# Ant Colony Optimization Sensitivity And The Traveling Salesman Problem

Ian Barczewski

## Executive Summary

In this set of experiments, we are determining exactly how sensitive the Ant Colony Optimization algorithm is when we tweak the amount of ants and cities that are used, and how this will affect such factors like computational time and the amount of cycles needed to execute the algorithm successfully. Ant Colony Optimization is a probabilistic technique that “allows the highly coordinated behavior of real ants to be exploited to coordinate populations of artificial ants that collaborate to solve computational problems. (Dorigo, 2)” By having “ants” traverse a map of “cities” like the traveling salesman problem, we can create an optimal path from the given solution space.

I made predictions that when the amount of ants in the algorithm were increased, the computational time would be increased more so than if we had increased just the cities. I also hypothesized that we would need less cycles to figure out the problem if we increase the amount of ants. Lastly, I predicted that, if I increased the amount of cities, the amount of cycles needed to solve the problem would be increased with each incrementing of cities. These three hypotheses became the back bone of the experiments, and our data collected would end up either proving or disproving these.

To be able to gather the data that I wanted to, I found and compiled some source code that coupled the ant colony optimization algorithm together with the traveling salesman problem. The program contained in the source code was able to let you change the number of cities and ants, while also tracking how long each run took, both in time (measured by milliseconds) and in cycles. I modified the settings to run for 20 ants and 20 cities, and then incremented the ants until I got to 100. I then repeated the process for 40, 60, 80, and 100 cities. Each run was executed and recorded five times, and then averaged out onto a separate table. From here, I produced four graphs, one representing the growth in time for increasing ants when cities were constant, one also representing the time growth, but switching ants with cities, and then the growth in cycles for increasing ants when cities were constant and vice versa.

My hypotheses were not entirely correctly. While initially, the computation time increased more than when cities were increased, eventually, cities came to take up more computational time, as the growth in time for increasing cities was exponential rather than linear like increasing the amount of ants. Also, when ants were increased, they showed some display of bringing the cycle count down on high amounts of cities, although it eventually flattened out to a pretty even rate. I was, however, correct that increasing the amount of cities did in fact increase the amount of cycles. These results

are relevant to the process of determining the amount of computational resources needed, the scope of the problem, and how well you need the algorithm to perform in terms of cycles, as if you set the factors too low, you will end up hitting an infinite number of cycles, thus never reaching an answer, or you will spend too many resources, thus also never reaching an answer, as the computer will simply not be able to compute the entire algorithm.

**Problem Description**

The Ant Colony Optimization algorithm technique uses many variables to determine optimal paths, but the variables that are most applicable to real life problems are ants and cities. Ants traverse the map, going to cities it has not yet visited, while cities act as stopping points on the map, with distances in between each to help determine the most efficient step in the path that the ant traces. Ants and cities can represent many things besides their namesakes, such as trucks, routers, packets, etc.

The problem here is figuring out if the algorithm is sensitive to changing values of cities and ants, and if so, how sensitive the algorithm is while using the traveling salesman problem. With more ants come more possible solutions, yet the algorithm must be ran for each ant, so computational time could be a substantial factor when dealing with large problems. Cities represent stopping points that the ants must visit, so this can also lead to large computational times as the paths become more complex with varying distances.

The relevancy of this comes when you start having to make decisions on which is the most efficient way to solve the problem at hand. When you start having a large amount of hypothetical cities, or you know that you'll be trying to cut costs computationally, the amount of ants and cities can start to become a big factor. These experiments are to figure out not only how much a change in ants and cities can cause in the computational time and cycle count of the algorithm, but to figure out when to apply more ants or cities to maximize our gains in a situation, such as cutting down computation time.

**Analysis Technique**

The Ant Colony Optimization algorithm is a technique used to finding the most optimal paths inside of a graph. The algorithm comes from the method ants use to efficiently find the fastest paths to food; as they lack the same perception of direction as animals such as humans do, they use other means to retrace their steps back. An ant will find the food source using any random path, and as it returns, it leaves a pheromone trail that attracts ants to that path. As more ants are attracted to that path, the runway will be strengthened as they leave pheromones on it. The other trails will eventually evaporate as they are traveled on less and accrue less intense amounts of

pheromone. If there are many paths, more ants will travel the shorter route, which will increase the attractive nature of that route (Back). Keeping this in mind, we will be applying this to the traveling salesman problem. Since the traveling salesman problem has a lot of real world variants where the shortest path must be found (routing trucks, manufacturing parts, networking, etc.) (Dorigo, 39-40), the application of the Ant Colony Optimization technique becomes an optimal choice for the solution to the problem.

