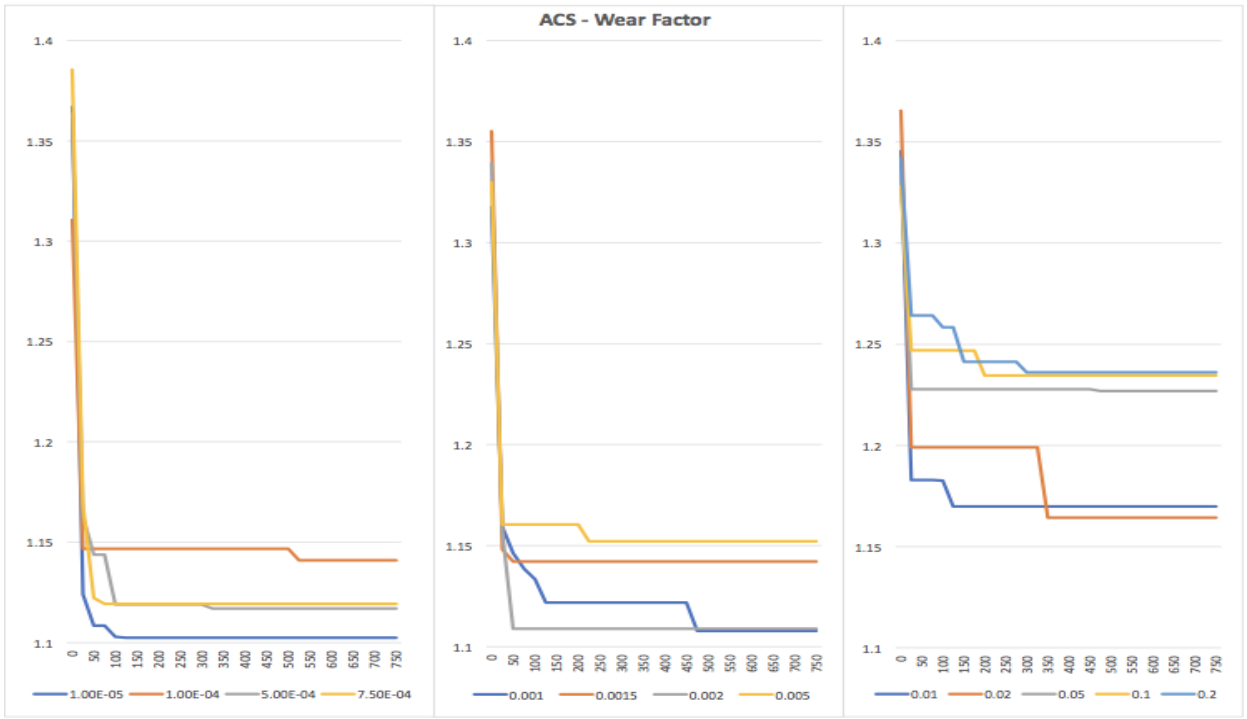
Graph 5: ACS, ACS, comparison of ratio of best found solution to optimal solution versus iteration number as alpha varies (d2013.tsp)



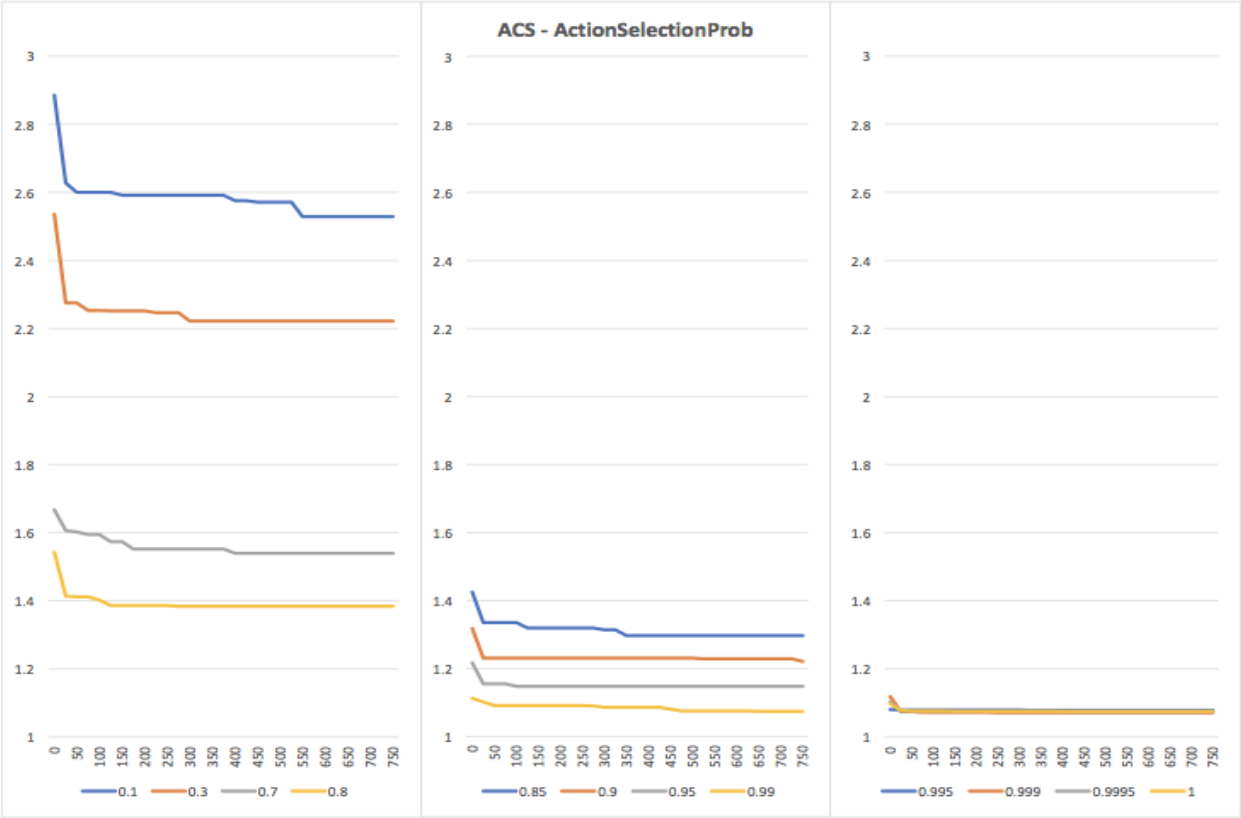
Graph 6: ACS, comparison of ratio of best found solution to optimal solution versus iteration number as beta varies (d2103.tsp)



