Ant Colony Optimization Algorithms as applied to path minimisation in Graph Theory (March 2021)

Author: SN.:19052055, Department of Physics and Astronomy, University College London

Abstract – The analogous behaviour of ant colonies as cooperative, algorithmic agents and computational heuristic algorithms is adapted as an approach to solve combinatorial optimisation problems. An ant colony optimisation algorithm is implemented using a probabilistic based algorithm for solving path optimisation problems. The pheromone-based communication of biological ants is used as the predominant paradigm and the main characteristics of this model are positive feedback and the use of a constructive greedy heuristic; positive feedback allows for discovery and reinforcement of optimal solutions, and the greedy heuristic helps find acceptable solutions in the early stages of the search process. We apply this methodology to both a simple, two path Minimal Spanning Tree Problem (MST) and the classical Traveling Salesman Problem (TSP). We discuss the effectiveness of this implementation and explore parameter selection to optimise results.

I. Introduction

ANT colonies achieve organisation and cooperation tasks using chemical signals and simplistic biological directive. These signals and directives allow biological ants to navigate to food sources and achieve other tasks. This is achieved in the real world as when ants find a source of food, they walk back to the colony leaving pheromone trails that show the path leads to food. When other ants come across the trail, they are more likely to follow this path with a set probability linked to the pheromone level- over time as more ants find the path the pheromone level gets stronger reinforcing the path. There will be some diffusion of the pheromone and this leads to optimisation of the path as if a shorter path is also found, as the pheromone level drops every time step, the shorter paths are travelled quicker and thus more frequently resulting in stronger pheromone level (Fig.1 shows this behaviour with an obstacle creating two paths). Hence, this naturally optimises the path, and a similar approach can be used find near-optimal solution to the traveling salesman problem and other optimisation problems.

A. Background:

This approach is applied to computational problems via an ant colony optimization algorithm (ACO), which is a probabilistic technique, inspired by real ants and their pheromone-based communication. This type of algorithm when implement with large numbers of ants can thus produce uniform collective behaviour from the decentralized, self-organised agents; this it is known as Swarm Intelligence and when coupled with the ant algorithm some metaheuristic optimisation can be obtained. This approach was first proposed and implemented by Marco Dorigo in his PhD thesis and later published papers [1][2], his implementation was based on the ant behaviour described above and was implemented to search for an optimal path in a graph. This approach has subsequently been applied to a wider class of computational problems drawing on various aspects of ant behaviour.

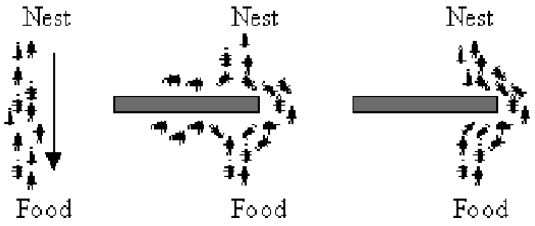


Figure 1: Diagram showing ants path through an environment. Longer path disregarded as pheromone diffuses through faster [3]

Similarly, we will be applying this approach to graph optimisation, this method is advantage over other heuristic algorithms due to its versatile in that it can be applied to alternative versions of the same problem and only requires minimal changes compared to other computational optimisation problems. Furthermore, it exploitation of positive feedback allows refinement of solutions and dismissal of suboptimal solutions, this is advantages over a purely greedy heuristic as it can miss global minima.

We will be applying this approach to first the Minimum Spanning Tree problem (MST) for a simple two path system and the Travelling Salesman Problem (TSP); the MST problem requires the ants to pick the shorter of two paths between two nodes, whereas the TSP is applied to a more complex graph. The travelling salesman requires that we find the shortest route that visit each node once and then return home. Using the same approach as Dorigo we will apply an ant swarm and for this the ants will have some global knowledge as well as the chemical signals.

B. Theory:

Simplistically, the ant system operates as follows; each ant generates a complete spanning tree in which they choose nodes/ edges to traverse probabilistically; ants are more probable to traverse short edges with a high level of pheromone, this is define by *state transition rules* [1][2] as in Dorigo's work. There is then some pheromone updating rule which is applied; for the MST pheromone is deposited each time an ant moves to a new cell/ each tick of the simulation, whereas for the TSP pheromone is deposited once a complete spanning tree is found. The pheromone update also includes some diffusion factor in which a fraction of the pheromone evaporates on all edges- thus edges that are less frequently updated become less desirable. Thus, edges short spanning trees are traversed quicker thus become more desirable as they receive greater amount of pheromone, this can be seen in Fig.2. As we are applying this artificial ant colonies as an optimization tool, as opposed to a simulation of real ant colonies, our system will have some differences from natural systems.

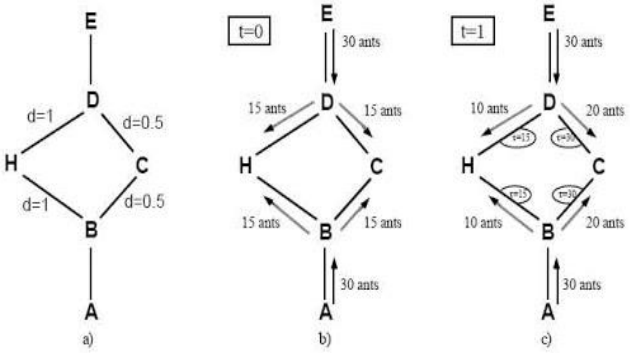


Figure 2: Illustration of artificial ants.
a) Initial graph with distances, analogous of ant colony block by path as seen in Fig.1.
b) Initially at t=0 as no pheromones are laid ants choose both paths with equal probability.
c) As short paths traversal time is shorter at t=1 the pheromone is stronger on shorter edges, therefore, it is more preferred by ants. (Illustration taken from Dorigo report) [1]

Our artificial ants will have some memory, will experience discrete time and for the TSP implementation will have some global knowledge. Thus, given a set of N nodes/ cities the TSP can be stated as the problem of finding a minimal length closed tour that visits each node once. The length of the path between node i and j is defined as d_{ij}, which in Euclidean space is given as:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

Each ant is then a simple agent with the following characteristics:
• They choose edge to travel with a probability that is a function of the distance and of the pheromone level along it
• They have memory of visited sites to constraint the ant to making complete spanning trees and weight movement to new nodes higher
• When all nodes are visited the ant retraces its path, laying a pheromone trail on each edge i, j visited.

The state transition rules for the k^{th} ant thus gives the probability of moving from state i to state j; to satisfy the constraint that an ant visits all the n different sites, each ant has a list containing the unvisited sites, we therefore will weigh the ant to visit unvisited nodes greater than others. When a tour is completed, this list is then emptied, and the ant is free again to choose. Thus, we define visibility, η_{ij}, as the quantity $\frac{1}{d_{ij}}$, from this we define the probability, P_{ij}, of choosing edge from site i to site j for the k^{th} ant as the normalised product of the visibility η_{ij} and the pheromone density, τ_{ij}, along the path ij giving the following equation:

$$P_{ij}^{k} = \begin{cases} \dfrac{\gamma \cdot (\tau_{ij}^{\alpha}) \cdot (\eta_{ij}^{\beta})}{\Sigma_{k \in allow_k} \gamma \cdot (\tau_{ik}^{\alpha}) \cdot (\eta_{ik}^{\beta}) + \Sigma_{k \notin allow_k} (\tau_{ik}^{\alpha}) \cdot (\eta_{ik}^{\beta})} & for \quad j \in allowed_k \\ \dfrac{(\tau_{ij}^{\alpha}) \cdot (\eta_{ij}^{\beta})}{\Sigma_{k \in allow_k} \gamma \cdot (\tau_{ik}^{\alpha}) \cdot (\eta_{ik}^{\beta}) + \Sigma_{k \notin allow_k} (\tau_{ik}^{\alpha}) \cdot (\eta_{ik}^{\beta})} & otherwise \end{cases} \tag{2}$$

where $allowed_k$ is the list of sites i not visited and where α and β are parameters that control the relative importance of pheromone versus visibility and γ is a constant that weights visits to unvisited nodes. Therefore, this transition probability is balance between the visibility, which weights immediately shorter edges more desirable giving a greedy constructive heuristic, and pheromone intensity, which represents indication of the popularity of a move and thus desirability creating a positive feedback structure.