With the traveling salesman problem, we can apply the ant colony optimization to derive a solution. As a salesman must go to every city once in the shortest distance possible, we can see how the optimization technique can apply here – this is essentially the exact same thing ants do when visiting a food source. While we are going to use a computer program to solve the problem by speeding up the solving process, we can dive deeper into the technique to understand how it gets there. Each ant is at a city and keeps a list of cities that it has already visited, known as a "tabu list." The ant will not come back to any of these previous cities on the list. The next city that the ant visits is decided by probability; to go from city *i* to city *j*:

$$p_{i,j}^{k}(t) = \frac{\left[\tau_{i,j}(t)\right]^{\alpha} \cdot \left[\eta_{i,j}\right]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{i,l}(t)\right]^{\alpha} \cdot \left[\eta_{i,j}\right]^{\beta}}$$

Where J(i,k) is the set of cities that the ant (labeled as k) still has to visit from city *i, n* is 1/d(i,j), which is the visibility (inverse distance) between the cities *i* and *j*, and T(i,j)(t) is the pheromone amount between cities *i* and *j* at time *t*. Distant cities will have less chance of being chosen, while the amount of pheromone intensity will increase the odds that the respective city is chosen. After the ant crosses each city and completes the cycle, it deposits pheromone on the path. The algorithm for this is:

$$\Delta\tau_{i,j}^{k}(t) = \begin{cases} Q / L^{k}(t) \\ 0 \end{cases}$$

The top part of the problem represents each edge *(i, j)* that the ant visited in the iteration *t,* otherwise it is zero. After this, "pheromone evaporation" is used. This is to simulate pheromone trails by ants evaporating over time. Each edge will have it applied

with a coefficient p that represents decay. The algorithm then is:

$$\tau_{i,j}(t+1) = (1-\rho)\cdot\tau_{i,j}(t) + \sum_{k=1}^{m}\left[\Delta\tau_{i,j}^{k}(t)\right]$$

In this algorithm m is the number of ants present in the current system (Meyer).

With these three algorithms at hand, we can then run the entire optimization. We decide on an amount of iterations that the optimization will run in order to end up with the shortest path. Each edge gets initialized to an extremely tiny, uniform level of pheromone. Each ant then is set onto a random city, and from there, each ant will build a tour with the probability algorithm above. After that, we check if the best tour built is better than the current solution if there is one, and if so, we update that to be the best solution. After that, we run the pheromone decay algorithm. We must remember that no pheromone will be deposited until an ant completes the cycle of cities, which will then increase the probability that the pheromone trail will be hit by another ant.