Graph 7: ACS, comparison of ratio of best found solution to optimal solution versus iteration number as beta varies (d2103.tsp)



Graph 8: ACS, comparison of ratio of best found solution to optimal solution versus iteration number as the wear factor varies (d2103.tsp)



Graph 9: ACS, comparison of ratio of best found solution to optimal solution versus iteration number as the action selection probability varies (d2103.tsp)

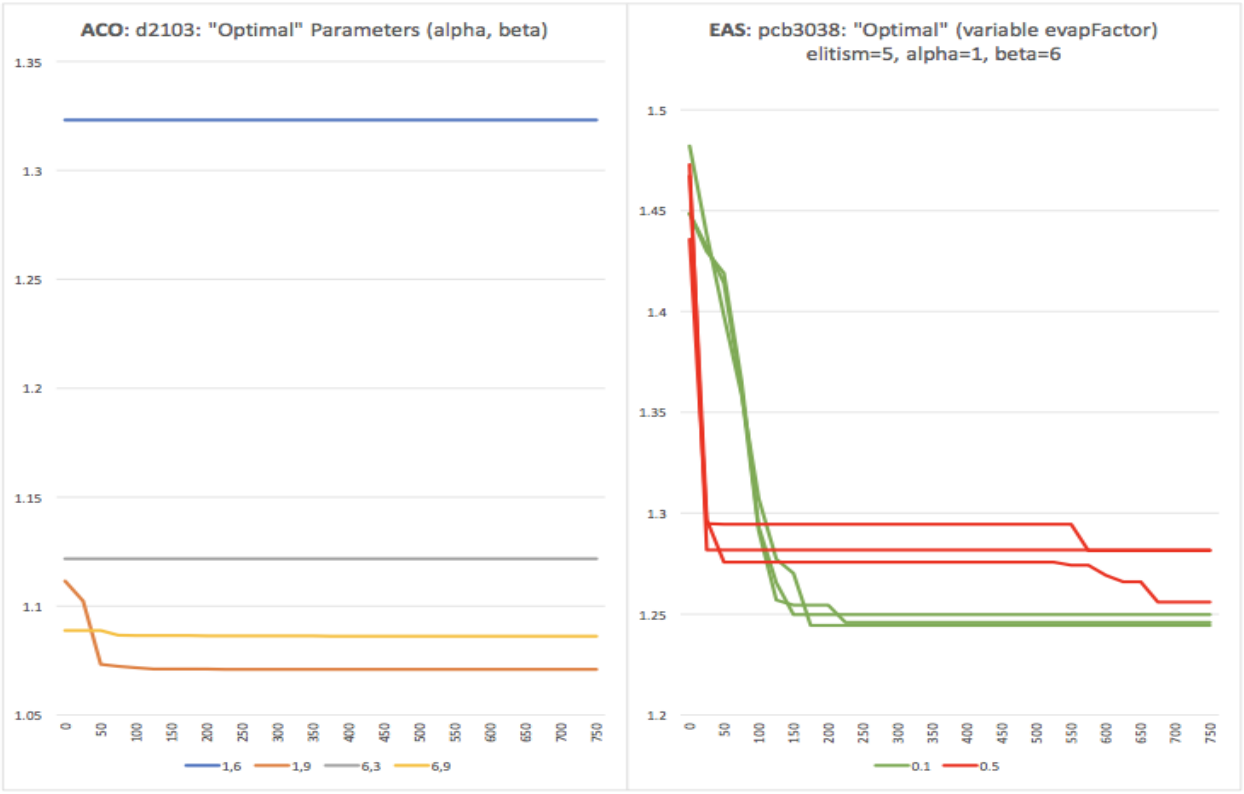
Two sets of optimal values for the parameters were discovered for ACS (as evidenced in Graphs 5, 6, 7, 8, and 9 above): one set closely adhered to the literature, while the other differed wildly. The first set that the graphs show as optimal solutions were as follows: an alpha of 1, a beta of 9, an evaporation factor of .1, a wear factor of .1, and an action selection probability of .9 (or rather, 90%). Note that the aforementioned set is essentially the set of standard, rule-of-thumb parameters, except for the beta. However, the combination of other parameters work just as well, and performs slightly faster.