For the MST problem the system is simplified, and the transition rule only considers the pheromone level in first cell of the path when selecting an edge. This treats the ants as much more simplistic, closer to nature, having no memory and as we deal with only two paths. Therefore, for the k^{th} ant the probability, $P_{ij,}$ of choosing edge from site i to site j is given as the normalised value of the pheromone strength, τ_{ij}, in the first along the path ij giving the following equation:

$$P_{ij} = \frac{\tau_{ij}^{\alpha}}{\sum_{k}^{N_i} \tau_{ik}^{\alpha}} \tag{3}$$

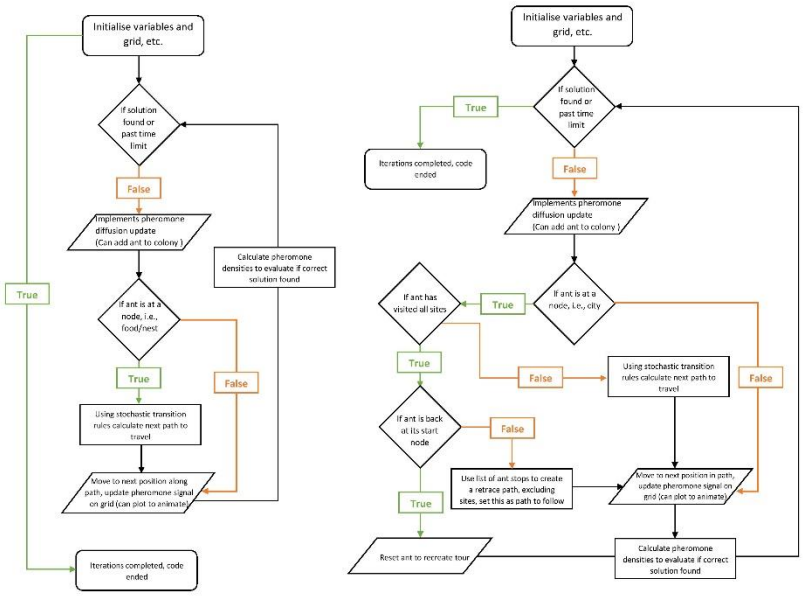
This simplistic two path system works only using pheromone as like the natural ants discussed above simply due the shorter paths quicker traversal time ants will travel the path faster and there will be less time before pheromone diffusion. This works as a positive feedback loop as it becomes preferable to the ants and the longer path will. For the MST model the state transition rule is slightly modified to be time dependent and introduces a constant, δ, which will be called the depression factor. This factor along with inclusion of time dependence acts to depress the importance of pheromone signals in some time interval $\mathrm{t} < \delta$, weighting all paths closer to 50% each, then pronouncing pheromone signal for time interval $\mathrm{t} > \delta$. This is achieved by altering the state transition rule as follows:

$$P_{ij}(\mathrm{t}) = \frac{\tau_{ij}^{\mathrm{t}/\delta}}{\sum_{k}^{N_i} \tau_{ik}^{\alpha}} \tag{4}$$

Thus, in this case as the exponent is set to $\frac{\delta}{\mathrm{t}}$, for the interval $\mathrm{t} < \delta$ the exponent will be less than one, this skews the distribution closer to a Bernoulli distribution with probability 0.5; for the interval $\mathrm{t} > \delta$ the exponent will be greater than one leading to smaller differences in pheromone of the edge having greater effect on probability of the edge choice (meaning its more desirable). This is used as for small path lengths the difference in traversal time is smaller enough, the longer path can become favourable as pheromone levels do not disperse any quicker than the short path. However, this depression factor can be interpreted as some initial depression period in which the ants do not considering the pheromone level as highly in their state transition rule.

C. Generic Algorithm:

The general form of the code for both systems is as follows: the variables and grid are initialised, the ants initialised to have be equally numbers positioned on each site/city. The code loops over time, counting ticks, the loop breaks either when total time limit passed or predetermined solution is found. During the main loop each ant is looped over, if the ant is at the position of a site the state transition rules are applied to pick the next path for the ant and advance it position one cell (this uses two for loops- one over all ants and one over all sites). If the ants is not at a site the ants advance their position each tick depending on its selected path. Ants therefore complete a spanning tree by repeatedly applying the stochastic greedy transition rule. For the MST problem while constructing the spanning tree, an ant modifies the amount of pheromone on the visited edges by applying adding a set pheromone incriment, the algorthim for this is reported in Fig.3a . For the TSP the pheromone update is slightly modified. Pheromone is not deposited by ants untill they have visited all the sites/ cities; once all the sites are visited the ants set to retrace a path along all the edges visited, as it retraces this path each tick it deposites pheromone on the grid. This is done as similar . The algorthim for this is reported in Fig.3b.



a. *b.*

Figure 3:
a) Flow diagram of generic algorithm for individual ants for the MST.
b) Flow diagram of generic algorithm for individual ants for the TSP .

For both systems during each loop the the pheromone level on the edges is also modified by applying the global pheromone update rule; this rule acts as a diffusion of pheromone as the pheromone is decreased by a factor ρ, in which $0 < \rho < 1$. These pheromone updating rules are thus intended to give more pheromone to edges which should be visited by ants, edges part of the shorter spanning trees and the diffussion allows for removal of incorrect paths and give positive feedback for desirable paths.

II. MINIMUM SPANNING TREE PROBLEM (MST)

A. *Methodology and Code:*

Given the outline in the preceding section, the ants algorithm for the simplistic case of the MST implementation follows. At time zero ants and variables are initialisation, during this phase the grid which is a two-dimensional array is constructed and the sites are initialised; the sites are represented by dictionaries containing the nodes position and the edges out of the node to the other nodes- as relatively simple, in this model, edges are hard coded paths set as tuple lists. Ants, which are represented by a dictionary containing their individual characteristics, are positioned at the nest site; the ant dictionary contain their unique number, current position, the path/edge they have chosen to travel- a code segment for this is displayed in Fig.4. Initial values of τ_{ij}, for pheromone intensity is set on edges. Thereafter, for each simulated time tick, every ant moves from site i to site j choosing the edge to traverse using the state transition rule with the probability being function of parameters δ and t, see formula (4). As the ants travel the selected edge, they deposit pheromone in each cell this gives information about how many ants have chosen that edge (i, j).

```python
def add_colony(colony, N, pos, sites):
    '''Adds N ants at node 'pos' in sites dictionary, to the colony dictonary:
    This will be a nested dictonary of ants 0-N, they will each have a dictonary containing there number,
    positon and path they'll follow'''

    # Loop adding N ants to the dictonary, finds size of existing colony array and adds N
    for n in range(len(colony), len(colony) + N):
        colony[n] = {
            'ant'   : n ,
            'pos'   : sites.get(pos)['pos'],
            'path'  : ()}

    return colony
```

Figure 4: Code snippet showing definition of function to create ant dictionary, storing individual variables.