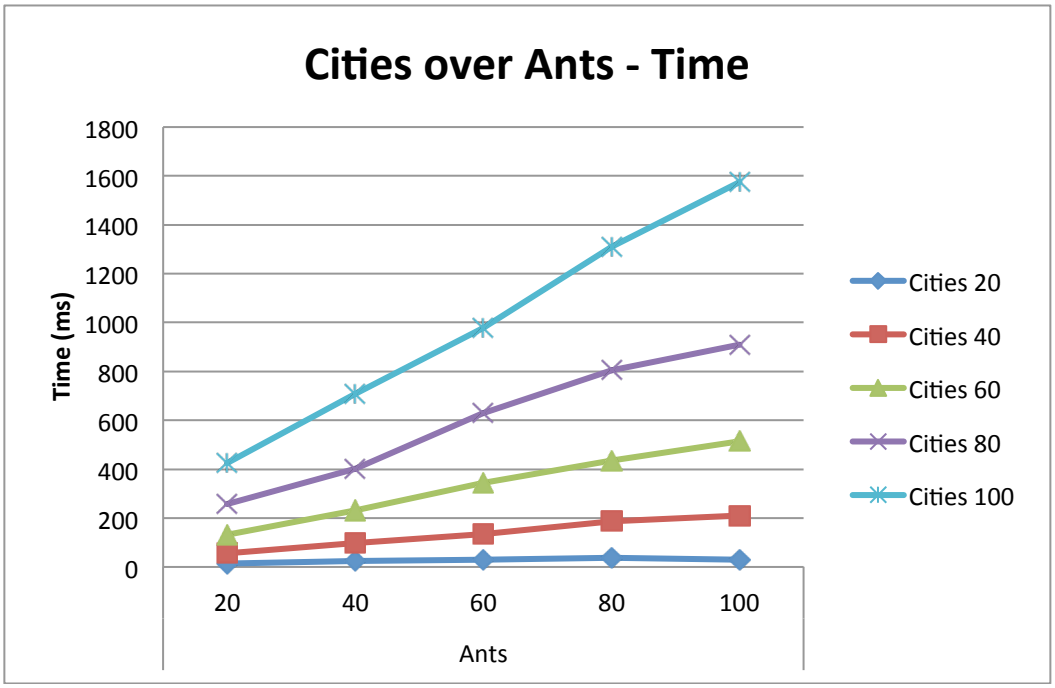
To solve the problems stated above, I used source code that I found called “AI Demo” written by Peter Kohout in C++. AI Demo allowed you to be able to modify the parameters of the program to be able to change how many cities and ants you would use for that run. The program then executes a run, showing what the distance of the best possible path is, the distance of the current path, and the actual graph, with the pheromone trails drawn in. While executing, the program outputs the current cycle – labeled as ‘epochs’ – and shows how long the algorithm has ran for.

I decided that, by using this program, the best way to track the sensitivity was to use an incremental system involving the amount of ants and cities used. I would start at 20 cities, make five runs using 20 ants, record the results each time, then increment the ants by 20 and repeat the process until I obtained results for 100 ants. After that, I would increment the cities by 20, reset the ants by 20, and restart the execution process for the ants.

I predicted that the amount of ants increasing would increase the computational time more so than the cities would. The algorithm runs for each ant in the equation, so there would have to be an increased amount of computation time for each run in the algorithm. I also predicted that, with increasing ants, would come less cycles over each run, almost to the point where very few cycles would be needed; on the inverse of that, with increasing cities, more cycles would be needed to be able to determine the optimal solution within the solution space.

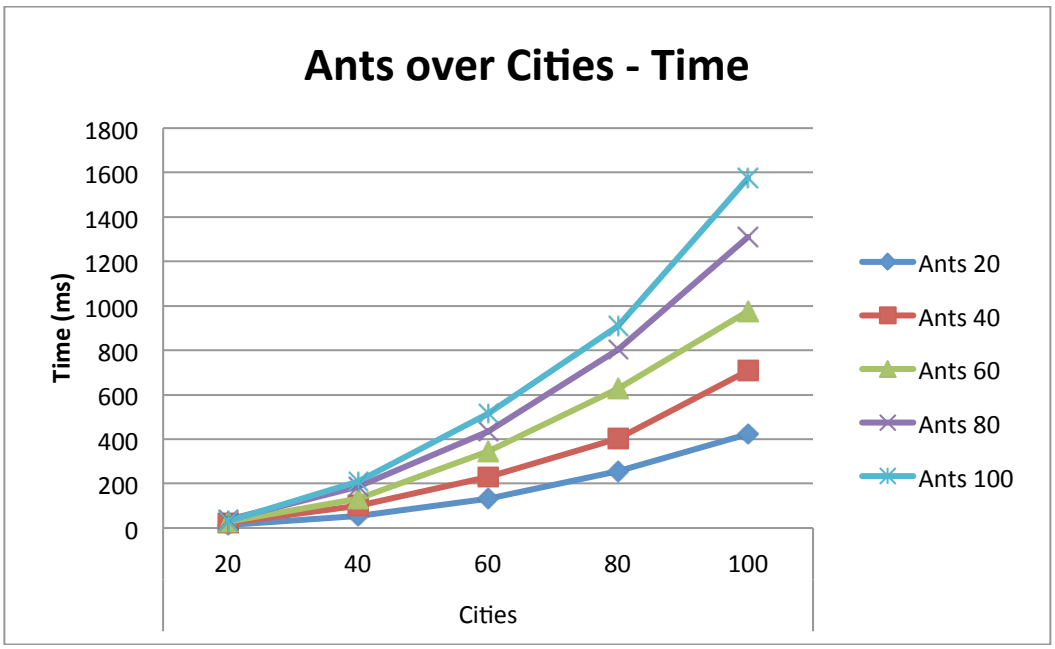
**Ants and Cities Over Time**

The graphs below shows the growth of average time spent as the cities and ants were increased.