The second set of optimal values that the graphs reveal were as follows: an alpha of 6, a beta of 9, an evaporation factor of .85, a wear factor of 1e-5 (0.00001), and an action selection probability of .99. In essence, the action selection probability is much higher, the wear factor is greatly reduced, and the evaporation factor is bumped up to an extreme. Effectively, the penalty for building paths is removed, ants rely much more on the best next node method for selecting their next node instead of the probabilistic method, and the best-so-far path receives even more pheromone at each iteration. This is because the best-so-far path pheromone deposit is weighted more heavily than the current pheromone on the path.

Parameter Values	Avg Soln Ratio	Running time (s)
EAS: $\alpha = 1,\ \beta = 6,\ \rho = 0.1,\ e = 5$	1.246679339	**622.9326667**
EAS: $\alpha = 1,\ \beta = 6,\ \rho = 0.5,\ e = 5$	**1.273041416**	616.6656667
ACS: $\alpha = 1,\ \beta = 9,\ \rho = 0.1,\ \varepsilon = 0.1,\ q = 0.9$	1.206734161	578.946
ACS: $\alpha = 6,\ \beta = 9,\ \rho = 0.1,\ \varepsilon = 0.1,\ q = 0.9$	**1.193620088**	565.345
ACS: $\alpha = 1,\ \beta = 9,\ \rho = 0.85,\ \varepsilon = 0.00001,\ q = 0.99$	1.199105257	566.951
ACS: $\alpha = 6,\ \beta = 9,\ \rho = 0.85,\ \varepsilon = 0.00001,\ q = 0.99$	1.215741513	**557.098**

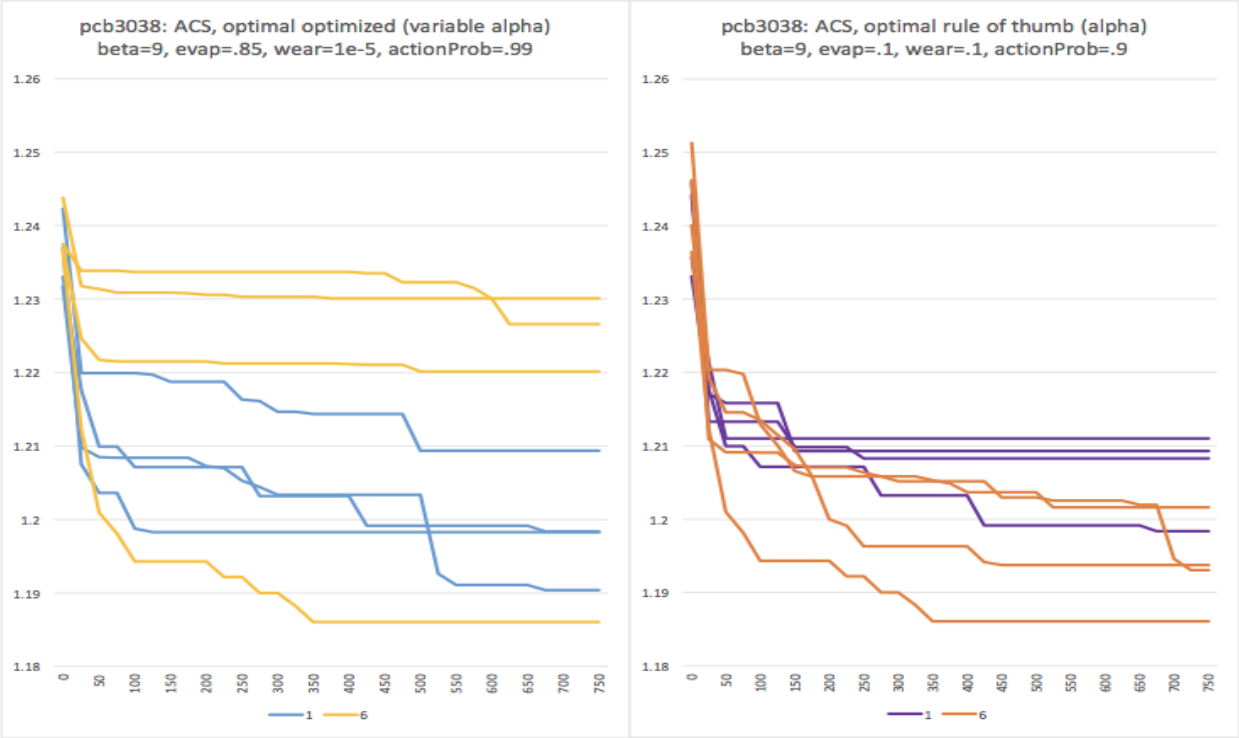
Table 1: Parameter settings with the average ratio between the found and optimal solution along with the running time after 750 iterations pcb3038.tsp

Interestingly, the bestNextNode implementation does not speed up code as fast as one might think. Computationally, consider the running time of the probabilistic method. For every ant, building a tour consists of calculating probabilities by analyzing every unvisited city--and in EAS, this is done for each progressive node. However, even with a 90% probability of disregarding this probabilistic method, and simply choosing the best city based on weighted pheromones, the algorithm on average performed only 8.329% faster. A 99% probability did not do much better: only 10.280% faster than the EAS with optimal settings. While such an increase is certainly nontrivial, it's not as fast as one might initially predict.