This process iterates over the simulated time and is stopped either when some pre-set time limit is reached or one of the paths reaches a maximum tolerance of pheromone density compared to the rest of the grid. Formally the algorithm is:

1. Initialize:
Set t=0 **{t is the time counter}**
Set variables such as δ, ρ, P, c and T
(Depression factor, diffusion const., const. pheromone increment, max. pheromone, and total time)
Set up grid as array as 2D grid
Initialise edges as user inputs, set up nodes and ant dictionary
(Ants dict. referred to as colony and nodes dict. referred to as sites)
For every edge (i,j) set an initial value τ_{ij}(t)= c,
(where c is the constant increment of pheromone)
Place the N ants on the M nodes.
Set solved = False

Loop over until t==T or until solved == True:
2. Apply global pheromone update, for all τ_{ij}
$\tau_{ij}(t+\Delta t) = \rho \cdot \tau_{ij}(t)$

3. For each ant in colony:
For all sites:
If ant kth is at site:
Choose the node j to move to, using state transition rule given by equation (4)
Else:
Move the ant to next index in its path

4. Calculate the pheromone density over either edge,
If the pheromone along the edge is greater then percentage tolerance of pheromone density of both edges, set solved = True, set solution as variable

If solved = False increment loop again,
Else: Print shortest edge

Stop

The implementation of this code gives successful converged on the correct path, the proceeding section discusses parameter setting and the optimisation of this system. The results of two simulations are show in Fig.5, these simulations are of graphs on a 22×22 grid, they have equal length short paths, however, ***a*** has a long path $3 \times$ the size of the short path, whereas ***b*** has a long path $2\times$ the size of the short path. This results in, as expected, the graph with the bigger difference in edge length takes less time to converge on a solution; this is as the ants the longer traversal time means the long paths pheromone diffuses much quicker relative to the reinforcement of the short path. This results in much faster pheromone build up on the short path as the long paths pheromones density remains diffuse- this can be seen as although both simulations have same percentage of pheromone density along the short path, the absolute pheromone density of the short path on for simulation ***a*** is only 61.93 compared to **b** which is 79.63. This shows that the faster convergence on a solution is from a quicker drop off in density along the long path.

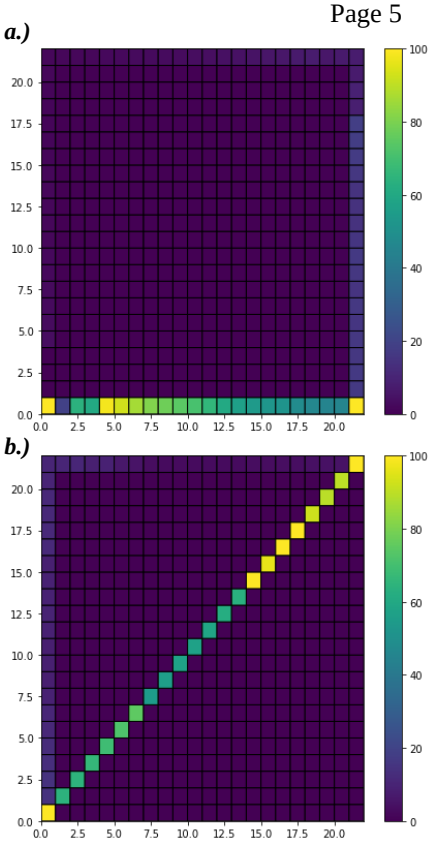


Figure 5: Diagram showing MST simulation results for c=1, P =100, $\rho = 0.9$, and $\delta = 40$.

a) Long path 3x length short path. Short path found in 91s with path density of 61.93 and 90.023% of pheromone density of whole graph.

b) Long path 2x length short path Short path found in 110s with path density of 79.63 and 90.464% of pheromone density of whole graph.

B. Experimental study: Parameter setting and optimisation:

From this several tests are implemented to investigate the relative importance and optimal values for the parameters. This is not a fully developed mathematical analysis of the models, but a collection of stochastic simulations used to collect statistical data for this purpose of optimising the simulation. These simulations are carried out on graphs with a short diagonal path and long path around the perimeter, as seen in Fig.5a, this graph can then be varied to be different size and taking in different variables values.

The parameters considered are those that directly or indirectly affect the computation of the probability in formula (3) and (4):

- α: constant factor that skews probability distribution;
- δ: the depression value;
- ρ: diffusion/ evaporation rate, $0< \rho <1$ (1- ρ can be interpreted as pheromone persistence

Alpha value:

First, we investigate the effectiveness of the code when using a set alpha value vs. the depression method. We set the grid size to be 31×31, with pheromone increment, c=1, and maximum pheromone being, P=100. The evaporation rate is set to ρ=0.85 and we run for 30 initial ants, with one added each tick, and the cut of time $T = 500s$. We have the percentage tolerance for a solution being 90% then run the simulation 400 times using the state transition rule (3) and taking different random values of α where $0<\alpha \leq 10$. *The* results are shown in Fig. 6., for all values of alpha there is equal spread over both the short and long path (with the long path taking loner). This is confirmed from the simulation giving 47.25% successfully rate in finding the short path and 42.00 % of long; 10.75% did not converging in the 400 ticks and this correlate to $\alpha < 1$ as these values skews the probability function to random 50/50 choice independent of pheromone level. This indicates for this implementation having any set alpha value will not yield consistent results; this seems to be because, as stated previously, there is chance for the ants to randomly favour the long path in the begin and thus the incorrect result is found as the positive feedback reinforces it.

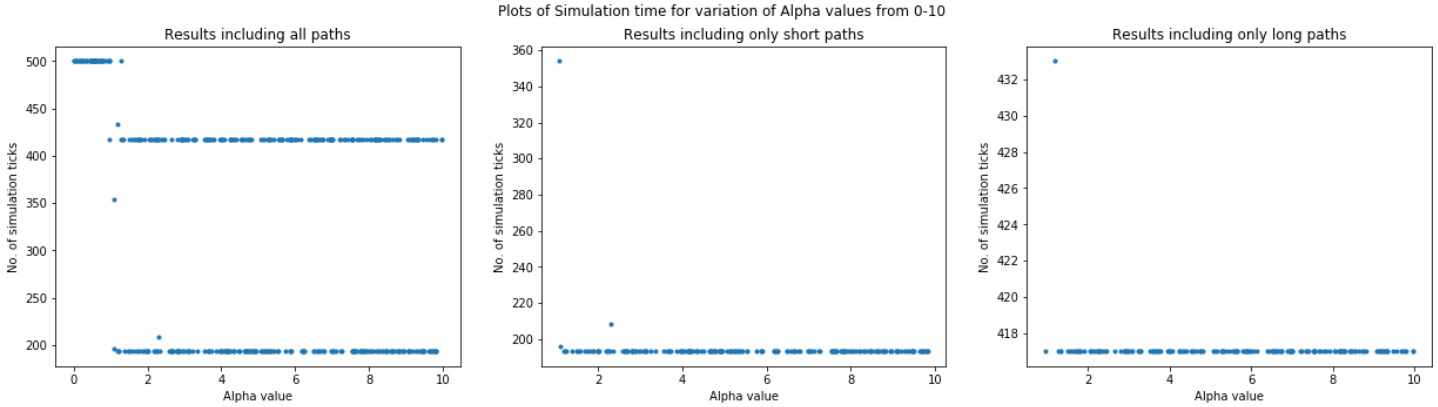


Figure 6: Plot for alpha value against simulation time, for 400 simulations

Depression value:

Now looking at the depression constant, using the same simulation variables but instead implementing equation (4) as the state transition rule and then varying the depression constant, δ, randomly such that $0 < \delta \leq T$. The "depression state transition rule" gives much better convergence on the short path compared to long path, however, it still has a low percentage of success for the whole range chosen- percentage of success is 56.75% and percentage converging on long path is 4.50 %. From Fig. 7 the 38.75% that fail to converge on a solution is a result of depression values, $\delta \gtrsim 300$. This is as the period in which pheromone signals are depressed in the transition rule is so long that the level of pheromone built up on each edge is so large that afterwards a solution cannot be decerned. In comparison, for depression values, $\delta \lesssim 60$ there is 100% convergence on the short path, all of which are in 200 steps, above this threshold (for $60 \lesssim \delta \lesssim 300$) there is a linear relation between the depression value and the number of time steps taken. This is as above 50 the depression factor acts to delay the system in finding a solution as it skews the probability of either path to be 50/50. Interestingly, this threshold, $\delta = 50$, corresponds to twice the grid dimension, i. e $2 \times 31 \approx 60$ which is the length of the long path.