Looking at the cities, as the amount of ants increased, the time seems to take larger jumps around the sets of 40 ants and 60 ants. However, it then seems that the increase in time starts to slowly flatten out near the end, around 80 and 100 ants. This could possibly be due to the fact that the computational time between maps with more cities becomes less as more ants tend to find a solution faster. Observing the graph, it is easy to conclude that there will be a linear increase when you increase the amount of ants in any amount of cities.

By having a linear growth over time, we can also safely predict a ballpark estimate on how much time the algorithm will take. If we can approach a problem by being able to increase the city count without increasing the ant count, we can get a good handle on the resource end of the program and be able to efficiently work with time in our favor.

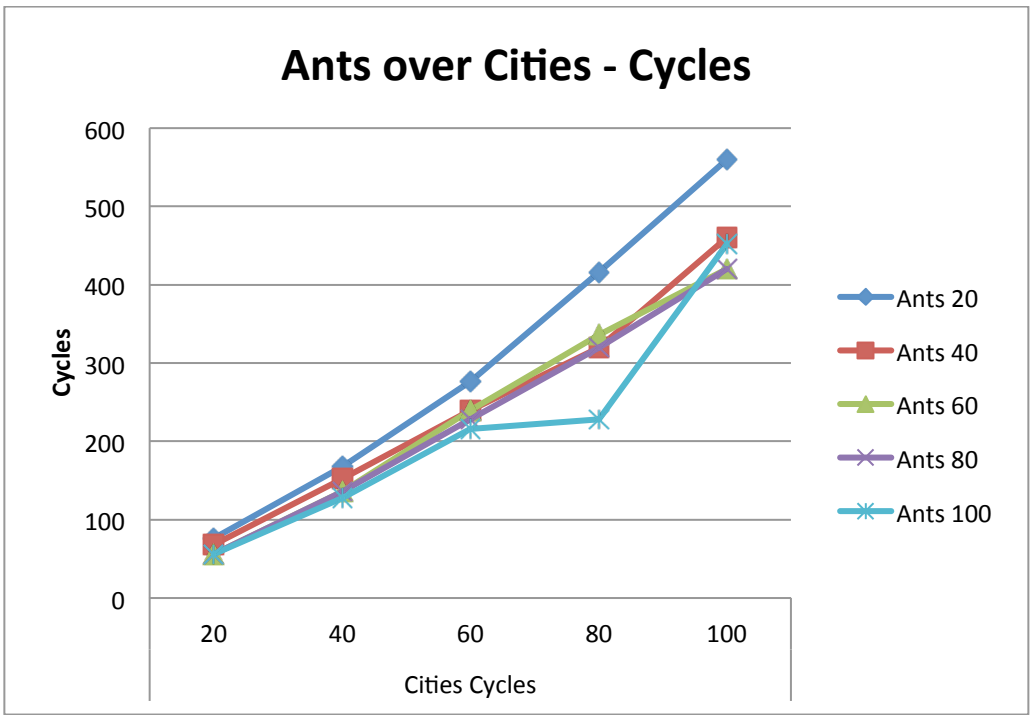



Switching to the ants graph, however, tells an interesting tale. All of the ants create gaps that become larger over time. The steps in the increase of their own time also increase. Clearly, each ant is showing behavior of exponential growth. I believe that this is due to the increase in the amount of computational time that ants will need to take to process each city before determining what the next step it needs to take is.

As seen above, by adding more ants, we set a severe spike in computational time up when we start approaching 100 cities, especially with using 100 ants. If fewer ants are used, the computational times seem to stay pretty tolerable, so if a problem could be restructured to use fewer ants, we may be able to create a pretty fast solution.

