ME 397P Project Course: Studying Ant Colony Optimization and Its Signal Convergence

Khai Yi Chin

Department of Mechanical Engineering, University of Texas at Austin

1 Introduction

Swarm intelligence draws much inspiration from nature. From fish schools to bird flocks, the behaviors of each member in the swarm gather to form a coordinated action, usually toward a target that prolongs their survival. The Ant Colony Optimization (ACO) algorithm is one prominent example, that mimics the behavior of ant colonies in order to compute an optimal path among many others.

Introduced as a new approach to stochastic combinatorial optimization, the ACO algorithm is a versatile and robust population based method, utilizing agents that explore the solution topology exhaustively. These agents, called 'ants', mimic their biological counterpart by marking each attempted solution with 'pheromone trails' – a value proportional to the optimality of the solution. Solutions with higher pheromone trail levels are more attractive to subsequent ants, thus reinforcing the optimal solution in a positive feedback process. Given two paths shown in Figure 1, the shorter route has a higher frequency of traveling ants, and with the pheromone laying it gets repeatedly more attractive to future ants. In the algorithm, the ants balance a greedy heuristic[1] – used to drive the ants' movement in the early stages – and attractiveness (pheromone trail level) of a specific route to decide their movement as they search for a solution [1].

First introduced by Dorigo, Maniezzo, and Colorni in 1996, much work has revolved around exploring and expanding the ACO algorithm's capabilities in solving a wider class of numerical problems, but rarely has any of them studied the underlying principles of its effectiveness. Sensing a disconnect in this information, Ph.D. student Jack Hall from the Neuro Engineering Research and Development Lab (NERD Lab) has proposed a model, called the "flow program", in an effort to study and model the behavior in the ACO algorithm and other machine learning algorithms [3].

Central to swarm intelligence is the concept of emergence – interactions between smaller/simpler entities as part of a larger entity reveal properties observable only on the macro level. In an attempt to express and provide intuition to current emergent algorithms, Hall's flow program consists of a network of interconnected nodes, which receive, process, and transmit the signals through the network. For my work in this course, I focused on a specific form of the flow program, suited to model the ACO algorithm.

[1] A heuristic, or more properly called a heuristic technique, is simply a method that assists in solving problems [2]. In the context of the ACO algorithm, a greedy heuristic (technique) typically drive towards suboptimal solutions (local optima) in a bid to guide the search towards the optimal solution (global optimum) quicker. If however, much importance is placed on the greedy heuristic, the ACO algorithm might never come across the global optimum, hence the term 'greedy'. As we will see in the following section, the greedy heuristic for the TSP problem solution is the closeness between the current city (the ant is at) and the possible destination cities.

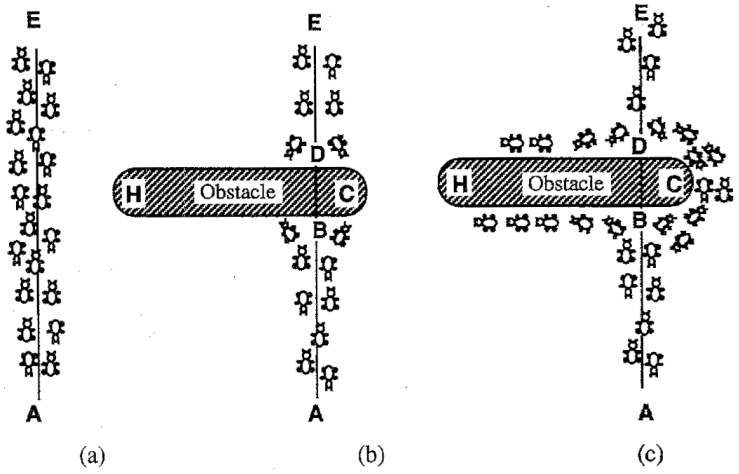


Figure 1: In (a), the ants travel on a single path from point A to point E. In (b), an obstacle is added, resulting in two paths of different lengths. As the path ABCDE is shorter than the path ABHDE, more ants would be able to pass through the former (in the same amount of time), laying more pheromones on said path in (c). Consequently, the frequency of ants on the shorter path means evaporation of pheromone trails occurs on a lower rate than the longer path. Figure obtained from [1].

To realize the objective of understanding the underlying dynamics of the ACO algorithm, my work revolved around examining the signal behaviors between the two methods of execution: the signal-based (ACO algorithm) execution method versus the node-based (flow program) execution method. The signal-based method places much emphasis on the abilities of the signals (ants) to search for a solution – through the selection of the most attractive route – while the node-based method focuses its functions on the nodes, where they modify signal properties as they move the signals about. The main difference between these methods is their approach to navigate through the search space; the signal-based method does so randomly relying on probabilities, whereas the node-based method does so with a deterministic model. Mathematically, the two methods are equivalent, in that the ant mass distribution across a graph – a network of nodes and connecting links – converge to the same equilibria [3]. My role in this study is to provide a comparison of the convergence rates between the two methods of execution, and show that the ant mass values match after convergence occurs.

In the following parts of the paper, I present my replication of the traveling salesman problem solution as demonstrated by Dorigo, Maniezzo, and Colorni (1996) using the ACO algorithm on the Oliver30 problem, as an example to show the working principles of the algorithm. Next, I provide the definition of the path-planning problem, adopted to test the algorithms. Then, I describe the structure of the signal-based algorithm and node-based algorithm respectively, both built and designed in Python 3.6, and used to run tests on different graph types. Finally, I discuss results and comparisons of tests ran on both types of execution methods.

2 Replication of the Solution to the Oliver30 Traveling Salesman Problem

The traveling salesman problem poses the challenge of constructing the shortest tour that visits all the cities only once in a given map. It is a NP-hard problem in combinatorial optimization [4]. Based on the paper by Dorigo et al. [1], I built the ACO algorithm in Python 3.6 to solve the Oliver30

traveling salesman problem, with the city coordinates obtained from Oliver, Smith, and Holland [5].