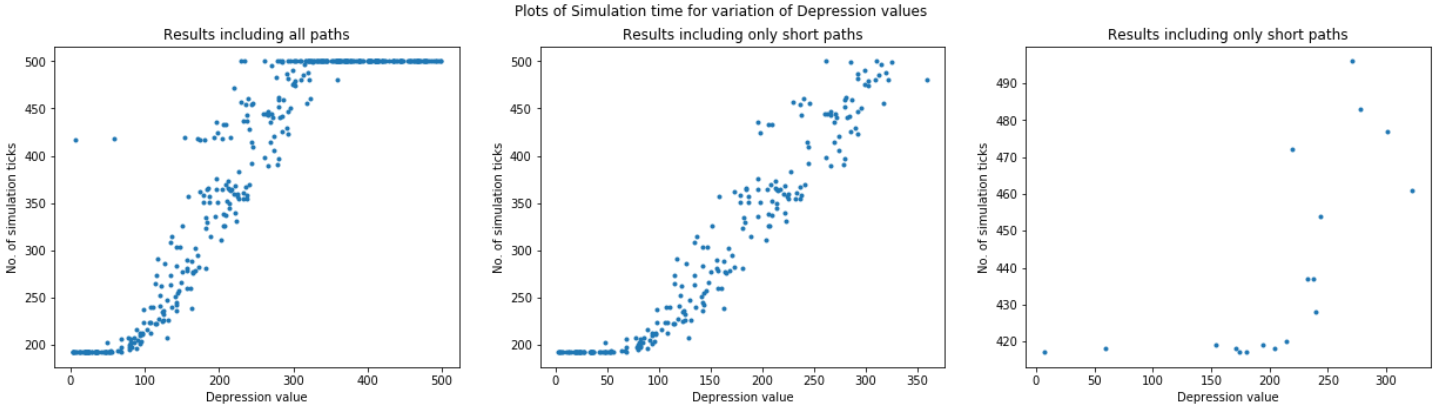


Figure 7: Plot for variation of depression value

This leads to looking at how the simulation varies with scaling the grid size. In implementing this, using the same variables as above (for constant depression value cut of time increased to 1000), the grid dimensions are varied from 2 to 101 (i.e., 2×2 *to* 101×101), meaning the short and long paths vary from 2 to 101 and 4 to 202, respectively. Two sets of simulations are implemented, one in which the depression constant, δ, is set to 40 for all simulations and another where we set it to be proportional to the grid size (i.e., varying between 2-101). From Fig.8 and Fig.9 varying the depression value with grid size gives much better convergence; for the constant depression value we get 81.75% of success compared to 95.0% for the scaled depression value. The constant depression value like alpha splits the results into two lines, one of the slower speed simulations for the longer path and one for the quicker shorter path simulations, this accounts for the lower success percentage, giving the scaled depression rate as more optimum parameter. The varying depression constant also produces a system the scales linearly with grid dimension which agrees with our model as this implementation relies directly on the physical traversal times- thus grid/ path length. Fitting a line for the results from the scaled depression value simulations the time taken for the simulation thus scales b 4.94 times the grid/ short path size.

This is then extended to find the optimal scaling of the depression value with the grid, we run 50 simulations for a grid of 40x40 with the depression value by $40 \times i$, recording the percentage of success, we then run 100 more of these set of

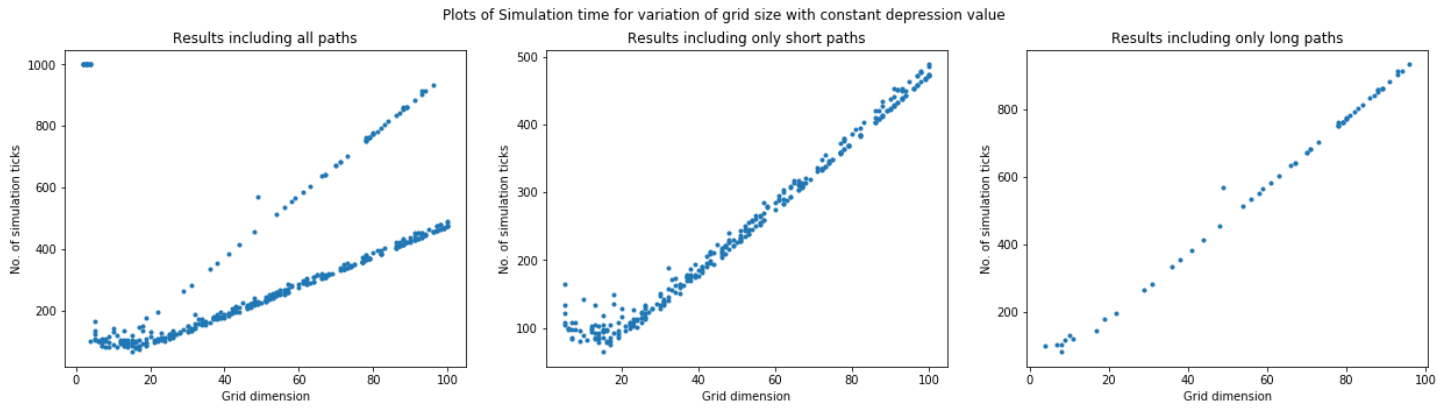


Figure 8: Plot for variation of grid with constant depression value:
Percentage of convergence on short path 81.75%
Percentage of convergence on long path 14.75

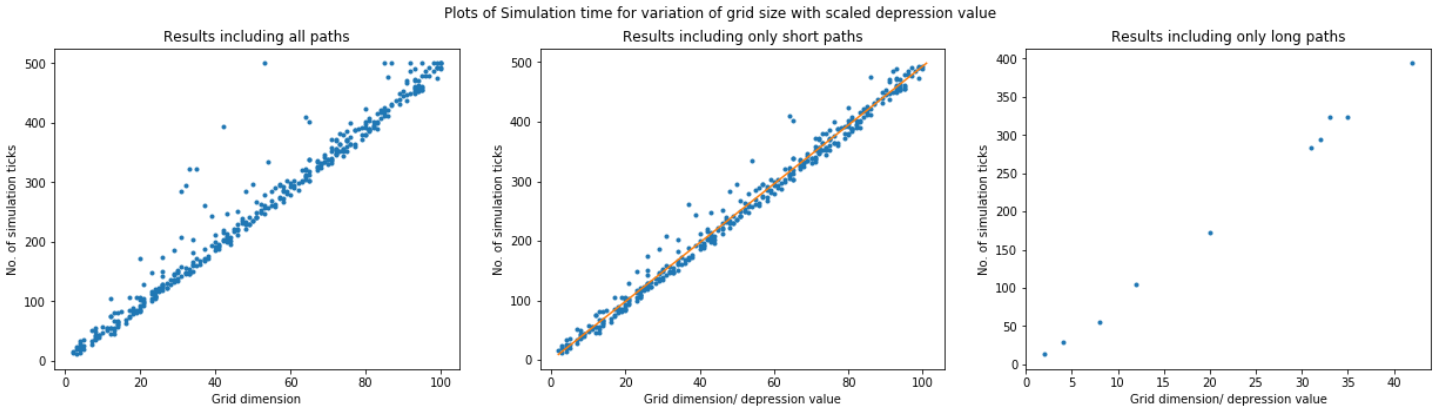


Figure 9: Plot for variation of grid with scaled depression value:
Percentage of convergence on short path 95.0
Percentage of convergence on long path 3.5

simulations, varying i for each such that $0 < i \leq 10$. Plotting these results, seen in Fig.10, give a plot displaying the efficiency of each scaling value; above 8 (i.e., $8 \times$ grid dimension) the code finds the short path for none of the simulations then below 8 the data shows quadratic behaviour. Using a quadratic fit we then can use the fitting parameters to find the maxima of this curve and thus optimal scaling. We thus find the optimal scaling is 1.89 ± 0.46, this agrees with the threshold value discussed above when analysing the variation of the depression value with a fixed grid in Fig. 7; this threshold corresponded to the value beyond which there was a linear relation between the depression value and the number of time steps taken.