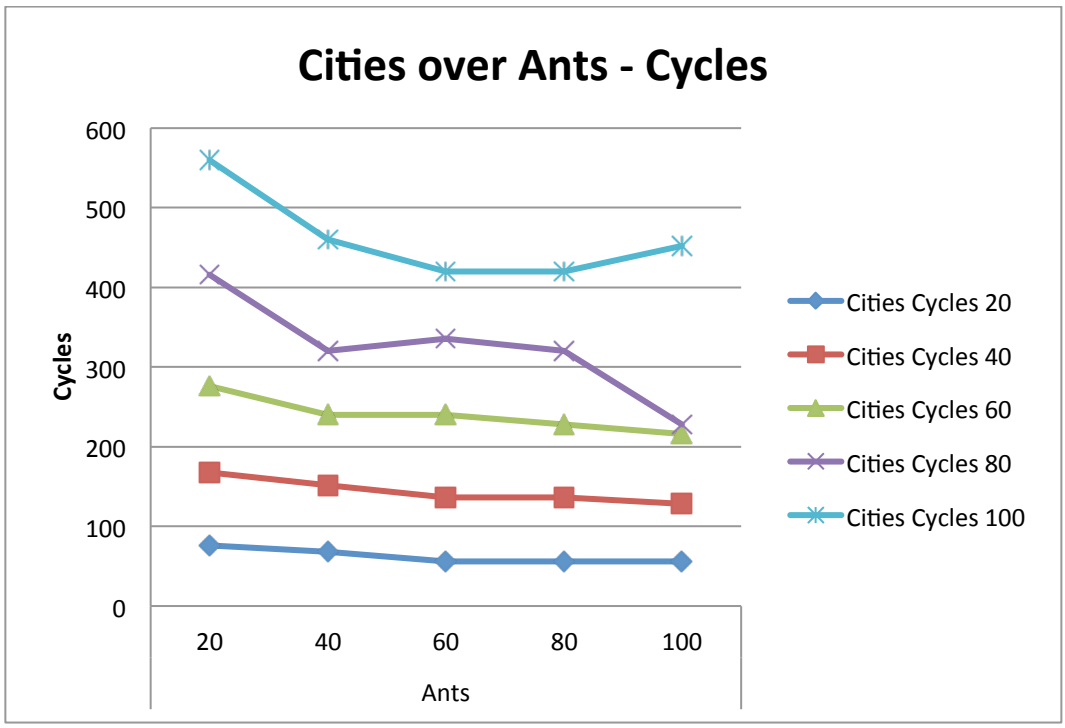
**Ants and Cities over Cycles**

The graphs below shows the growth of average time spent as the cities and ants were increased.




Observing the growth of ants over the amount of cities increased, the growth is clearly linear between all ants, although it does slightly change. What was interesting is where the line of 100 ants diverts at 80 cities. This was due to the runs at 100 ants, 80 cities reaching 240 cycles in two of the five runs that it had, which pulled the average down pretty heavily, as the average run before that was roughly around 320 cycles. This makes sense, as the runs have far more ants to solve the increase of cities, as opposed to the rest of the ants. The average for 100 ants comes back into conformity with the other ant runs at 100 cities.

Also interesting to note is the line that represents 20 ants. There is a clear disparity between the line and the rest of the graph. At a certain point, when there are too few ants to the amount of cities, the program runs forever, never reaching the optimal solution. This is due to not having enough ants to do such things like update the pheromone trails to keep them dependable. I believe that this disparity is an approaching result of just that. If we were to increase the amount of cities, yet let the amount of ants the same, I believe that each set of ants would eventually start drifting off higher and higher until the “infinity point” is reached.




This graph is actually pretty interesting, as there is not any real conforming pattern across the increase of ants. It seems that from cities 20-60, there is a pretty consistent amount of cycles, with a slight trend towards a decrease, though nothing that should be made into a major issue. When we observe lines 80 and 100, we start noticing that the consistency is abandoned for a much more sporadic result. This can almost be concluded to the fact that there are not enough ants near the beginning, yet it regains a slight sense of consistency near the end. However, the line that represents 80 cities takes a sudden dive near the end, due to the results having significantly less cycles on average. Overall, it is clear that with the increase of cities, no matter what, the cycle count will always be higher across the respective ants.

**Assumptions**

I assumed that the source code that I received was valid and working. I also assumed that all of the data that I entered was free of human error.

**Results**

Based on the results derived in the Analysis Technique section, I was able to make conclusive answers on the problems described in the Problem Description section. I was able to conclude that the Ant Colony Optimization algorithm is indeed rather sensitive to both changing values in both cities and ants, albeit in different ways. Knowing this, after having tested the sensitivity of the algorithm, measured both by time and by cycles, we can pinpoint exactly how sensitive the algorithm actually is.