In the algorithm, each ant visits all the cities, completes the tour, and then 'lays' pheromone trails divided by the length of the entire tour. The pheromone trails on each tour/solution are then inversely proportional to the distance of the tour distance of each ant. For every cycle, the ants are sent to explore the graph one at a time until the last one sent completes its tour. Then, one by one the ants track back the route they traveled, laying pheromone trails as they move back to their starting point. The cycles are continued until an arbitrary threshold is reached; the shortest tour in each cycle is collected and replaced with the next shortest one, storing only the shortest of all cycles when the algorithm reaches the cycle threshold.

Given the set of n cities N, the set of m traveled cities M, and the set of allowed neighboring l cities L, the decision making mechanism to move from city i to city j of the k^{th} ant is governed by the transition probability:

$$p_{ij}^k = \frac{{\tau_{ij}}^{\alpha}{\eta_{ij}}^{\beta}}{\sum_l {\tau_{il}}^{\alpha}{\eta_{il}}^{\beta}};\ i \in M,\ j \in L \text{ and } L \subset (N \setminus M) \tag{1}$$

where τ_{ij} is the pheromone trail level on the link connecting city i to city j, η_{ij} is the distance between city i and city j, and α, β are parameters that control the relative importance between pheromone trail level and closeness of cities. η is provided by the graph, while τ is updated after the t^{th} cycle with the following equation:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_k \frac{Q}{L_k};\ 0 < \rho \leq 1 \tag{2}$$

where ρ is the trail evaporation parameter, Q is the pheromone constant for the ants, and L_k is the total distance traveled by the k^{th} ant in cycle t. In this example, all links in the graph are initialized with a uniform pheromone trail level q such that $q \ll \eta, Q$. This is so that the transition probability does not have a zero as its denominator, and with such a small q the transition probability is not biased by the trail level variable.

For the Oliver30 TSP solution replication program (as well as programs in the subsequent sections), I utilized a package called `NetworkX` to aid in the creation of the graph structure. The number of ants used in this solution replication example is the same as the number of cities (i.e. 30), with each ant having a unique starting location. The pseudocode for the program is shown in Algorithm 1.

The program was executed for the different number of cycles listed in Table 1, and the shortest tour distance (for each number of cycles) was averaged over three trials. With $\alpha = 1$, $\beta = 5$, $\rho = 0.5$, and $Q = 100$, the program was able to generate to within 0.04% of the solution in [1] in the same amount of cycles of 5000. Since the goal of this solution replication exercise was to reinforce my understanding of the ACO algorithm, I was not too concerned with the tiny margin of difference between my and the compared solution.

Table 1: The shortest tour distance for different cycle threshold values.

Number of Cycles Ran	100	250	500	1000	2500	5000
Shortest Tour Distance	429.903	428.194	425.685	424.153	424.789	423.912

Algorithm 1 Traveling salesman problem solution using the ACO algorithm

```
1: procedure ACO_METAHEURISTIC()
2:     Set number_of_cycles := k        ▷ k is the arbitrary parameter of maximum cycles intended
3:     Set cycle_counter := 0
4:     Create Oliver30 graph                          ▷ Graph created with the NetworkX package
5:     while cycle_counter ≠ number_of_cycles do                          ▷ Run the cycles k times
6:         Run the ants_generation_and_activity_cycle function from line 13
7:         Store the shortest solution
8:         Make graph evaporate pheromones          ▷ Evaporation using self defined object method
9:     end while
10:    Organize and display the shortest tour across k cycles
11: end procedure
12:
13: function ANTS_GENERATION_AND_ACTIVITY_CYCLE(graph)
14:    Create an array of ant objects ant_array
15:    for all the cities in graph do
16:        Assign each ant a starting point in the graph
17:    end for
18:    for each ant in ant_array do
19:        Move ants through graph                          ▷ Ants move via object methods
20:    end for
21:    Set shortest := l                    ▷ l is arbitrary s.t. l ≫ longest tour distance
22:    for each ant in ant_array do
23:        if length of current ant tour < shortest then
24:            result := array of toured cities in current ant tour
25:            shortest := length of current ant tour
26:        end if
27:        Send ants back to lay pheromones based on the last term of Equation 2
28:    end for
29:    return result
30: end function
```

3 Definition of the Path-Planning Problem

In the current and following sections, I address 'cities' as mentioned in Section 2 as 'nodes'.

In the interest of comparing the signal-based algorithm to the node-based algorithm, I looked at a different problem without the requirement of ants having expansive memory[2]. This is because the signals in the node-based algorithm have no such capabilities, given that the goal for the flow program is to model natural ant colonies more closely than current swarm intelligence algorithms do. This led to the investigation of a common problem present in swarm robotics: the path-planning problem.

The problem is defined as follows: given a starting point in a graph of nodes, find a combination of nodes that make up the optimal path to the end node. Typically, the optimal solution is defined to be the shortest path from the start to end, although there could be constraints such that the shortest path does not necessarily reflect the best solution.

[2]In the ACO algorithm for the TSP, the ants have memories proportional to the size of the graph – specifically, number of nodes. This is an artificial trait that was added to enhance the algorithm's problem solving capabilities, which is absent in biological ants.

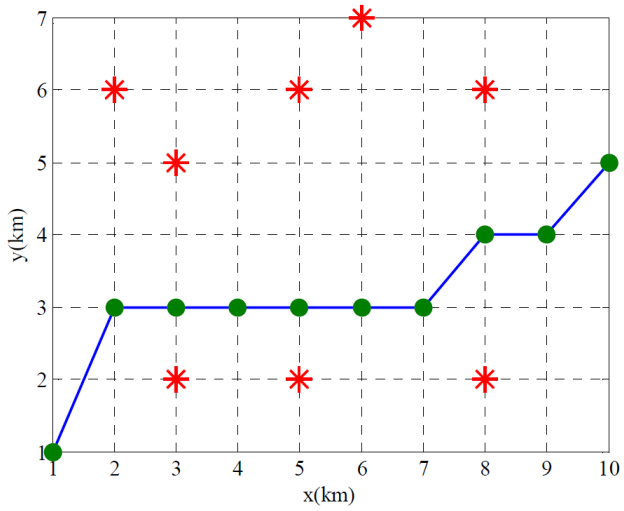


Figure 2: An example of a path-planning problem studied by Zhang, Zhen, Wang, and Li, where they used the ACO algorithm to perform assignment planning for UAVs flying across a region [6]. The red asterisks represent threats which decreases the optimality of a solution (in blue), should the path (from the start node at (1,1) to the end node at (10,5)) come close to it. Figure obtained from [6].

