Grid-based ant colony simulation using Ant Colony Optimization

Diego Sanz, Miguel Echeverria | July 2021

1 Introduction

Ants are small creatures that seem quite useless on their own, and indeed they are. However, when they work together, they can achieve incredible things. Unfortunately, they have very limited vision and hearing, so how do they communicate with each other? To understand this, the concept of stigmergy needs to be introduced. Stigmergy refers to the communication of individuals in a group by the modification of their environment. This is precisely what ants do: they leave chemical substances behind called pheromones to communicate or guide other ants to food sources, the nest, or other points of interest (or even steer them away from dangerous places!).

Stigmergy is a technique that has not been pursued that often for group behaviors in game AI, but is one that could give surprisingly good results [3]. In our project, we explore this idea by making a grid-based ant colony foraging behavior simulation, based on mathematical models from the Ant Colony Optimization (ACO) algorithm, which is usually applied for solving graph optimization problems. [1]

In section 2, we briefly explain how the ACO algorithm is applied to the Travelling Salesman Problem, in order to introduce the mathematical models that were used in our simulation. In section 3, we state the scope and goals of the project, whereas in section 4, we talk more in detail about how we came to implement the simulation. In section 5, we show the results obtained from some of the simulations we ran. Finally, section 6 concludes the paper and section 7 contains all the references used during the project.

2 Ant System for the Travelling Salesman Problem

One of the many applications of the Ant Colony Optimization algorithm is the Travelling Salesman Problem. In the following sub-sections we briefly describe how the mathematical model for the algorithm works [1], using a simplified version of an approach to solving TSP that we came across in [2].

2.1 Problem statement

Given a set of cities, represented as vertices in a weighted graph, we want to find the shortest Hamiltonian cycle. A Hamiltonian cycle is a closed loop where each node in the graph is visited exactly once. This can be imagined as a salesman that needs to go through a set of cities exactly once, and end up in the starting city, minimizing the time it takes, hence the name of the problem.

2.2 Algorithm pseudo-code

The algorithm will run for a configurable number of iterations. On each of these iterations, a set of ants will be released from the starting node (the number of ants can also be configured), and these ants will traverse the graph for a configurable set of turns, or until all of them have completed the tour. On each turn, each ant will decide which adjacent node to traverse to, based on probability. The probability of choosing each edge/vertex is computed using Equation 1. After the current set of ants have finished, each edge that was traversed by the ants that found a solution is updated with a new pheromone value (that is computed using Equation 2, and the most optimal solution found so far is saved/updated.

Listing 1 Pseudo code for TSP using the Ant System algorithm [2].

```
For each group of ants
    For each ant in the current set of ants (until turns end)
        If ant has completed tour
            Stop exploring with this ant
        For each adjacent feasible vertex
            Compute edge probability (Equation 1)
        Decide edge to traverse based on the probabilities
    Update the pheromones on the edges traversed (Equation 2)
    Update most optimal path found so far
```

2.3 Edge probability

The probability of selecting an edge that goes from vertex i to j, depends on the cost of traversing that edge, and the amount of pheromones in it, and it is computed as follows.

Equation 1

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{e_{il}} (\tau_{il}^{\alpha} \cdot \eta_{il}^{\beta})} & if\ edge\ ij\ is\ feasible \\ 0 & otherwise \end{cases}$$

Where the e_{il} represents each of the feasible edges that start from the vertex i, τ_{ij} represents the current pheromone value in the edge that goes from vertex i to j (same for edge from i to l), and η_{ij} is a value inversely proportional to the cost of edge from i to j (same for edge from i to l). Finally, α and β are configurable parameters to adjust whether more importance should be given to the pheromones on that edge, or the cost of traversing it. The value of η_{ij} is given by Equation 2.

Equation 2

$$\eta_{ij} = \frac{1}{d_{ij}}$$

Where d_{ij} is the cost of traversing the edge from vertex i to j.