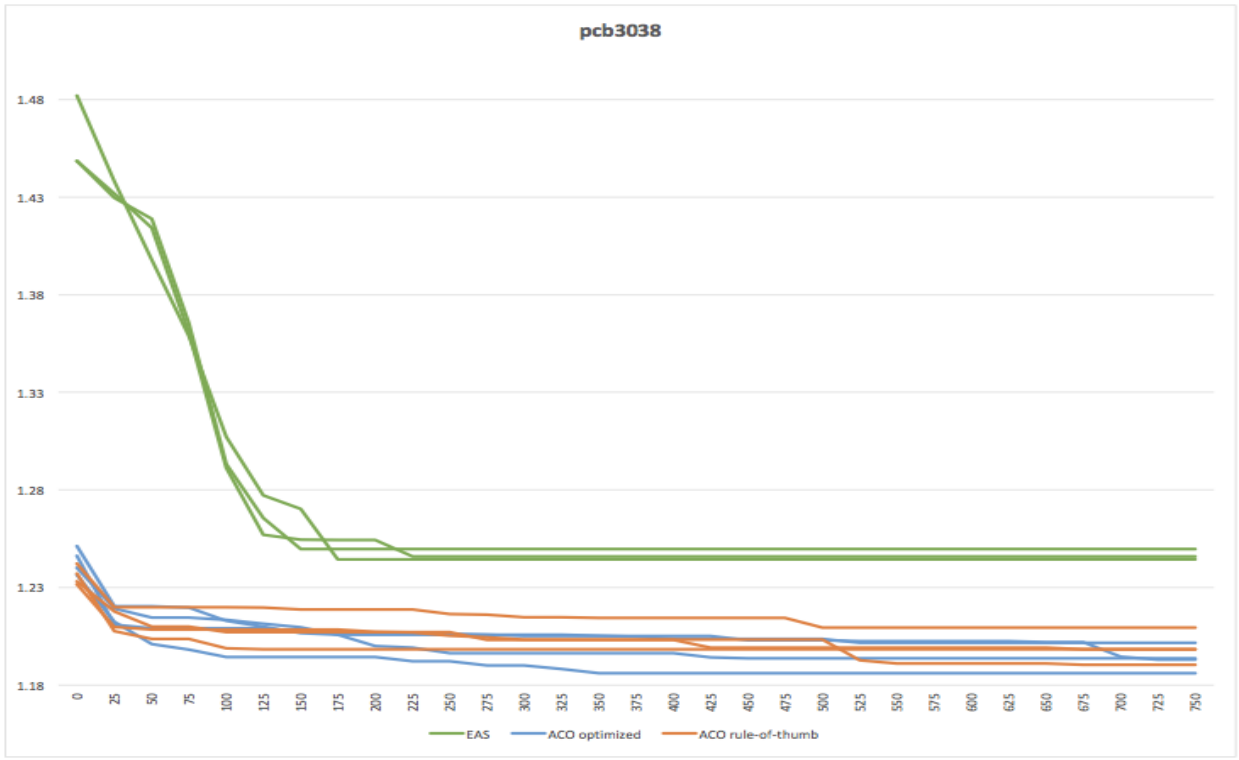


Graph 10

Left: ACS: Comparison of combinations of optimal α, β. *Other params fixed to rule of thumb. Right: EAS: Comp. of comb. of opt. evap factor. Other param. fixed to discovered optimal values.*



Graph 11: ACS, comparison of various sets of optimal parameter values (pcb3038.tsp)



Graph 12: Comparison between optimal parameter value sets EAS, ACS optimized and ACS rule-of-thumb

As evidenced by Graph 12, with the optimal settings, the first best-so-found path for ACS would consistently be better than that found by the EAS; this is certainly due to the best next node rule, and furthermore shows that this rule in practice is not that far off from solutions generated by the algorithm.

Referencing Graph 12, in pcb3038, the final solutions returned by EAS were often worse than the first best-so-far solutions discovered by ACS. Within ACS, in early iterations, the BSF paths returned by the rule-of-thumb values are better than those returned by the extreme values. Theoretically, it seems like the opposite should be true: the extreme values offer a 99% probability of using the best next node method to select the next node when building tours, versus a 90% probability for the rule-of-thumb values.

Generally, as predicted, EAS converges much quicker. The right side of Graph 10 shows that for pcb3038, the algorithm would often converge within 100-200 iterations; however, the evaporation factor of 0.5 as opposed to 0.1 resulted in a slight improvement in late iterations around the 550 mark. As evidenced by Graph 11, ACS with extreme parameter values had nearly identical behavior, which makes sense considering that the values used reduce exploration--an increase in probability of simply taking the best next node, and further increasing pheromone on the bsf tour. Graph 11 also shows that ACS with rule-of-thumb parameter values takes longer to converge and stagnate, usually after 400 iterations.

ACS variants return bsf solutions very close to their final solutions earlier than EAS does; however, the difference is that later iterations of ACS generally offer continuous improvement in the bsf solutions.