In this study however, I am only interested in the behavior of the signals as they reach equilibria, thus no optimal solution was sought. Moreover, the ants only move from the start to the end node for one cycle (signal-based) or in one direction (node-based) – without them updating the pheromone trail level on the links – because I am only concerned about the signal dynamics (ant behavior) responding to constant – or more appropriately, quasi-static – state dynamics (pheromone trail levels). The variables of interest are then:

1. the number of ants (signal-based) or the number of node executions (node-based) for convergence to occur, and
2. the properties of graph used in terms of the number of nodes and the number of connected links.

Therefore, as I demonstrate in the upcoming sections, the algorithms move the signals (ants) in graphs that follow the conditions of a path-planning graph with respect to the variables, until the signals display steady-state behavior in its ant mass distribution.

Furthermore, the graphs are given additional certain constraints/conditions.

1. The graph is randomly generated based on a given number of nodes and links.
2. The graph can be visualized as a planar map with nodes situated across this map, connected by links.
3. In regard of future works on this study, the direct connection between the start node and the end node is removed[3]. As such the maximum number of links for a graph of n nodes is determined by:

$$\kappa_n = (\sum_{k}^{n-1} k) - 1 \tag{3}$$

[3]In this paper the signal-based and node-based algorithm only runs for one cycle (signal-based) or one direction

4 The Signal-Based Algorithm

As mentioned in Section 1, the signal-based execution method gives agency to the ants as signals. It depends on the probabilistic behavior of ant signals choosing paths to move to, which is otherwise quite random at first. However, the seemingly random behavior slowly converges to a solution after a while, which I test and show the results in Section 6.

This algorithm is modeled closely to the ACO algorithm, but they are not the same. One major difference is that the ACO algorithm relies heavily on the concept that the ants have memory, used to prevent circling or repeating of nodes, whereas the signal-based algorithm has no such restrictions, which manifests in its transition probability (of the k^{th} ant moving from node i to node j):

$$p_{ij}^{k} = \frac{{\tau_{ij}}^{\alpha}{\eta_{ij}}^{\beta}}{\sum_{l}{\tau_{il}}^{\alpha}{\eta_{il}}^{\beta}};\ i \in M,\ j \in L \text{ and } L \subset N. \tag{4}$$

where N is the set of nodes in the graph, M is the set of traveled nodes, L is the set of neighboring nodes to node i, τ_{ij} is the pheromone trail level of link (i, j), η_{ij} is the distance of link (i, j), and α, β controls the relative importance of pheromone trail levels to distance between nodes.

Notice that, compared to Equation 1, L is no longer a proper subset of the relative complement of M with respect to N, but instead of the entire set of N. This means that the ants are free to choose any nodes next to the ones they are presently at, which could lead to them circling certain areas of the graph while trying to reach the end node.

By sending ant signals through the graph, I look at the probabilistic ant mass across the graph to be compared with the node-based execution method's ant mass. The probabilistic ant mass is essentially the probability of an ant traveling on a certain link, and for any link (i, j) it is obtained with the following equation:

$$\lambda_{ij} = \frac{\sum_{k}^{p}\mu_k}{p} \tag{5}$$

where p is the total number of ants sent through the graph and μ_k is the number of times ant k traveled on link (i, j) in its tour.

The pseudocode for the program is shown in Algorithm 2.

5 The Node-Based Algorithm

The node-based algorithm aims to provide understanding of emergent systems, or in this particular study, ant colonies. Hall [3] claimed that his flow program attains the same ant mass value as the signal-based execution method once the signals settle, except that the path to convergence is deterministic for the node-based execution method while convergence occurs after initial randomness for the signal-based execution method. I ran experiments to test this and show the results in Section 6.

(node-based), which meant that pheromone trail levels across the graph is uniform and will not be updated. In future works investigating the convergence properties past the first cycle (or forward direction), the direct link connecting the start node to the end node would naturally have the highest pheromone trail levels (since that link's distance is the shortest possible connection); this defeats the purpose of testing on graphs with such a 'shortcut', and hence the removal.

Algorithm 2 Path-planning movement using the signal-based algorithm

```
1: procedure SIGNAL_BASED()
2:     Create or load graph                                  ▷ Graph saved in .yaml format
3:     Run the ants_generation_and_activity_cycle function from line 6
4: end procedure
5:
6: function ANTS_GENERATION_AND_ACTIVITY_CYCLE(graph)
7:     Create an array of ant objects ant_array
8:     for each ant in ant_array do
9:         Move ants through graph                           ▷ Ants move via object methods
10:        for each link in graph do
11:            Compute and store data on ant mass
12:        end for
13:    end for
14:    Process data into plots                ▷ Data processing done with self defined function
15: end function
```

In this study, only the forward direction of the ant colony flow program is studied. As mentioned in Section 3, the signals flow in from the start node and move around, before reaching the final node. In this execution method however, the ant (or ant mass) signals differ from the ant signals described in Sections 2 and 4. Here, the ant signals can be thought of as globs of volume, which gets divided and consolidated by the nodes, like how water flows through branching canals and junctions. As shown in the upcoming equations, the ant signals' movement is decided by mathematical functions, and is therefore deterministic.

The flow program for ant colonies executes one node at a time: the nodes receive the signals, process information contained in the signals, redefine the signals (using the processed information), and send them out to other nodes through the connected links. In order to do these, certain functions have to be defined.