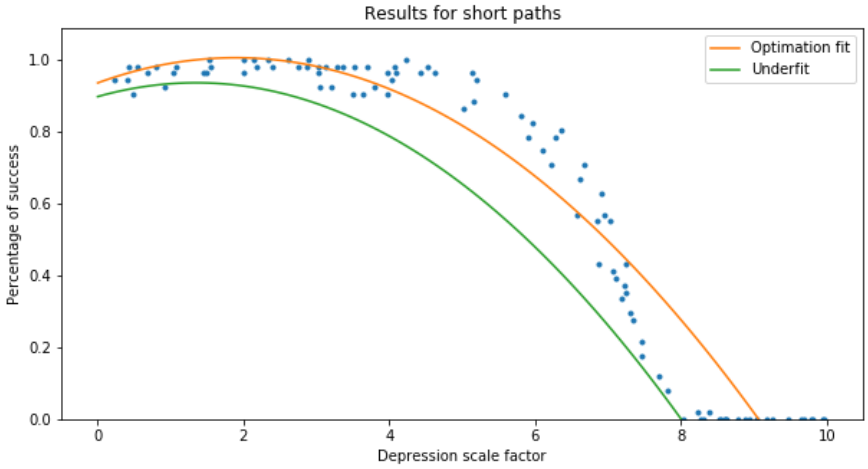


Figure 10:
Plot showing MST simulation efficiency for varying depression scale factor.

Optimal value of scaling is 1.89±0.46.

Diffusion Value:

Next, we look to optimise the diffusion value, we run similar simulations as above, we set the grid size to be 31×31, with as before pheromone increment, c=1, and maximum pheromone being, P=100. We run for 30 initial ants, with one added each tick, and the cut of time $T = 750s$. We have the percentage tolerance for a solution being 90% then run the simulation 400 times using the state transition rule (4) and taking randomly chosen evaporation rates for each simulation such $0<\rho<1$. The results, seen in Fig. 11., show for $0.6 \lesssim \rho \lesssim 0.95$ there is a negative linear relation between the simulation time and the diffusion value; in comparison for $\rho \lesssim 0.6$ and $\rho \gtrsim 0.95$ the simulation could not converge on a solution in the time limit. This is as for $\rho \gtrsim 0.9$ the diffusion is too weak for the long path pheromone to disperse and for some value $\rho \lesssim 0.6$ the diffusion is too greater for pheromone to build on either edge. This gives a skewed hyperbolic relation which is asymptotic at $\rho \approx 1$ and some value $\rho \lesssim 0.6$ (A hyperbola line is overlayed over the data for visualisation- not fitted but an approximation). This gives the optimal diffusion value at the minima of this hyperbola, such that $\rho \approx 0.95$.

Optimal Values:

Overall, we get the optimal values for the simulation are such the pheromone diffusion, $\rho = 0.9$, and the depression value, $\delta = 1.89 \times$grid size. Running 1000 simulations with these values give success percentage of 98.25%.

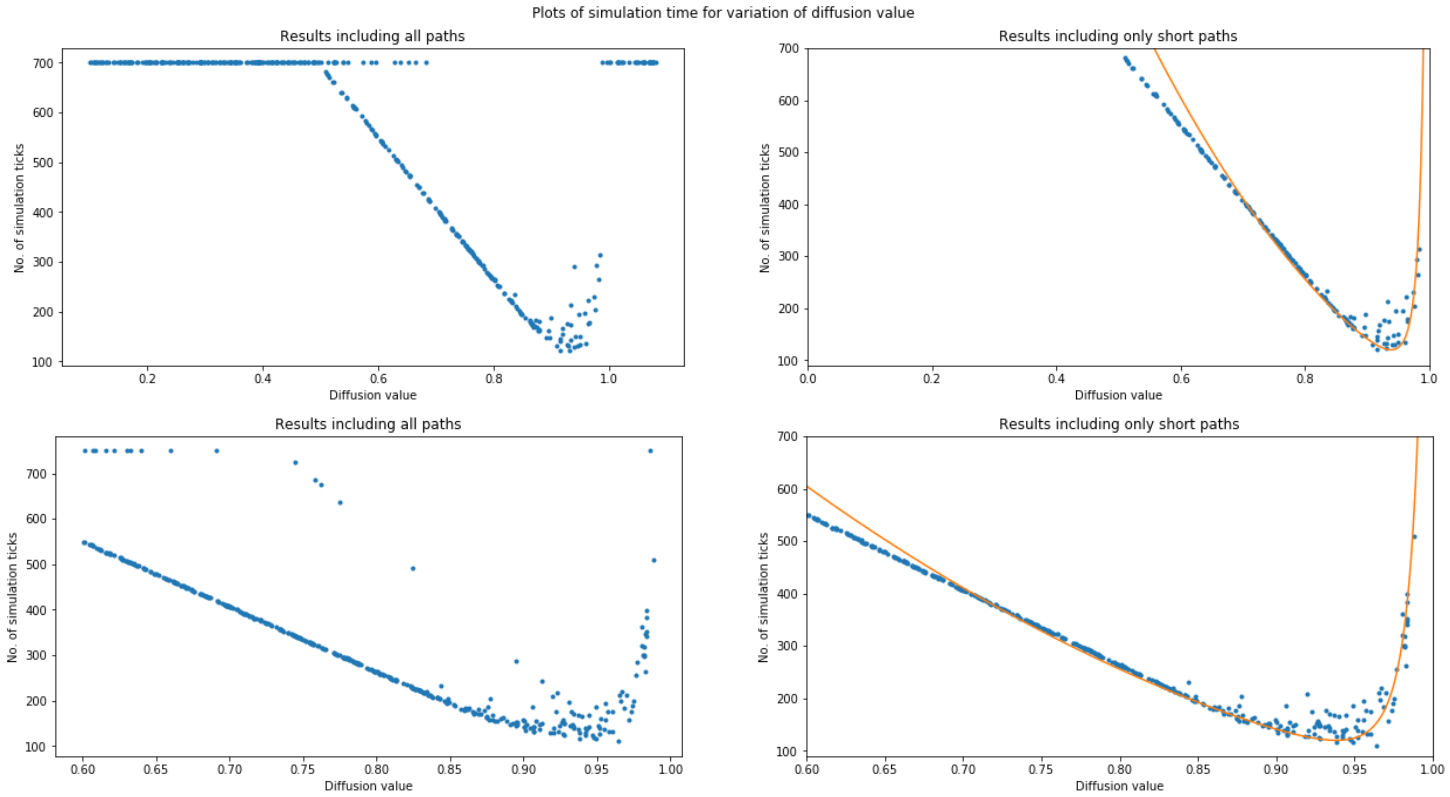


Figure 11: Plot for variation of pheromone diffusion constant
a) Shows results for 400 simulations with 0< ρ < 1.
b) Shows results for 400 simulations with 0.6< ρ < 1.