As to why higher beta values seemed to produce better solutions; a higher beta value enhances the importance of the distance of an arc. This makes sense because the distance to the next node should be a primary factor in selecting the next node.

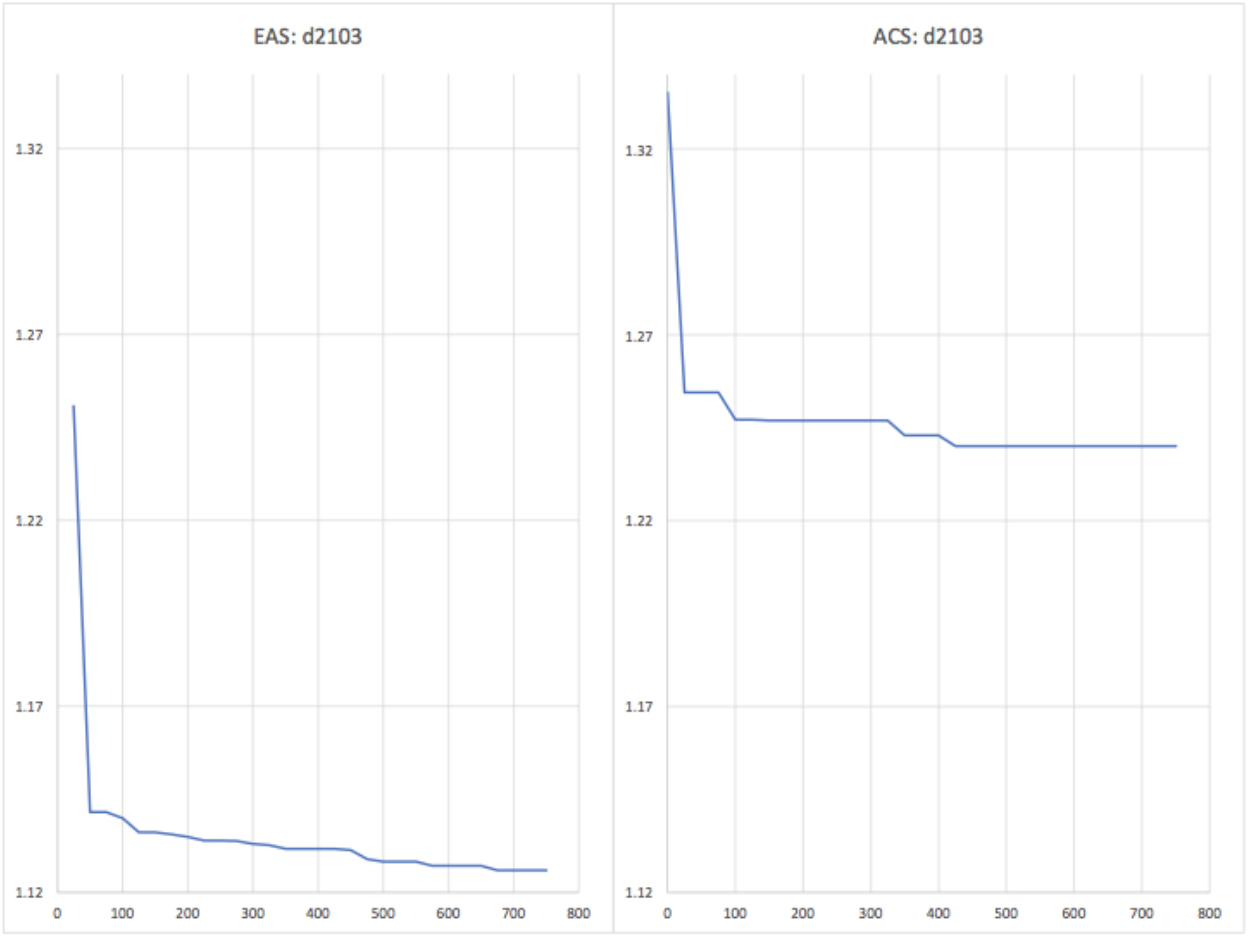
An interesting finding occurred when our algorithm was “incorrect.” Until we ran our actual experiments, for EAS, initial pheromone values that involved powers of the nearest neighbor tour length (and not the inverse of it) provided better solutions, at least for the d2103 problem under rule-of-thumb parameter settings. In addition, pheromone update rules that ensured symmetric pheromone values between cities (i.e. the amount of pheromone on the arc from i to j is the same as that on the arc from j to i) provided better solutions as well.

We did not record actual numbers, since this such tests would be based on an erroneous algorithm. However, given an optimal solution score of 80450, whereas we had to manually experiment with parameter values to return our optimal solutions scores of around 86000, our