2.4 Pheromone updating and evaporation

After each set of ants has finished. The pheromone values of each of the edges need to be updated, so that the next set of ants can make better decisions. Moreover, we need to simulate evaporation of pheromones over time, so that paths that are not refreshed are not taken again. All this is conveyed in Equation 3.

Equation 3

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

Where ρ is the evaporation rate, a configurable parameter in the range [0, 1] (no evaporation to max evaporation) that controls how much of the existing pheromone is kept. Therefore, τ_{ij} represents the current pheromone amount in the edge from i to j. Finally, on the right side of the equation, the new pheromone amount is added: k and m represent the current ant and the total number of ants respectively, and $\Delta\tau_{ij}{}^{k}$ represents the amount of pheromone laid on the edge ij by ant k. This amount is given by Equation 4.

Equation 4

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \mathit{if\ ant\ k\ traversed\ edge\ ij} \\ 0 & \mathit{otherwise} \end{cases}$$

Where L_k is the total cost of the path found by ant k. This way, ants that found better solutions (less cost) will add more amount of pheromone to the edges they traversed, guiding other ants to similar solutions.

3 Project objectives

The aim of this project is not to create the most accurate or realistic ant simulation. Instead, we are more interested in developing an interesting demo that showcases the concepts of the ACO algorithm, how ants explore the terrain forming more or less optimal paths between the food sources and the nest, and how they adapt to form new and more optimal paths when the terrain changes [5]. Our work was inspired by the videos [4] and [5].

3.1 Ant colony foraging simulation

Like previously stated, our goal is to create an ant foraging simulation. The user will be able to place the nest for the ants to spawn at the beginning (they will start with random directions and speeds), and these ants will explore the space in search of food. When they find food, they will bring it back to the nest, following a certain path. This path between the nest and the food source will be formed thanks to the pheromones laid by the ants.

3.2 Dynamic food and obstacle dropping

The user will also have two terrain editing modes: food and wall editing. When the food editing mode is active, users will be able to drop food in the terrain by holding left mouse button. When the wall editing mode is active, however, users will have two options. They will be able to either place walls with left mouse button or delete already existing walls with right mouse button. This dynamic wall placing and removing will sometimes cause ants to form new paths [5].

3.3 Grid-based approach

The ACO algorithm is mainly used for solving graph problems. However, on our case, we will be using a uniform grid to represent the terrain, so our implementation will be a little bit different, as we will explain in section 4.

3.4 Steering behaviors

As previously mentioned, the ACO algorithm is used for graph problems. Nevertheless, we want to implement an actual ant simulation, so the ants need to have a fluid movement with some small random steering, so traversing a graph of waypoints did not seem like a great idea. Instead, we have opted to use steering behaviors for the movement of the ants.

4 Implementation details

In this section we will mention how we came about implementing some of the features in our simulation.

4.1 Floor and uniform grid parameters

For our simulation, we used a scaled quad as the floor. The scale of this quad on the x and z axes was made a constant, and it was calculated so that it had a 16:9 aspect ratio. In order to represent the food, the pheromones, the walls etc, we created different uniform grids that were overlayed on top of this quad.

For each of these grids, we added an option to change the number of columns in it, and as a result the number of rows was recalculated automatically, so that each cell had the same width and height. For each different grid, the necessary cells were drawn with the corresponding cell size as colored cubes. Increasing the number of cells in each grid made the cell size smaller, therefore giving higher resolution. However, having more cells means having more data to process per grid, and more cubes to draw, so for performance reasons this value was not made very big. Also, having extremely small cells could lead to problems if the ants were bigger, taking up the space of multiple cells.

4.2 Food representation

The food was represented with one of the already mentioned uniform grids. Each of the cells in this uniform grid has an integer associated to it. This number represents the amount of food units that are in that particular cell. If the number of food units in a cell is bigger than one, then a green cube representing food is drawn in that cell (with the scale of a food grid cell). When an ant sees a food cell (refer to section 4.4 for ant sensing), it will steer towards the closest food cell, and pick up 1 unit of the food in that cell, thus reducing the number by one.