III. TRAVELLING SALESMAN PROBLEM (TSP)

A. Methodology and Code:

Given the outline in the preceding section, the ants algorithm for the TSP implementation follows. At time zero ants and variables are initialisation, during this phase the grid which is a two-dimensional array is constructed and the sites are initialised; the sites are represented by dictionaries containing the nodes position and the edges out of the node to the other nodes- in this model a function is made that produces this dictionary from list of locations and connections (Fig.12a). Ants, which are represented by a dictionary containing their individual characteristics, are positioned in equal number at the all the sites; the ant dictionary contain their unique number, current position, their start node, an entry for cells the ant stops at, the path/edge they have chosen to travel, their index in their path and the sites yet to visit (a code segment for this is displayed in Fig.12b). Initial values of τ_{ij}, for pheromone intensity is set on edges. Thereafter, for each simulated time tick, every ant moves from site i to site j choosing the edge to traverse using the state transition rule with the probability being function of parameters $\alpha, \beta,$ and γ, see formula (2). Once an ant has completed a spanning tree, such that the site key in their dictionary has no values, the ant retraces its steps back to the start node deposit pheromone in each cell; this gives information about how many ants have chosen that edge (i,j). This process iterates over the simulated time and is stopped either when some pre-set time limit is reached, or the user-set solution reaches a maximum tolerance of pheromone density compared to the rest of the grid. Formally the algorithm is:

```
1. Initialize:
    Set t=0                                                {t is the time counter}
    Set variables such as α, β, γ, ρ, P, c and T
      (State transition variables, diffusion const., const.  pheromone increment, max. pheromone, and total
    time)
    Set up grid as array as 2D grid
    Initialise user inputs node and edges, set up site and ant dictionary using functions
      (Ants dict. referred to as colony and nodes dict. referred)
    For every edge (i,j) set an initial value τ_ij(t)= c,
      (where c is the constant increment of pheromone)
    Place the N ants on the M nodes.
    Set solved = False
```

Loop over until t==T or until solved == True:

2. Apply global pheromone update, for all τ_{ij}

$\tau_{ij}(t+\Delta t) = \rho \cdot \tau_{ij}(t)$

3. For each ant in colony:
For all sites:
If ant kth is at site:
If ant sites key is empty:
If it isn't its start node, set retrace path, which excludes nodes, otherwise reset ant
Else:
Choose the node j to move to, using state transition rule given by equation (2)
Else:
Move the ant to next index in its path- If ant site key is empty update pheromone on each move

4. Calculate the pheromone density over either edge,
If the pheromone along the edge is greater than percentage tolerance of pheromone density of both edges, set solved = True, set solution as variable

If solved = False increment loop again, Else: Stop

a) Code that creates tuple list, from Point To Point, used in defining edge between nodes

```python
# 21/02/21

def ptp(r0, rf):
    '''Returns path from point to point
    Input:
    r0  Tuple of start of path , form (y,x)
    rf  Tuple of end of path , form (y,x)'''

    # Extractin x and y positions
    y0, x0 = r0
    yf, xf = rf

    # If x difference is greater than y change, use x difference number points in path
    if abs(yf-y0) <= abs(xf-x0):
        d = abs(xf-x0)+1

    # If y change is greater than x change, use that number points in path
    else:
        d = abs(yf-y0)+1

    # Create arrays of the x and y values for the tuples along path
    # As we ensure the biggest difference is used we ensure we dont miss any grid points
    # We ensure the arrays are integers, if one difference is much smaller than other we get duplicates of values
    # Given steeper or shallow gradient, but point always connected, if difference of one is zero, gives array of 0
    y = np.linspace(y0, yf, d, dtype= int)
    x = np.linspace(x0, xf, d, dtype= int)

    line=[]
    # Joins the x and y array to get list of cells in path
    for i in range(d):
        line.append((y[i],x[i]))

    return tuple(line)

```

b) Code that creates graph/ sites dictionary, containing node positions and edges out

```python
# 22/02/21
def graph(loc, links):
    '''Takes list of nodes(locations) and a string list of edges(links) between them, and sets up a dictonary of nodes
    with their positions and edges specify paths.

    Inputs:
    loc     Ordered array of node location as tuples, ie node0 in 0th position in form (y,x)
    links   String list of connections between nodes in form, node0_1, where 0 would be the start node and
            1 would be the end node. Condition for string must be in form node0_1

    Outputs:
    graph   Dictionary containing all nodes and their positions, and the edges out'''

    graph = {}
    for i in range(len(loc)):
        # Adds to dictonary item as string of form node0 where 0 is the node no
        # The value of each item is another dictonary with first item is their pos
        graph['node'+str(i)] = {'pos' : loc[i]}

    for l in links:
        # Links should be in form node0_1(as long as it ends in 0_1), so take  last 3 letters and remove underscore
        # This gives sting of two numbers, the start node then end node, this are then used in ptp function
        nd1 = l[4:].replace('_', ' ')
        nd = nd1.split()

        # Set item of sub dictonary for each node as path from start to end point extracted above
        # Using the string produced to call start and end point into ptp function,
        # Used sting addition to call pos of items from the dictonary itself;
        # Note: `graph['nd'+nd[i]]['pos']` gives position of node nd[i], i={0,1}

        # Update 'node'+nd[0] (start node) of graph dictonary, by adding ['node'+nd[1] (end node)
        # The value of which is the path from start to end,
        graph['node'+nd[0]]['node'+nd[1]] = ptp(graph['node'+nd[0]]['pos'], graph['node'+nd[1]]['pos'] )

        # Update 'node'+nd[1] (end node) of graph dictonary, by adding ['node'+nd[0] (start node)
        # The value of which is the reverse path- end to start
        graph['node'+nd[1]]['node'+nd[0]] = graph['node'+nd[0]]['node'+nd[1]][::-1]

        # Thus each node has edges to another node, set as the node its going to

    return graph
```

c) Code that creates ant dictionary, contain whole colony and individual ant variables

```python
# 01/03/21

def add_colony_SM(colony, N, node, sites):
    '''Adds N ants at node 'pos' in sites dictionary, to the colony dictonary: This will be a nested dictonary of ants
    0-N, they will each have a dictonary containing:

    Pos     The current position of the ant, tuple (y,x), at start will be the site they are initialised to
    Start   The start position of the ant, tuple (y,x), remaining unchanged
    Stops   List of tuples (y,x) of cells the ant has visited (not including sites)
    Path    The tuple list of tuples the ant will follow
    Index   The ants position in its path
    Sites   The sites the  ant is yet to visit

    Inputs:
    Colony  The colony dictonary to add ants to, can be empty or already populated
    N       Number of ants to add
    Node    The start node the ant begins at
    Sites   Sites dictonary containing all sites and paths '''

    # Loop adding N ants to the dictonary, finds size of existing colony array and adds N
    for n in range(len(colony), len(colony) + N):
        colony['ant '+ str(n)] = {
            'pos'   : sites.get(node)['pos'],
            'start' : sites.get(node)['pos'],
            'stops' : [sites.get(node)['pos']],
            'path'  : (),
            'index' : 0,
            'sites' : list (sites.keys()) }

    return colony
```

Figure 12: Code snippet showing definition of function for TSP to create edges(a), sites(b) and ant(c) dictionary, storing individual variables. (Use of dictionary suggested by project lead)

The implementation of this code successfully converges on the correct route for various graphs of varying size, the proceeding section discusses parameter setting and the optimisation of this system. The results of two simulations are show in Fig.13, these simulations show the versatility as they vary in both size and arrangement. We will use the graph in Fig. 13b to investigate the optimal variables.

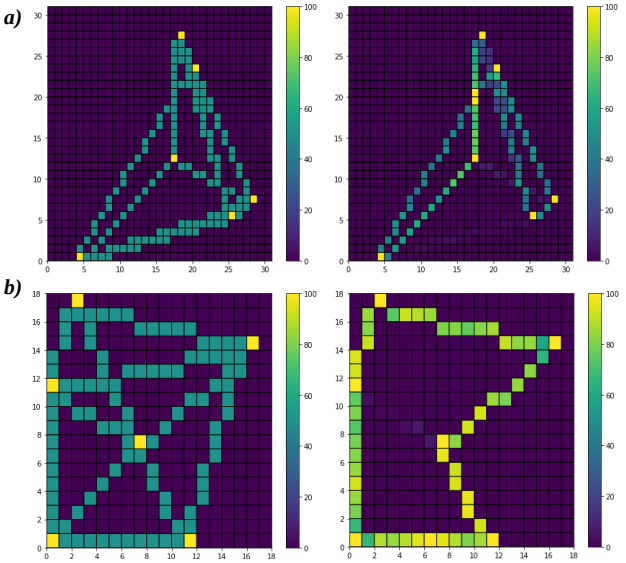


Figure 13: Plot of graphs and solution produced by TSP code

B. Experimental study: Parameter setting and optimisation:

The parameters considered are those that directly or indirectly affect the computation of the probability in formula (2):

- α: Constant that weighs importance of pheromone strength;
- β: Constant that weighs importance of path length;
- γ: Constant that weights visits to unvisited nodes;
- ρ: diffusion/ evaporation rate, 0< ρ <1 (1- ρ can be interpreted as pheromone persistence

For all tests, the simulations are run on the graph in Fig.13b, which has grid size of 18×18, with pheromone increment, c=1, and maximum pheromone, P=100. This is run for a colony of 300 initial ants, with 5 added each tick to a random site. The maximum time, $T = 1000$s and percentage tolerance is set at 90%.