Firstly, a 'fold' operator is introduced as a function that aggregates all incoming signals into a node. The fold operator defined is commutative and associative, which combines only two signals at a time to produce a combined signal. The following equation exhibits the fold function in processing ant mass n from two incoming signals x_i and x_j into combined signal x_k:

$$x_k[n] = x_i[n] + x_j[n]. \tag{6}$$

This process is then repeated for all incoming signals until one combined signal remains, which is then passed in to the node to be processed by another operator, called the 'node' function. For the ant colony flow program, the node function is trivial; it returns the folded signal as received:

$$y[n] = x[n]. \tag{7}$$

Next, to send out the folded signal (passed through the node function) a 'split' operator is used to separate the signal. The split function does so with rules obtained from another operator – the 'split-fold' operator. Basically, the split-fold function computes the pheromone trail levels in the outgoing links, and determines which link gets what ratio of the split signal:

$$w_k[p] = w_i[p] + w_j[p], \tag{8}$$

where $w_\eta[p]$, $\eta = \{i, j, k\}$ describes the pheromone trail levels p on the links. Noticeably, the split-fold function combines link states (pheromone trail levels) like the fold function combines signals, until one combined state – which becomes the state of the first link to be branched out by the split function – remains.

With this information, the split function for the ant mass n in signal y_i is defined as follows:

$$y_j[n] = \frac{y_i[n] \cdot w_j[p]}{w_j[p] + w_k[p]}, \tag{9}$$

$$y_k[n] = \frac{y_i[n] \cdot w_k[p]}{w_j[p] + w_k[p]}, \tag{10}$$

where y_j and y_k are the branched signals. Note that the split function simply divides the ant mass into smaller portions, and since the pheromone trail levels are uniform in this study, this leads to a ratio of ant mass based on the number of (outgoing) links connected to the node.

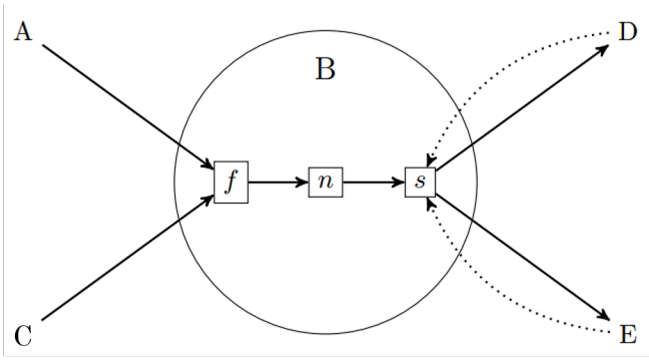


Figure 3: An example node B in the node-based execution method for the ant colony flow program. The signals from A and C are combined with the fold function f, which is then passed through the node function n, and broken up by the split function s, using the information acquired by the split-fold function. The dotted arrows from D and E represent the split-fold function combining the states of outgoing links towards D and E. Figure obtained from [3].

Before the split signals arrive at the next nodes (which they then get folded with other signals, and the whole process repeats), the signals will be passed through a 'link' operator. At least for this study, the link function is quite similar to the node function, in that it passes the ant mass forward to the next node as received[4]:

$$x[n] = y[n]. \tag{11}$$

Hence, the movement of a signal exiting node A (after the node function) and moving towards node B (before the node function) can be viewed as such:

$$A \xrightarrow{s} y[n] \xrightarrow{l} x[n] \xrightarrow{f} B$$

where s is the split function, l is the link function, and f is the fold function.

[4]The link function actually does more than just passing the ant mass forward. In the full program (forwards and backwards directions), the link function processes the average distance traveled for the signals, which also converges [3], as well as updating the link states. For now the focus is only on the ant mass, and thus the link function is trivial in this case.

The pseudocode is shown in Algorithm 3.

Algorithm 3 Path-planning movement using the node-based algorithm

```
1: procedure NODE_BASED()
2:     Create or load graph                          ▷ Graph saved in .yaml format
3:     for each node execution do
4:         Run the node_execution function from 10
5:         Store data obtained
6:     end for
7:     Process data into plots          ▷ Data processing done with self defined function
8: end procedure
9:
10: function NODE_EXECUTION()          ▷ This is defined as an object method in my program
11:     Execute split-fold function    ▷ All the functions defined as object methods in my program
12:     Execute split function
13:     Execute link function
14:     Record data in links
15:     Execute fold function
16:     return data
17: end function
```

6 Experiments and Results

Both the algorithms were tested with various graph sizes. The primary graph size would be one without any links removed (as per the conditions in Section 3), and then the secondary graph is the graph which has the same number of nodes, but now with some links removed until it matches the amount of links of the next primary graph (which has a fewer nodes), and the tertiary graph would have even more links removed to match the following primary graph (with even fewer nodes).

The primary graphs tested were graphs with 5, 10, 15, and 20 nodes. The maximum number of links for each of the primary graphs is 9, 44, 104, and 189 respectively. In Table 2, the node-based data was obtained for 200 node executions, and the signal-based data was obtained for 10,000 ants sent through the graph, averaged over three trials. The data shown obtained is a result of the sum of probabilistic ant mass / ant mass across all the links.

Table 2: The ant mass from the node-based algorithm compared to the probabilistic ant mass from the signal-based algorithm. 'NB' stands for node-based, while 'SB' stands for signal-based.

Number of Nodes	5		10		15		20	
Execution Method	NB	SB	NB	SB	NB	SB	NB	SB
9 Links	6.000	5.999	8.000	8.037				
44 Links			11.000	10.947	14.466	14.456	28.473	28.726
104 Links					15.983	15.913	21.671	21.673
189 Links							20.999	21.009

From the results in Table 2 it is evident that the signal-based algorithm does well to settle within 1% of the node-based results. Hall's deterministic flow program model of ant colonies matches the stochastic behavior of the signal-based model replicating the ACO algorithm (to some extent, as discussed in Section 4). As for convergence rates for the node-based algorithm, node executions required for convergence scale with node number for a constant link number, but decreases with link number

for a constant node number, shown in Figure 4.