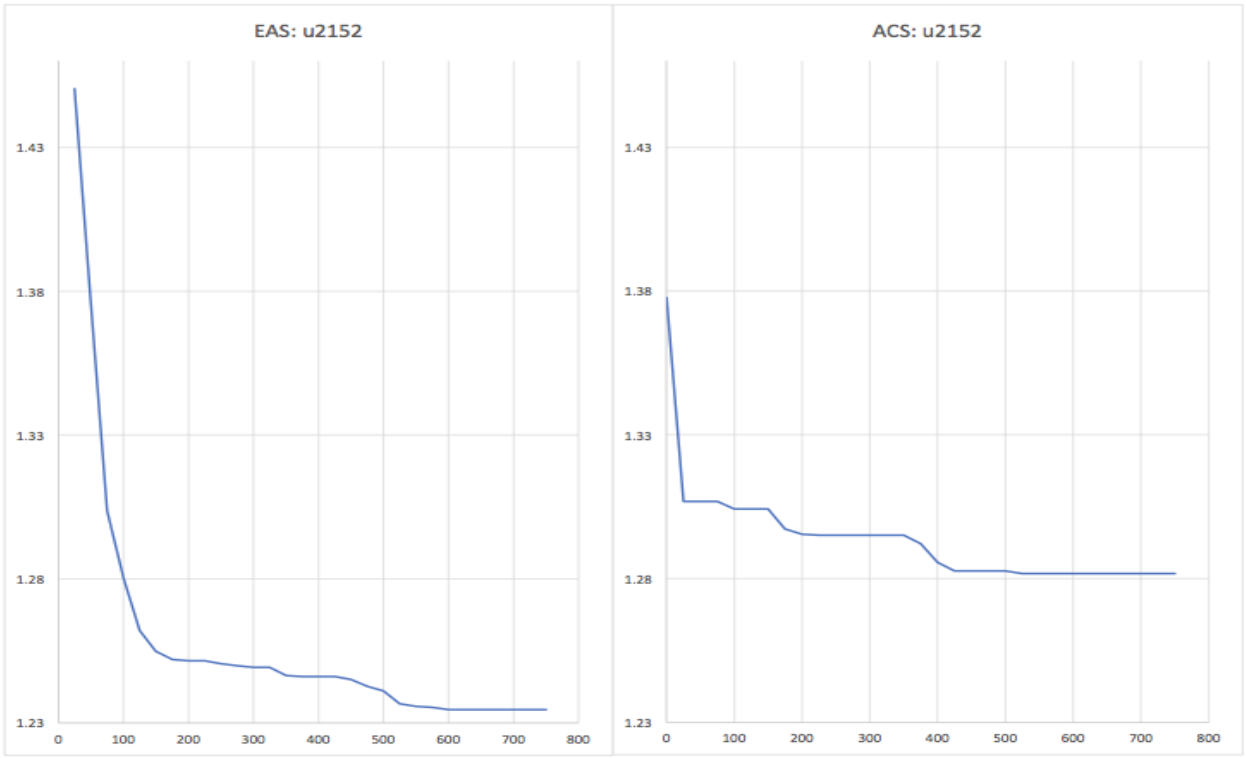
incorrect variant of EAS consistently returned values of 82000-84000. In practice, this is a significant difference: as an algorithm approaches the optimal solution, it is clearly more difficult to find better solutions.

Graphs 13, 14, 15, 16, and 17 below show the relative performance of EAS and ACS on five different TSP problems, solely using rule-of-thumb parameters. Namely, for EAS the following values were used: alpha of 1, beta of 3, evap of 0.5, and elitism equal to the number of ants (20). For ACS: alpha of 1, beta of 3, evap of 0.1, wear of 0.1, and action selection probability of 0.9.

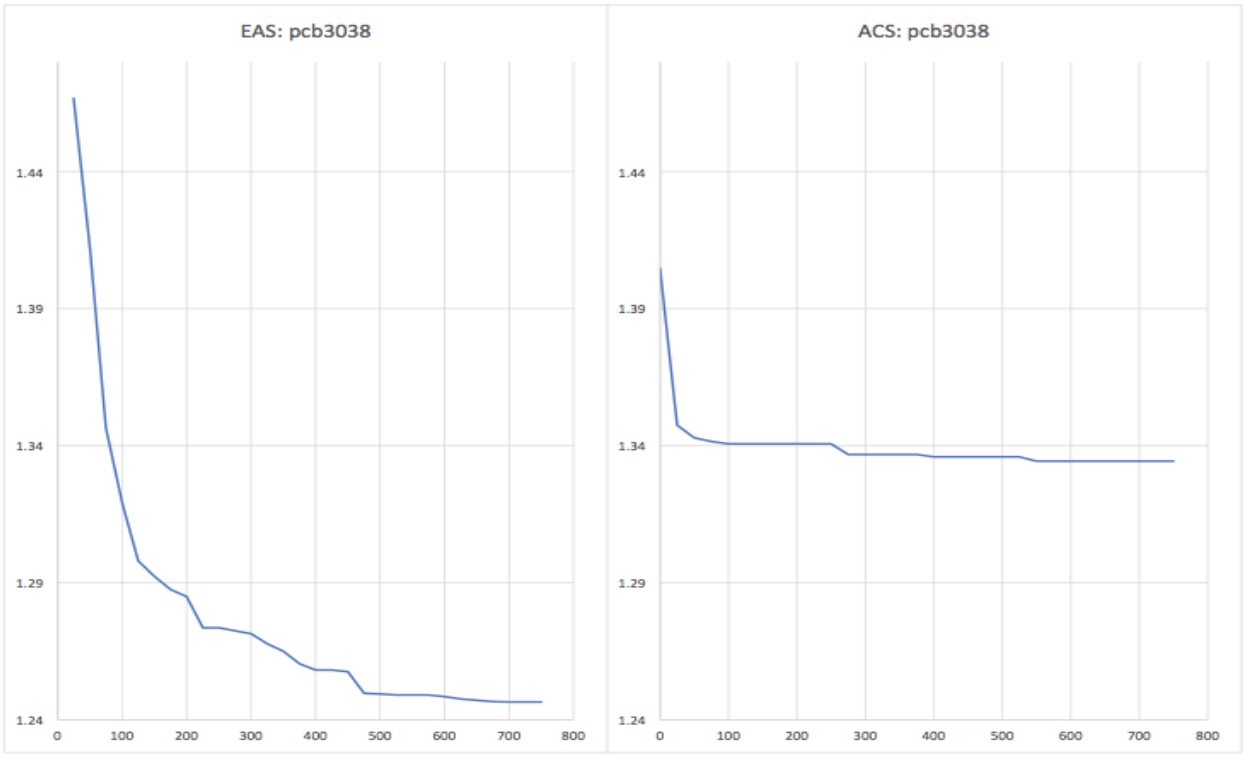
As evidenced by the graphs, EAS was always better, which is the opposite of the other results. Both algorithm variants converged nearly instantaneously (around 100 iterations) for problems with around 2000 cities, as shown by Graphs 13 and 14. For larger problems, the algorithms would take longer to converge, around 300 iterations, as evidenced by Graphs 15-17.