The reason all the cells have this number, instead of simply a Boolean indicating whether there is food or not, is because we found that with increasing number of ants, the grid cells had to be made extremely small for the same food sources to last a decent amount of time. And as we previously explained, to achieve this, the number of grid cells needs to be greatly increased, which is not the best for performance.

4.3 Pheromone representation

Regarding the pheromones, we also used a uniform grid. However, we used two different pheromone grids, one for the food pheromones (displayed in green in the simulation), and another one for the nest pheromones (displayed in purple/blue in the simulation). Both of these have the same cell size, but they have smaller cells than the food grid (therefore, more cells).

Each of the cells contains a pheromone intensity value as a float. This value can be in the range [0, MAX_INTENSITY], where MAX_INTENSITY is a constant that we specified beforehand. When drawing the pheromones, those that have a smaller intensity than a small epsilon threshold are discarded, and the rest are drawn with a color C_i that is computed as follows.

Equation 5

$$C_i = F_i \cdot C_f + N_i \cdot C_n$$

Where the subindex i represents the cell of the pheromone grid at index i, F_i and N_i represent the normalized intensity of food pheromones and nest pheromones respectively (at cell i), and C_f and C_n represent the colors used for food and nest pheromone representation, respectively. To obtain the normalized intensities, it is just a matter of dividing the intensity by MAX_INTENSITY.

The amount of pheromone that is dropped on a cell by an ant gets smaller over time, unless said ant goes back to the nest/food source depending what type of pheromone it is dropping. This way, we ensure there is more concentration of pheromones near the interest point. which allows ants to always follow paths that end up leading to them, instead of taking random detours. It is worth noting that we also added a timer to control the rate at which ants drop pheromones.

Finally, we also simulated evaporation of pheromones over time, so we implemented that by applying the left side of Equation 3 to the intensities of each pheromone cell, at the beginning of each frame update.

4.4 Ant movement

As outlined in section 3.4, we chose to represent ant movement by using steering behaviors. At first, when the nest is placed in the map, ants start searching the terrain looking for food with a constant speed, that is different by a random amount for each ant. The initial direction in which they start exploring is also random.

When ants first start exploring, they are in the lookout for either pheromones or for a food source. If they see food, they will steer towards it and grab one unit of food. On the other hand, if they don’t see food, they will look for pheromones. But if they don’t see any pheromones either, they will simply move around the environment using the wandering steering behavior.

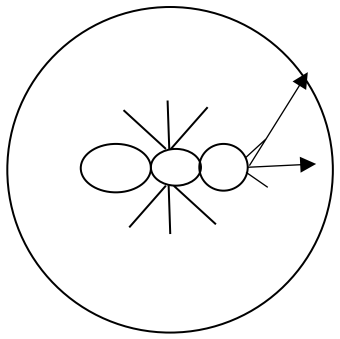


Figure 1 An ant with its forward vector, wandering circle, and desired vector.

To implement wandering, we defined a circle of a configurable radius, at a configurable distance from the ant in the forward direction. In our case, we set the distance to 0 so that ants can rapidly turn in the opposite direction. Every certain amount of time, we took a random angle, and set that as our target angle, and until the next random angle is taken, the current angle is interpolated from current to target. The current angle then gives us a point in the circumference that we use to get the desired vector. Figure 1 illustrates this process.

However, if they do see food pheromones, they will make a random decision of which direction to steer towards between 3 possible directions. The probability of each direction will consider the amount of pheromone in the area. This is explained in more detail in section 4.4. There will also be a small chance of ants just continuing to wander randomly. The reason for this is explained in section 4.5.

Once an ant has found food, it will follow the same exact behavior, but this time it will be interested in the nest, and in nest pheromones.

4.5 Ants’ sensing capabilities

Ants sense the environment around them through a Field Of View mechanism. The FOV we implemented is defined by an angle, and a radius. We set the angle to 180 degrees, but since ants are known to have limited vision, the radius was set to a very small value. Whenever we want to decide the ‘state’ of an ant (wandering, going to food/nest, or following pheromones), we simply establish which cells of the grid belong to the FOV of that ant, and we then query the grids with those cells to find the information we are looking for.