Further tests were done on smaller sized graphs of nodes 4 to 6, with varying link numbers to study the convergence behavior on each individual links. One notable occurrence was that as the signals approach the later links[5] of the graph, the probabilistic ant mass on these links converges to a lower value as compared to the earlier links. This means that the signals spend less time of its entire tour going through these late links as compared to the links in the beginning. The ant mass convergence values thus seems to be an indicator of how close the signal is to the end node.

Looking at Figure 5, there seems to be three 'stages' for the signals in the graph – the first three links with common convergence values at 1 represent the earliest stage, then the intermediate stage shared by three other links at 0.66, and finally the last stage is the three links that the convergence values should sum to 1 (since they are all connected to the end node).

This phenomenon seems to depend on how the graphs are connected, not just on the number of nodes and links are present in the graph. I performed further tests on three different graphs with 5 nodes and 6 links, shown in Figures 6, 7, and 8. The results reinforces the notion that the further the links are (relative to other links in the same graph) from the end node, the convergence value is higher. Hence, the earlier the link the signal is in, the more times it will circle in this region of early links, before getting more focused as it wanders into later links.

In addition to that, I found that there is a certain connection that has a convergence value of 1 in Figure 7, which shows up in other graphs of different sizes tested (in particular the graph with 6 nodes, 9 links and 7 nodes, 9 links), but is not always the case as seen in Figure 8. Further tests need to be done for a conclusion to be drawn.

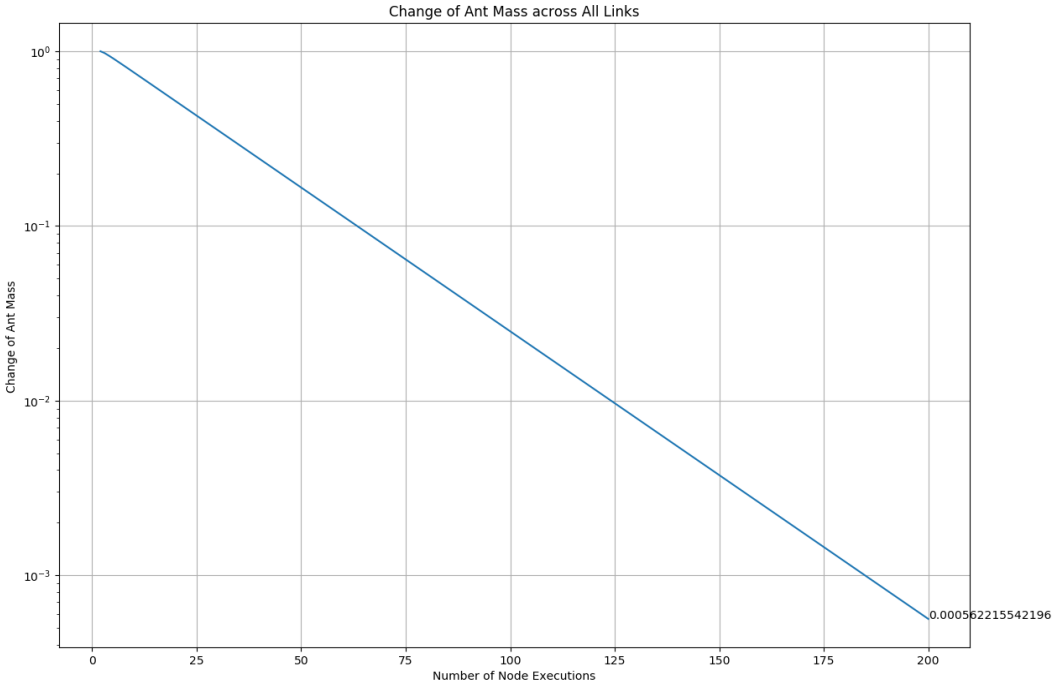
7 Conclusion and Acknowledgements

In this Project Course, the objective was to learn about the Ant Colony Optimization (ACO) algorithm as a machine learning technique for robotic swarms, as well as to gain an insight to the underlying dynamics of the ACO algorithm. I started off by building a program to solve the Oliver30 Traveling Salesman Problem with the ACO algorithm, in order to understand the working principles of the algorithm, and showed results of my solution.

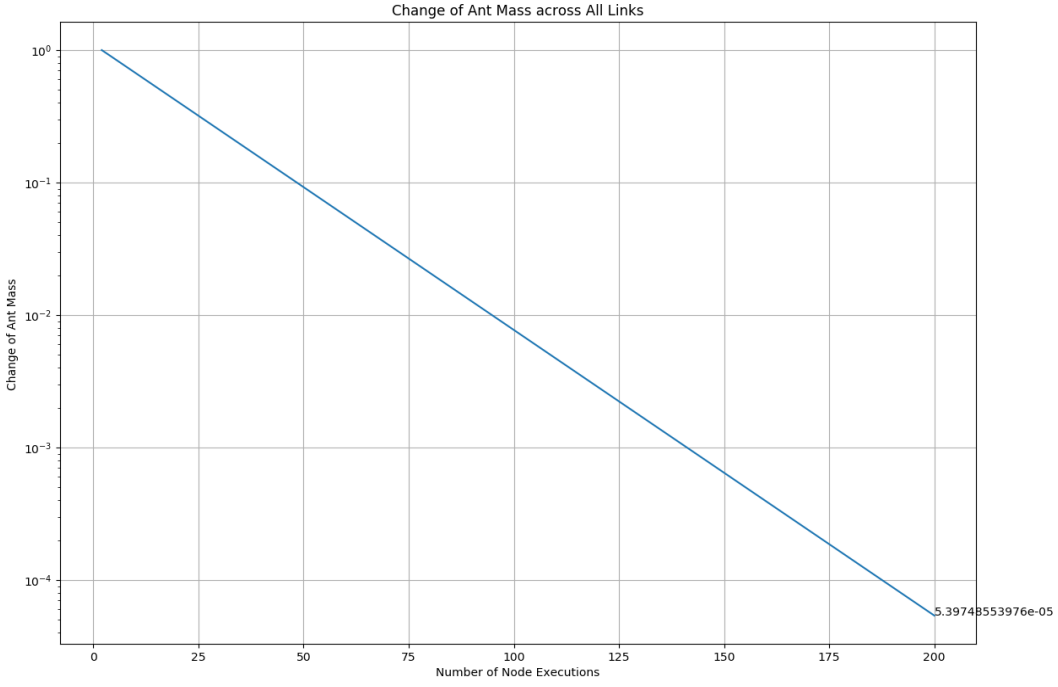
Next, I worked with Jack Hall on using his flow program model to simulate a node-based execution method to compare with the signal-based execution method, which many machine learning algorithm focuses. I simulated the convergence of the ant mass (of the node-based algorithm) and confirm that it matches the convergence value of the probabilistic ant mass (of the signal-based algorithm), as suggested in [3]. Moreover, I did some tests to reveal some interesting signal behavior, in that its randomness slowly fades as it approaches the end node. However, future tests have to be planned along with a bidirectional flow program in order to uncover the underlying dynamics of the ant colony system.