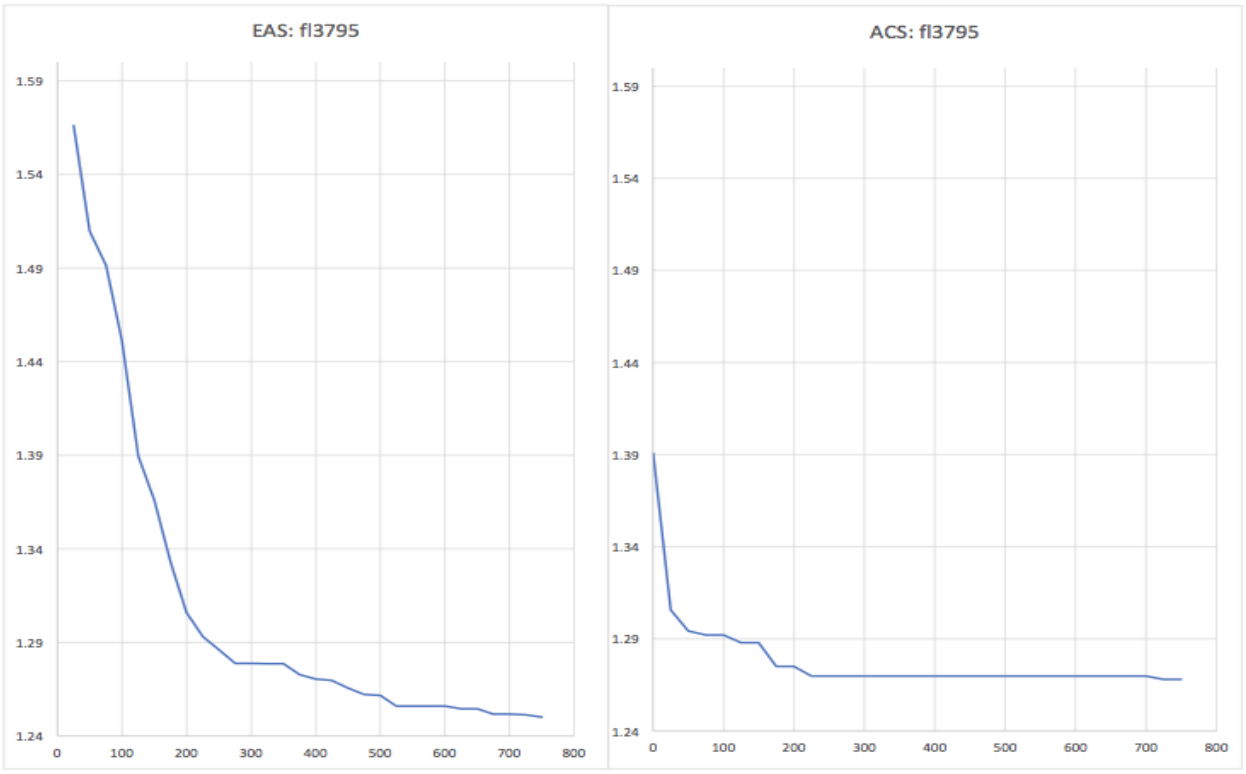
Graph 13: Initial Comparison between EAS and ACS using rule-of-thumb parameters, ratio of best solution to optimal vs. number of iterations (d2103)