Diffusion Value:

We look to optimise the diffusion value thus we run 100 simulations taking randomly chosen evaporation rates for each simulation such $0<\rho<1$. The results, seen in Fig. 14. Show the same trend as discussed in the section above. The data has asymptotic behaviour for $\rho \approx 0.95$, in which pheromone does not evaporate enough to converge on a solution and does not converge for $\rho \lesssim 0.5$. This trend thus gives an optimal diffusion constant of $\rho \approx 0.90$.

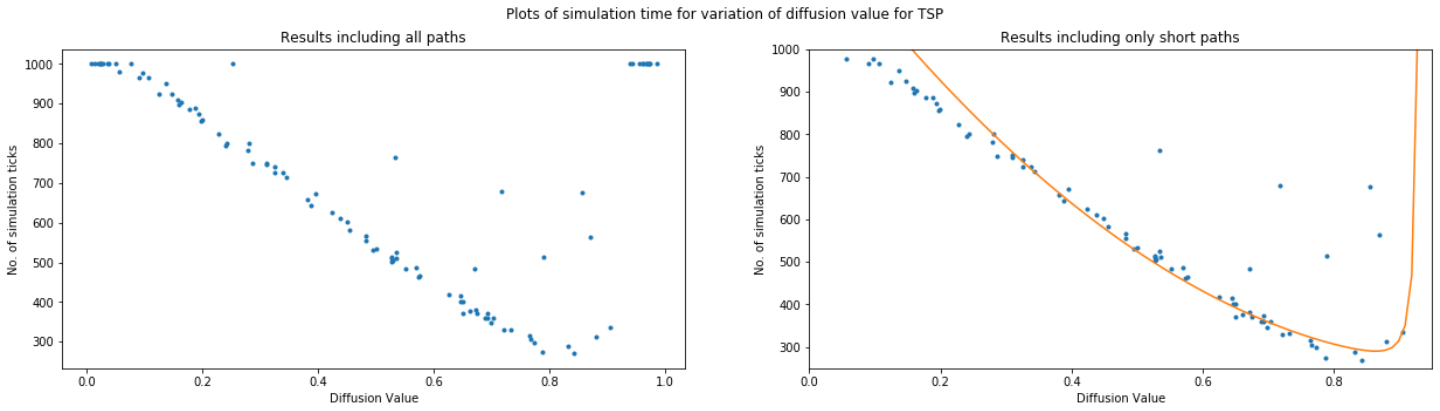


Figure 14: Plot of graphs and solution produced by TSP code

Gamma Value:

Next, we look to optimise the gamma value thus we run 100 simulations taking randomly chosen gamma values for each simulation such $1<\gamma<100$. The data shows an no proportionality between gamma and simulation time, as seen in Fig. 15, the simulation times for all $\gamma > 5$ all simulations take approximately the same time. For all $\gamma < 5$ we see that no solution can be found, this is as ant do not find new nodes favourable, this can lead to large loops before the ants retracing their solution and thus the pheromone on each edge will not directly correspond to its favourability. This is discussed further in proceeding discussion section.

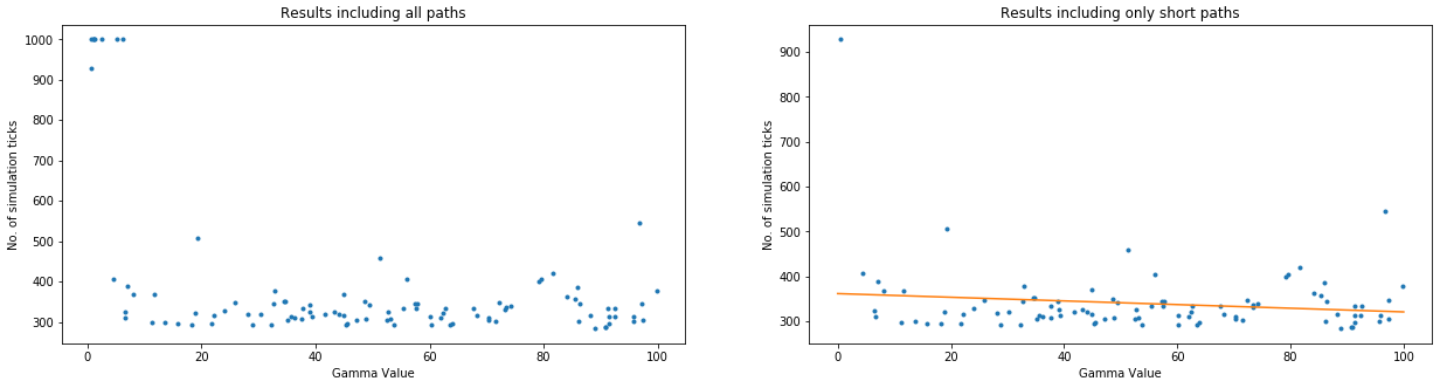


Figure 15: Plot for gamma value against simulation time, for 100 simulations, percentage of success 93.0%

Alpha and Beta Value:

Next, we look to optimise the alpha and beta value, due the implementation of the state transition rule (2) these two values are dependent; first, we run 2 separate sets of simulations each of 100 simulations in which one varies alpha such that $0 < \alpha < 10$; then, another set which varies beta such that $0 < \beta < 10$. ($\alpha = 2$ $\beta = 3$ when not being varied) The results of these simulations, displayed in Fig. 16, show an exponential for alpha as it converges to 300s for α>2 . The alpha plot also shows asymptotic behaviour at α≈ 1, this is as α →0, the phenome level is no longer considered as the transition only relies on β, this thus turns the algorithm to a greedy stochastic algorithm, this removes the positive feedback from pheromones and thus the solution is not converged upon. For the beta simulations we see no trend in the individual data, this could be due to the dependence on alpha value.