I would like to thank Ph.D. student Jack Hall from the Neuro Engineering Research and Development Lab (NERD Lab) for his guidance on this project.

[5]Links that are much closer to the end node. For example, in a connection of N1—N3—N4—N6—N2—N7 (where N1 is the start node and N7 is the end node), link (2,7) is 'later' than link (6,2), and link (4,6) is 'earlier' than link (6,2).

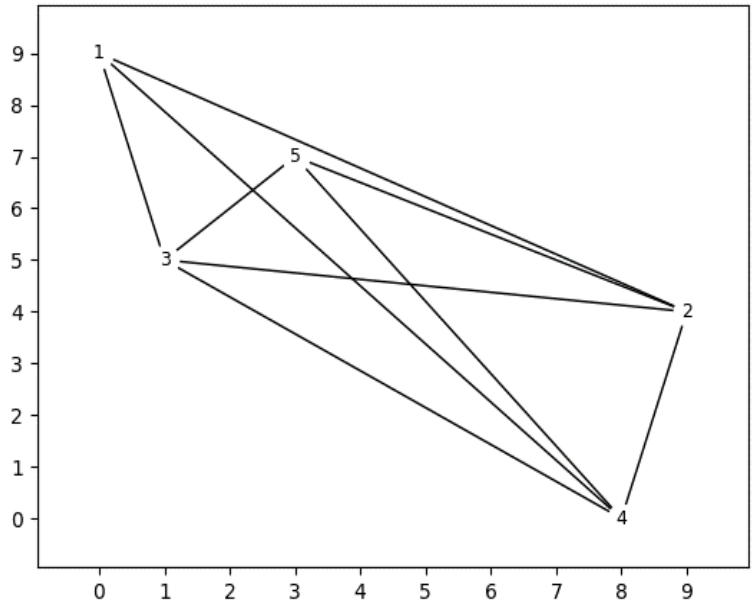


(a)

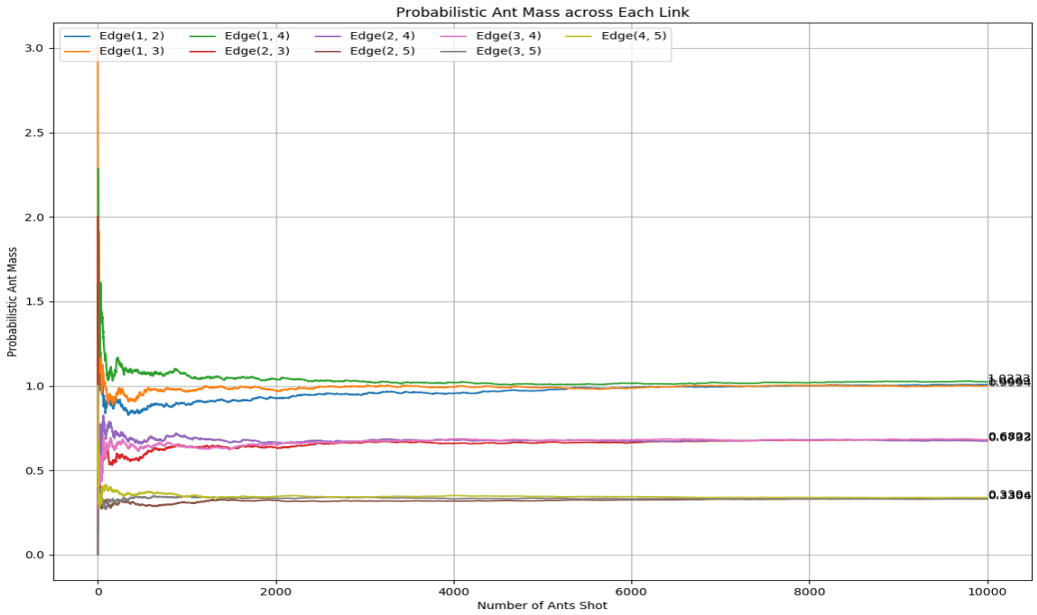


(b)

Figure 4: The change in ant mass in a 20 node graph for the node-based algorithm, with 44 links (a) and with 104 links (b). The convergence seems to be slower for the graph with the fewer links than the one with more links.

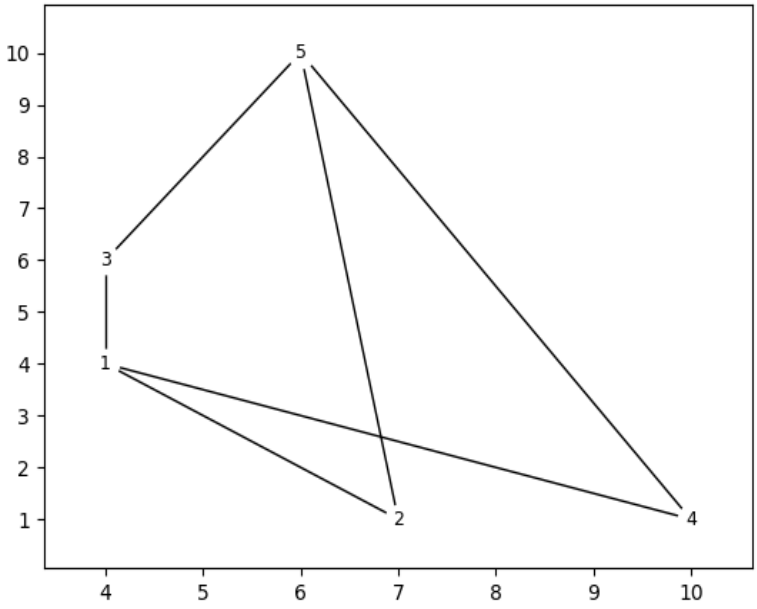


(a)