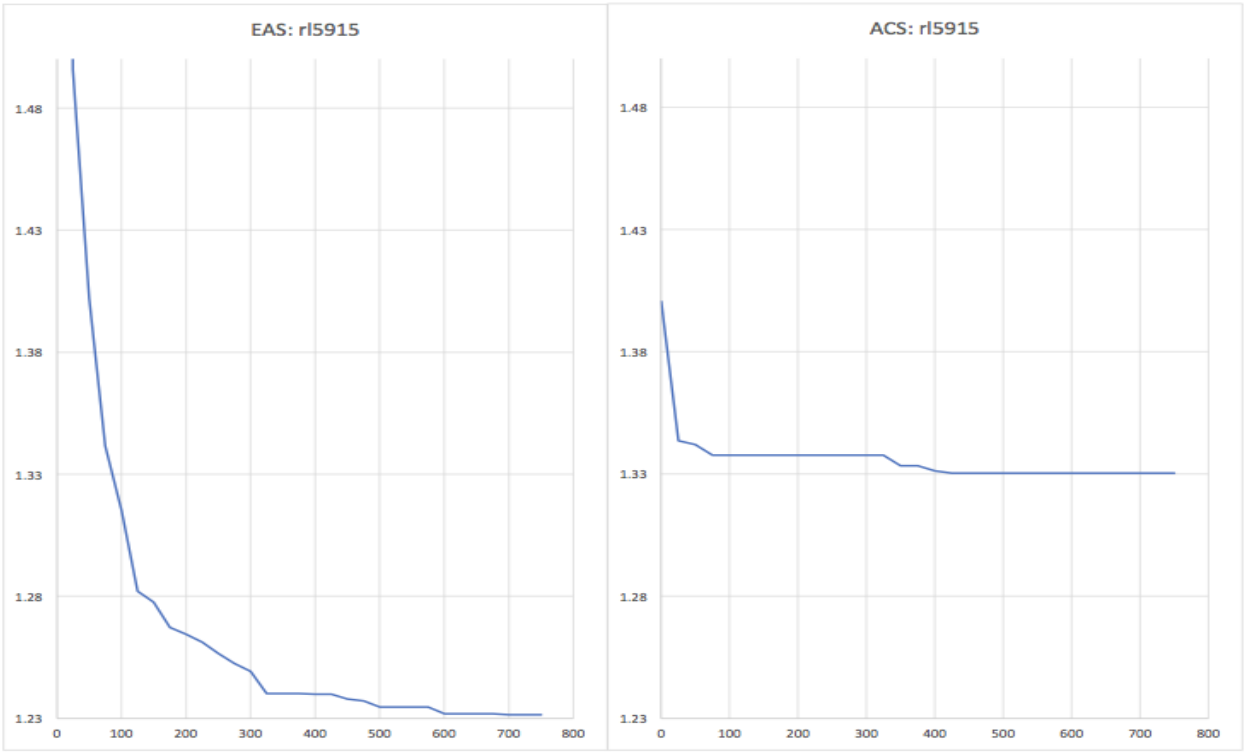
Graph 14: Initial Comparison between EAS and ACS using rule-of-thumb parameters, ratio of best solution to optimal vs. number of iterations (u2152)



Graph 15: Initial Comparison between EAS and ACS using rule-of-thumb parameters, ratio of best solution to optimal vs. number of iterations (pcb3038)



Graph 16: Initial Comparison between EAS and ACS using rule-of-thumb parameters, ratio of best solution to optimal vs. number of iterations (fl3795)



Graph 17: Initial Comparison between EAS and ACS using rule-of-thumb parameters (rl5915)

9 Further Work

The effect of adding the min-max bounds to the pheromone matrix would be an interesting problem; literature states that this variant of ACO performs much better, and the theoretical reasoning is sound: unlike EAS and ACS, it prevents premature convergence to the best paths found early in the algorithm.

In future tests, we would examine the effects of implementing a min-max bound on the pheromone matrix. Our data converged very quickly; often after 50-75 iterations we were essentially at the final solution. We hypothesize that we could be converging too quickly on a very good path, but because this very good path is found early, the pheromone levels grow so high that it may throw off the balance between exploitation and exploration. We could implement this by having bounds on the pheromone values. While updating the pheromone, if a value exceeded or fell below our upper or lower bounds, we would just set it to that bound. This would hopefully allow the ants to eventually find paths that do not include the segments with pheromone values that grew very quickly. The minimum bound would also prevent some paths from being practically eliminated. Where to put this bound would require significant testing.

Additionally, we could adjust the number of iterations vs. number of ants. While still preserving the total number of tours created, we could double the number of ants and halve the number of iterations. This would allow for more exploration early on before pheromone levels adjusted, but then a more dramatic narrowing as large amounts of pheromone are put down quickly. The opposite is also possible: we could halve the number of ants and double the number of iterations.

We also could test how the initial pheromone levels affect the results. We used 1 over the nearest neighbor heuristic, but it is possible a different value would be more successful.

10 Conclusion

ACS performs faster and more accurately than EAS when the correct parameters are used. We found parameter manipulation did significantly affect the results of the algorithms and the choice of parameters. Optimal parameters for both generally follow the rule-of-thumb values, except for the beta factor. ACS also includes a set of parameter values that are very extreme, but essentially allow it to mirror behavior of EAS (converging to a bsf solution quickly) while retaining exploration aspects of ACS (a low wear factor allows ants to keep exploring other paths).

Ultimately, we recommend the ACS variant with the following parameters:
$\alpha = 6,\ \beta = 9,\ \rho = 0.1,\ \varepsilon = 0.1,\ q = 0.9.$ It offers better solutions than ACS with extreme parameters,

with nearly identical speed. Keep in mind though, that the conclusion may be the result of the specific problems the algorithms were tested on and further testing is required to confirm that these results generalize to TSP problems of different sizes and city arrangements. The timing may also have been computer dependent due to the time restraints placed by the long run time of the problems.