In the case of pheromone following, however, the process is a little bit different. We need to divide the FOV in three distinct regions of 60 degrees each (shown in Figure 2) and calculate the probability of the ant following each region. This probability is calculated using Equation 1 but ignoring the distance term. Therefore, τ_{ij} now becomes the pheromone concentration on the current region, which is obtained by simply adding up all the pheromone intensities of each cell of the FOV, that belongs to that region.

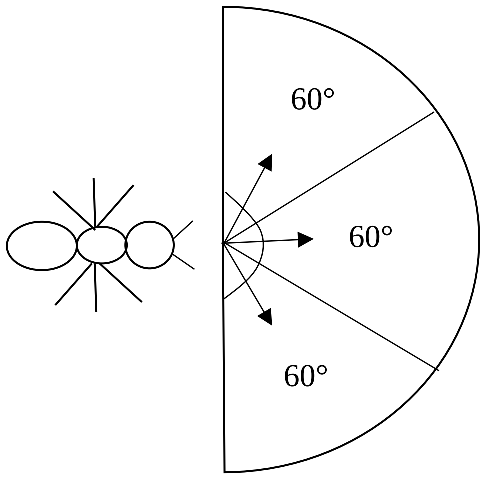


Figure 2 An ant's FOV, divided into the three 60° regions.

4.6 Path optimization

Apart from the probability of each of these three regions, there always needs to be a small probability of ants just wandering off randomly, instead of following one of the three regions. This is so that once a path between food and nest is established, a few ants can deviate slightly from the path, and over time find a more optimal path that will end up becoming the one that almost all ants follow [5].

4.7 Wall representation

Walls, like food and pheromones, have their own uniform grid. However, the ones dedicated to walls has cells of much bigger size (therefore less total cells), as it makes it easier for the user to place walls and cut off certain areas.

4.8 Obstacle avoidance

Ants must avoid two things, the map's edges and the walls placed by the user. Avoiding the edges is a straightforward implementation. Whenever an ant gets out of the map, its forward vector gets reflected by the normal of the map's edge.

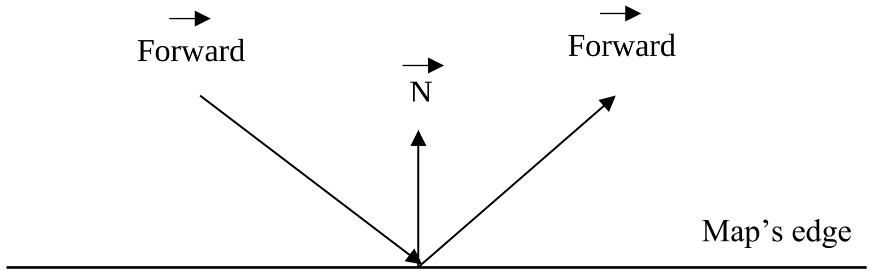

$\overrightarrow{\text{Forward}}$
$\overrightarrow{N}$
$\overrightarrow{\text{Forward}}$
Map's edge

Figure 3 Ant's forward vector gets reflected when going out of bounds.

In the case of walls, it is slightly more complicated. Ants will sense the cell (wall grid) that is in front of them. If there is a wall, the ant will avoid it by reflecting its forward vector. If there is no wall, the ant will still need to check for diagonal walls. If there are none, the ant will be able to continue its path.