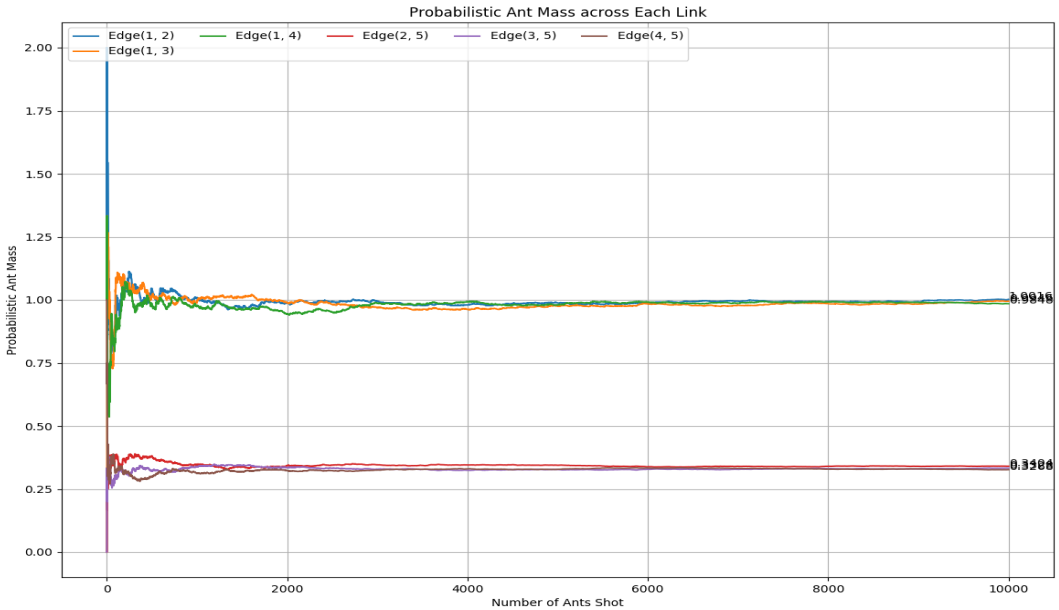


(b)

Figure 5: The graph of 5 nodes and 9 edges (a) and the signal-based probabilistic ant mass distribution (b) of 10,000 ants sent. Notice the convergence value is higher in the earlier links (links that are connected to node 1 and 2) as compared to the links closer to the end node.

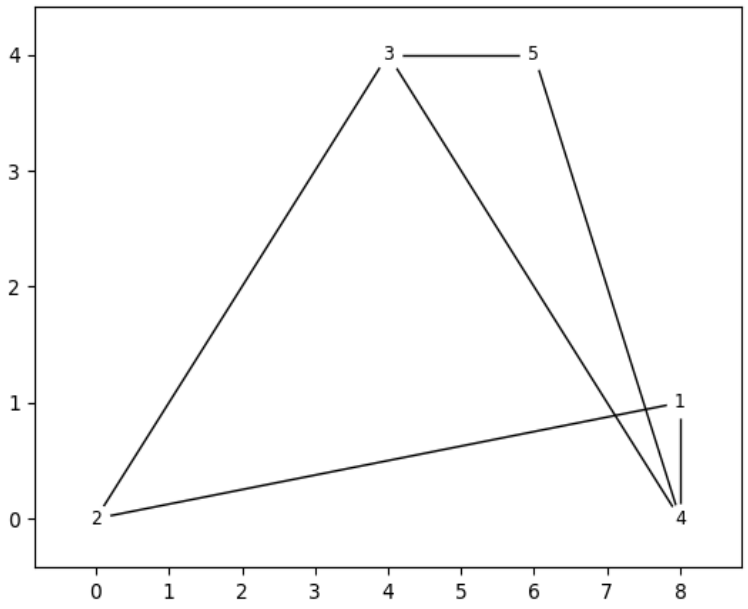


(a)

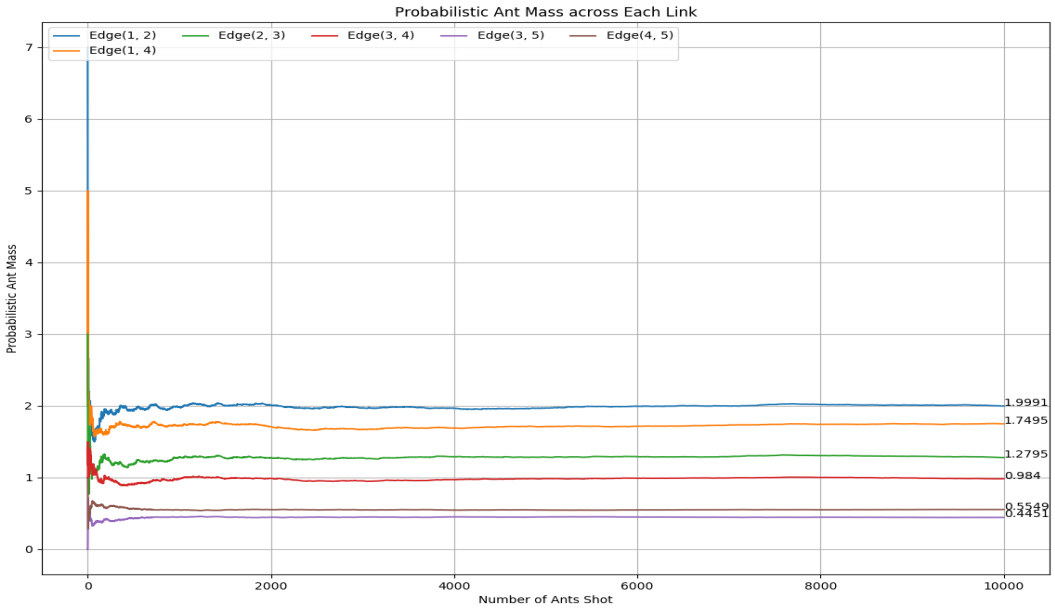


(b)

Figure 6: The first of three graphs with 5 nodes and 6 edges and 10,000 ants sent. In this graph, the start node is connected to the end node of at most two links, which shows in the convergence of the probabilistic ant mass in (b) as two stages – the lower convergence value are links in the later stage than the earlier ones with higher convergence values.