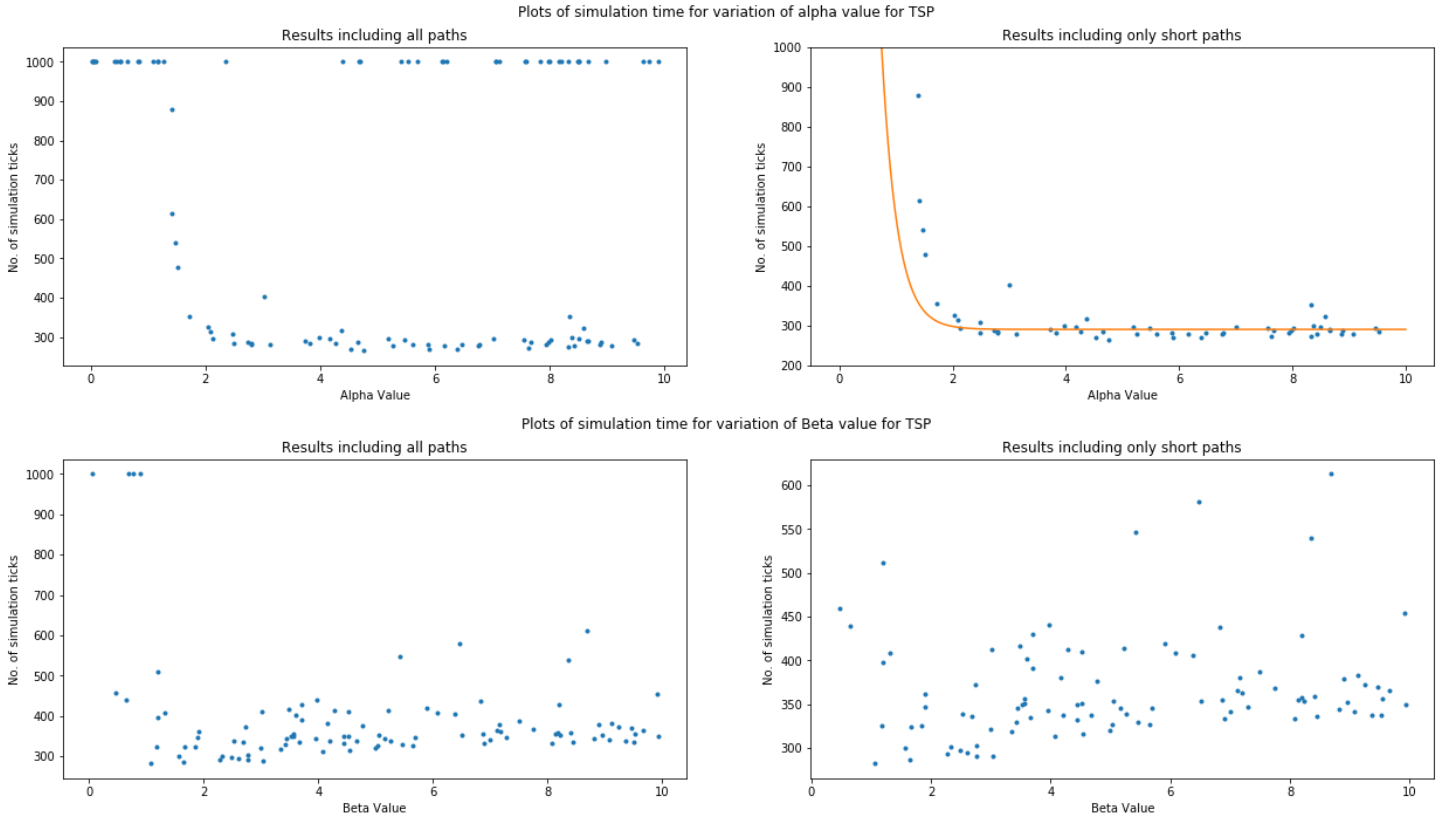


Figure 15: Plot for independent variation of Alpha and Beta value against simulation time, for 100 simulations each

We now run a simulation in which both alpha and beta are varied such that α and β range from 0→10 in incriments of 0.5, we then run a simulation for each of these combinations giving a total of 400 simulations. Plotting these we get a 2D surface, shown in Fig. 16, which is a quadratic surface fitted as $a\alpha^2 + b\beta^2 + c\alpha\beta + d\alpha + e\beta + f$, taking the differential and from some manipulation we get the minma as;

$$\alpha_{\min} = \frac{ce-2bd}{4ab-c^2}$$
$$\beta_{\min} = \frac{cd-2ae}{4ab-c^2} \quad (5)$$

Using the fitting parameters from these simulations we get the minma is corresponds to α=3.30, β=3.12. These values agree with the independent plot and thus we take these as optimal.

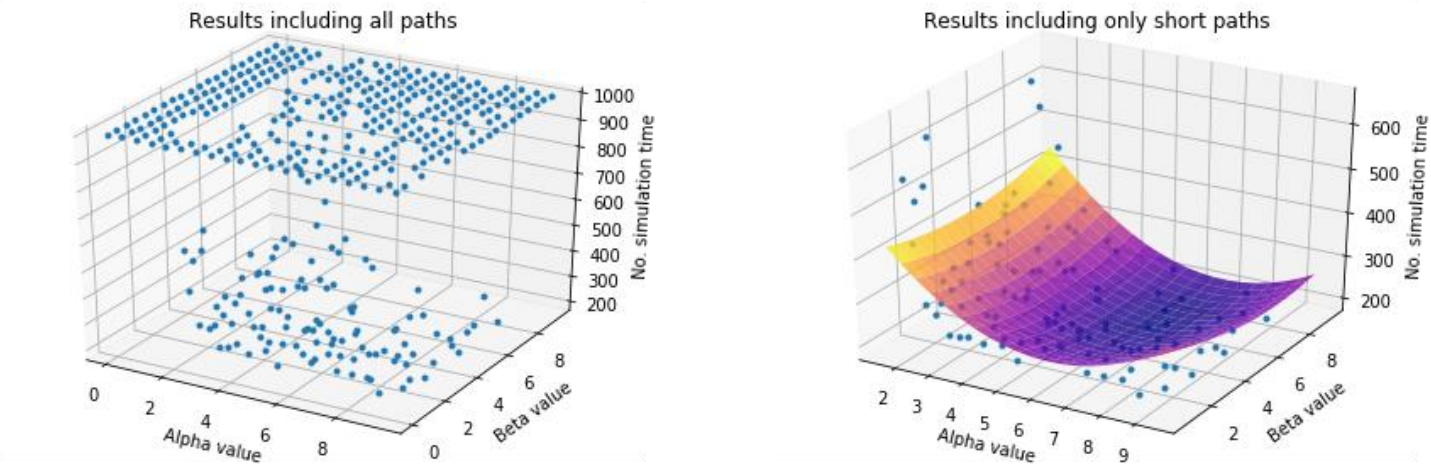


Figure 16: Plot for Alpha and Beta value against simulation time, for 100 simulations each

Optimal Values:
Overall, we get the optimal values for the simulation are such the pheromone diffusion, ρ = 0.9, alpha value, α=7, beta, β= 6 and the gamma value, γ >5. Running 1000 simulations with these values give success percentage 92%.

IV. CONCLUSION

A. *Discussion and Extensions:*

This system has been demonstrated to be a robustness approach to these forms of optimisation, the Ant System has been successfully applied to this optimisation problem and we have discussed its optimisation. Future work can apply this system to other optimization problems such as the asymmetric traveling salesman, the quadratic assignment, and the job-shop scheduling. This system has some limitations, in that the grids rely on physical geometric arrangement; test on larger complex graphs return only partial solutions with loops as shown in Fig.16; and we have not considered a rigorous mathematical model for the system. We have found very some preliminary optimisation, however, there is further variables, such as colony size and graph complexity, that can be explored.

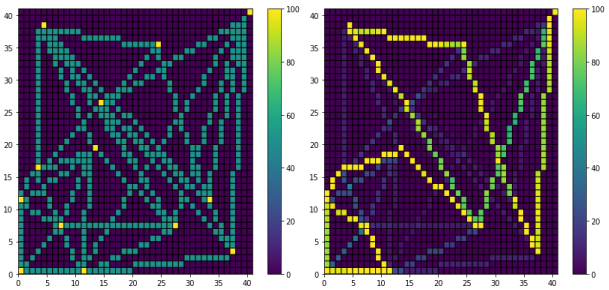


Figure 16: Plot for complex TSP implementation; this gives partial solution with separate loop

The code was implemented with dictionaries as suggested by our project lead, however, future implementations could create a vectorisation of the code for optimisation. The use of time incrementation also slows the code and future implementations could use global pheromone updates based on a continuous time to cut run time. This would be advantages as the draw back of this implementation is the computation speed.

Overall, this lends an insightful overview of this paradigm of algorithmic optimisation, taking a biological system and transposing it to a computation lens. This can be applied to real world optimisation in realms such as logistics in transportation and scheduling optimisation.

REFERENCES

[1] M. Dorigo, V. Maniezzo and A. Colorni, "Ant system: optimization by a colony of cooperating agents," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 26, no. 1, pp. 29-41, Feb. 1996, doi: 10.1109/3477.484436.

[2] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," in IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 53-66, April 1997, doi: 10.1109/4235.585892.

[3] Parpinelli, Rafael & Lopes, Heitor & Freitas, Alex. (2001). An Ant Colony Algorithm for Classification Rule Discovery. 6. 10.4 01 8/978-1-930708-25-9.ch010.