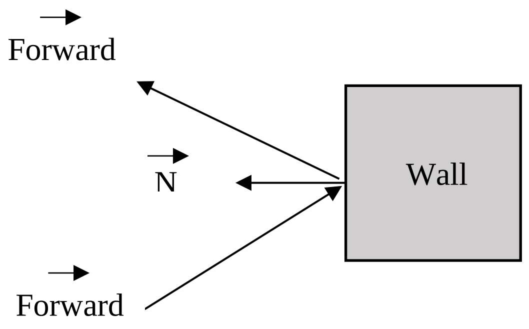
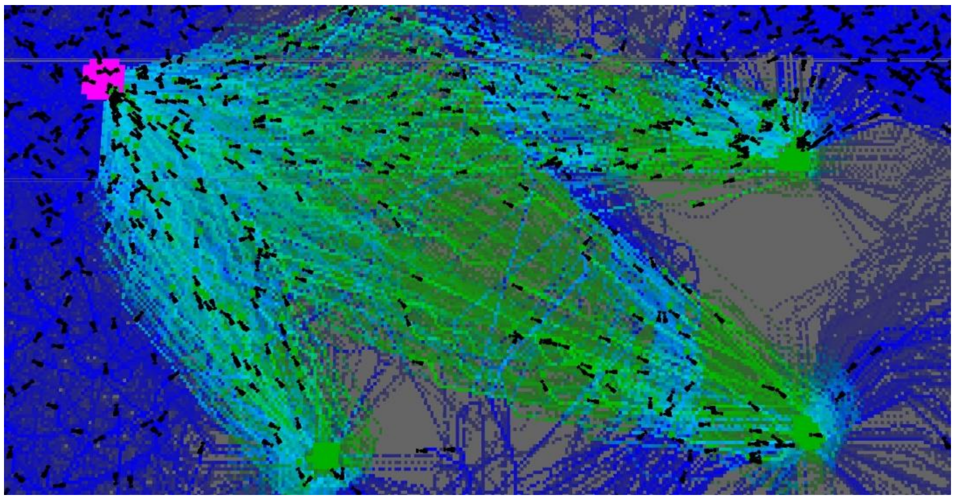

$\overrightarrow{\text{Forward}}$
$\overrightarrow{N}$
Wall
$\overrightarrow{\text{Forward}}$

Figure 4 Ant's forward vector gets reflected when detecting a wall

5 Results

After a lot of parameter tweaking, and bug hunting, we concluded that it was best to remove some of the probabilities. This is because there were too many parameters to tweak, and it was not working completely as it should. Therefore, ants now always follow the region with more concentration of pheromones, which is not ideal, as they will tend to miss paths that may end up being more optimal. However, in our simulations, this didn't happen that often, and paths also tended to optimize themselves when for example, removing walls.



6 Conclusion

Stigmergy is a very interesting form of inter-agent communication, and we encourage the reader to find new uses for this type of behaviors, as it has not been explored that often in videogames. Possible improvements to our simulation, apart from smoothing out the movement of the ants and adding some random turns when following the pheromones, include having a more sophisticated sensing model, or implementing an instanced rendering system, to improve performance and be able to have much bigger simulations.

7 References

[1] Dorigo, Marco & Birattari, Mauro & Stützle, Thomas. (2006). Ant Colony Optimization. Computational Intelligence Magazine, IEEE. 1. 28-39. 10.1109/MCI.2006.329691. https://www.researchgate.net/publication/308953674_Ant_Colony_Optimization

[2] Fernando, T.G.I. & Kalganova, Tatiana & Fernando, W.A.C.. (2006). A Grid-based Ant Colony Algorithm for Automatic 3D Hose Routing. 48 - 55. 10.1109/CEC.2006.1688289. https://www.researchgate.net/publication/224645695_A_Grid-based_Ant_Colony_Algorithm_for_Automatic_3D_Hose_Routing

[3] Courtney, Joshua, "Using Ant Colonization Optimization to Control Difficulty in Video Game AI." (2010). Undergraduate Honors Theses. Paper 147. https://dc.etsu.edu/honors/147

[4] Sebastian Lague (Youtube, 2021), "Ants and slime simulation" https://www.youtube.com/watch?v=X-iSQQgOd1A&t=0s

[5] Pezza's Work (Youtube, 2021), "C++ Ants Simulation 2, Path optimization" https://www.youtube.com/watch?v=emRXBr5JvoY&t=0s