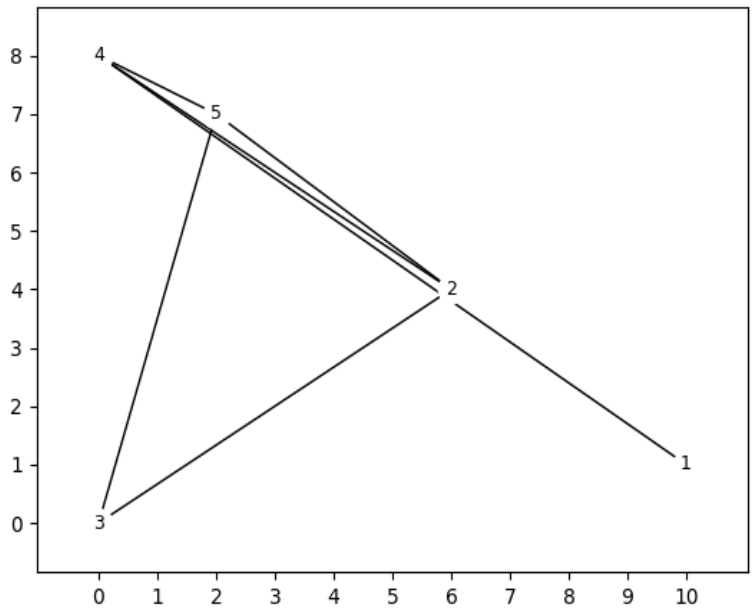


(a)

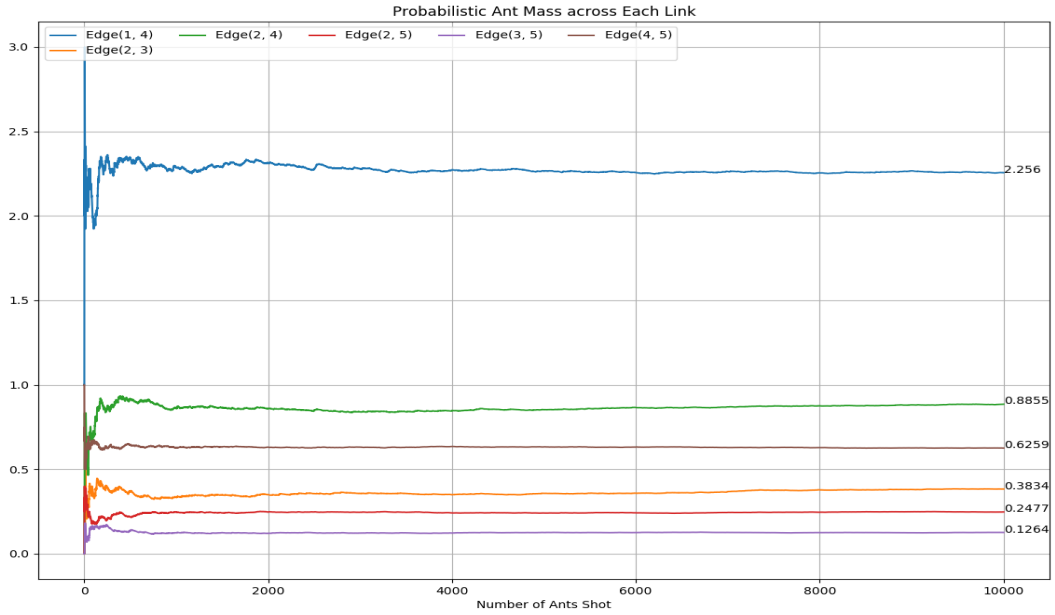


(b)

Figure 7: The second of three graphs with 5 nodes and 6 edges and 10,000 ants sent. In this graph, the maximum number of links between the start node and the end node is 4, while the minimum is 2. An interesting observation was that for link (3,4) connected to nodes 3 and 4, which is connected to the final node via link (3,5) and link (4,5), the convergence value goes to 1. This was a common phenomenon in some other graphs that has such a connection as well – a link connected to two nodes that lead to the final node – although as we see in Figure 8 there are some exceptions.



(a)



(b)

Figure 8: The third of three graphs with 5 nodes and 6 edges and 10,000 ants sent. In this graph, only one link leads out of the start node, and the maximum number of links between the start and end node is 4, while the minimum is 2. As observed in Figure 7, there is such a connection in link (2,4), but the convergence value is not 1. I speculate that this might be related to the way the nodes preceding the end node receive ant mass signals; in this case, node 2 is only accessible to signals that has passed node 4. In the previous figure (and in other graphs where the connection yields a convergence value of 1) the two preceding nodes receive signals independently.

References

[1] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1), 1996.

[2] Heuristic. https://www.merriam-webster.com/dictionary/heuristic.

[3] J. Hall. *Design of Emergence*. Ph.D. Proposal, University of Texas at Austin, 2016.

[4] E. W. Weisstein. Traveling salesman problem. http://mathworld.wolfram.com/TravelingSalesmanProblem.html.

[5] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230, 1987.

[6] C. Zhang, Z. Zhen, D. Wang, and M. Li. UAV path planning method based on ant colony optimization. *2010 Chinese Control and Decision Conference*, pages 3790–3792, 2010.