When tracking the sensitivity of the algorithm across time, I noticed that, when increasing the amount of ants used, the growth of the time that it took to derive the optimal path increased rather linearly; however, increasing the cities, while keeping the amount of ants at a constant number, seemed to make the growth increase at an exponential rate, possibly reaching an infinite amount of time where the computer simply cannot make the calculations necessary. This exponential increase could be due to the fact that there are simply too many ants covering the solution space with multiple paths being recorded, while keeping the large amount of current pheromone trails "fresh."

Recalling my previous prediction, it turns out that I was only partially correct about this. While increasing the amount of cities did not get to the higher time measurements as fast as increasing the ants did, the jumps in time measurements between the increments had begun to become larger. I believe that if I would have continued this experiment across 200 cities and 200 ants, we would see the increasing cities demanding higher amounts of computational times as opposed to the increasing ants.

Knowing that increasing cities cause an exponential growth in time, as opposed to the linear growth of increasing ants, we can say that increasing cities may not be the most effective approach to a problem if we know that computational time and resources are issues that must be factored in, especially if we are going to be computing a large problem. We could be waiting for an answer that may never come, due to the sheer amount of computational time needed to calculate every cycle through – something that I myself experienced when working with large amounts of ants against large amounts of cities. If possible, it may be more efficient to figure out a way to restructure the problem into sub problems, or to try to use fewer cities.

Looking over the cycles section of the experiment, we can see that, as we increase the amount of cities while the amount of ants stays constant, the growth of cycles tends to stay pretty linear; however, the line for 20 ants is veering off into more cycles. This shows that, as the number of cities increases, we can start having problems finding the solution. Eventually, we will reach a point where we will have too few of ants to be able to reach a conclusive solution, as there are simply too many paths to keep the pheromone trails fresh as they decay.

What is interesting to see is the plot of the amount of cycles as ants are increased while cities stay constant. The amount of cycles, for the most part, stays pretty constant, even though the amount of ants increases over time. I think this shows that when you hit a set amount of ants, they are able to solve it at a certain amount of cycles every time, due to having so many ants and not enough random paths to lay a wide range of pheromone trails. Essentially, through probability, a large amount of ants

across a small amount of cities will make an extremely strong pheromone trail in their first run through; this trail will be like a magnet to the ants when choosing where to go next, as opposed to having fewer ants, since they will not lay down as much pheromone trail.

Again, it appears that increasing the amount of cities is not very efficient for the amount of cycles. If there is a possibility of cutting down the city amount by subdividing the problem, we can then get a handle on the cycles. I found that the cycles ended up being a rather telling gauge of the performance of the algorithm – if the cycle count started to climb higher than the other runs, as it did with 20 ants and100 cities, we would be getting near a point where the algorithm would not be able to find an answer. Also, when increasing the ants, they would eventually hit a point and flatten out the amount of cycles it needed. From there on, it seems that increasing ants is a waste in terms of computational time, as you will still be getting an optimal performance from the algorithm itself. This is a tradeoff that must be noticed when determining algorithm variables.

I had predicted that more ants would cause less cycles to occur when cities were set at a constant number, while more cities would cause more cycles when ants were then the constant. I was partially correct about the former; as the ants increased, a few of the plots had lowered the amount of cycles it had taken to solve the problem. However, the amount of cycles had begun to flat line and stayed close to a constant amount, as explained above. I was correct in my prediction about the latter; with each incremental increase in the amount of cities, the cycles increased at a near-linear rate. This makes sense, as there is less pheromone to be attracted to as it decays over time as they search such a large space.

From these results, we can conclude that, as stated before, that the algorithm is sensitive to both changing amounts of ants and cities when measured by time and cycles; however, both have different costs. Using the results at hand, we should be able to use the research I have gathered and be able to apply it to a real world problem by figuring out the best way to balance computational time without letting the amount of cycles get out of hand, or even worse, not derive a solution at all.

**Notes**

-Algorithm images obtained from Bernd Meyer's presentation on Ant Colony Optimization; see Works Cited

-There is only one citation for each author; however, Meyer's citation covers most of the algorithm details and Back's citation covers the majority of how ants function.