Active Walkers

Jacob Toller

Weeks 7-12, Semester 2, 2023

1 Abstract

Active Walkers are a representation of simple robots which explore the area around them and complete a provided objective. The Walkers navigate by depositing markers which can influence other walkers to prefer a certain direction in their random decision making; as they have no other method of communication/localisation. Walkers move by sampling the area in-front of them for markers, randomly choosing a move biased towards the strongest marker.

This is useful as the emergent behaviour of a group of Walkers can be used in robotics to replace the role of processing-intensive AI models and expensive sensors such as LIDAR; with advantages in cost, reliability and size.

We discovered that the efficiency of the walkers to complete their task of discovering and retrieving resources to their base was highest when they acted with less randomness, almost always walking towards the strongest marker.

Our model also has applications in simulating the behaviours of ants that use pheromones to navigate; and were able to replicate real life phenomena which occurs in real life, namely Ant Mills.

Contents

2 Introduction

The task for this project was to develop a model to simulate 'active walkers'; which are able to roam a physical space in order to discover and return resources back to their base location. These walkers should only be able to indirectly communicate through markers that are left behind as they explore, with no centralised control/thought.

The applications of this is that it allows for complex behaviour to emerge from simple robots with low processing power; instead of more high-level approaches that might make use of AI and computer vision, or approaches which utilise a centralised control center to manage all of the robots in the cluster. This offers an advantage in the cost of the robots, energy-efficiency[1] and reliability.

We will start by declaring the modelling methodology that we used; how we modelled the active walkers; and then explore their behaviour and how it can be optimised for reliability and efficiency.

3 Previous Literature

3.1 Agent-Based Modelling

Agent-Based Modelling, as described in (Crooks and Heppenstall 2011); is a type of discrete model which seeks to model a scenario based on an environment, and the individuals that exist within the environment. An agent has the following important properties:

[1]My thesis project is a centralised control system for a robot utilising computer-vision. It drains my laptop battery in half an hour.

- Autonomy; where the decisions made by the agent are influenced only by its internal states (such as the direction it is facing, or whether it is holding anything) and its immediate surroundings (this can include interactions with other Agents)
- Mobility; where the agent can roam the space that it exists in, in order to achieve its goals and execute its rule-set
- Interactive; where the agent can influence its immediate surroundings and the environment

This type of approach lends itself quite well to modelling different social and biological processes, as it allows the modeller to explicitly define its components. This is in contrast to Cellular Automata, where the future state of the entire system is derived from a globally applied rule-set; while it would be possible to model our scenario using a Cellular Automata; it would be a lot more complex/time-consuming to design and implement, as well as less intuitive to discuss, present and interpret results from.

An agent in a model is defined by two things:

- Its current state; such as its current position, which way its facing etc.
- Its ruleset, which it applies by evaluating its current state and the state of its surrounding neighbourhood to transfer into a new state, such as a new position.

3.2 Expected Behaviour

Within the brief of our project, we were provided with Active Random Walkers Simulate Trunk Trail Formation by Ants (Schweitzer, Lao, and Family 1997)

This source was incredibly useful to us as it acted as a starting off point for our model; and gave us intuition as to what to expect. This paper starts by describing and observing the behaviours of ants, and the trails that they make; before proceeding to create a model to simulate the trails of the ants.

Their model consists of active walkers which take a biased random walk with a tendency to continue moving forwards and away from the starting nest, as well as markers which can be deposited to influence the movement decisions of the walkers that encounter them at a later time. However, these markers will deplete over time if not replenished by another walker.

The first version of their model has a single type of marker, where a walker is expected to depart from a central nest, collect food, and follow the trail that it has just left in order to locate its way home.

The second iteration of their model utilises two different types of markers. In short, a 'scout' will initially leave the nest depositing marker A, and sensitive to marker A; but will deposit a different type of marker, marker B, once it discovers food and then follows its trail of marker A on the way back. It will then recruit 'carriers'[2] which are sensitive to marker B, in order to locate the recently discovered food source. On their way to the food source, they will replenish the trail of marker A. Once they reach the source, they will become sensitive to A and deposit B to find their way home. This results in two trails being established, one of type A which signifies "follow me to find your way home"; while the other, B, says "go this way to find food".

Our model differs from this by removing the walkers ability to track roughly where they are relative to their starting nest; and places their ability to locate entirely on following markers. Additionally, we simplified the walkers into two distinct job roles, which define the markers they deposit and are sensitive to: -

- All walkers leaving the nest start as 'Scouts', which are sensitive to marker B while depositing marker A. They will transform into the 'Carrier' role once they discover food:

[2]The original source here labels them as 'recruits', but carriers is used to be consistent with our labelling later.

- 'Carrier' walkers are those that have food and need to return it to the nest. They are sensitive to marker A and will deposit marker B on their way home. Once dropping their payload off at the nest, they will revert to the role of 'Scout'.

4 Our Model

4.1 Environment

We define the area that our walkers can roam around in as a discrete grid of cells, with dimensions of our choosing. For simplicity, the cells in our grid are modelled such that they can accept any number of agents, markers or other objects inside them with no collision; and simply assume that the physical space the cell represents is large enough to contain them all.

The boundary conditions that we chose to use were that of a solid wall, where a walker up against the boundary would have to choose from its remaining valid options or do a 180° turn on the spot if no valid options existed.

Passage of time in our model is modelled as discrete turns/timesteps; where for each turn, every walker will make its move at the same time.

$$\overrightarrow{x} = (x, y) \in [0, N] \times [0, M] \tag{1}$$

$$t \in \mathbb{Z}^+ = \{0, 1, 2, ...\} \tag{2}$$

Where N,M are the width and height of the grid.

4.2 Walker Ruleset & Decision-Making

Now we've established what an Agent-Based model is; lets now introduce our agent.

To define our walkers current state, we will keep track of three things:

- Its current position on the 2D grid
- Its current facing direction; as one of the four cardinal directions and four diagonal directions
- What its current job role is; 'scout' or 'carrier'

Similarly to the model discussed in 3.2, we want our walker to move in a way that it prefers to go forwards. We can do this by allowing the walker to choose a move from the three cells in front of it, relative to its current position and direction. See diagram 1. This move will define its position and facing direction in the future state.

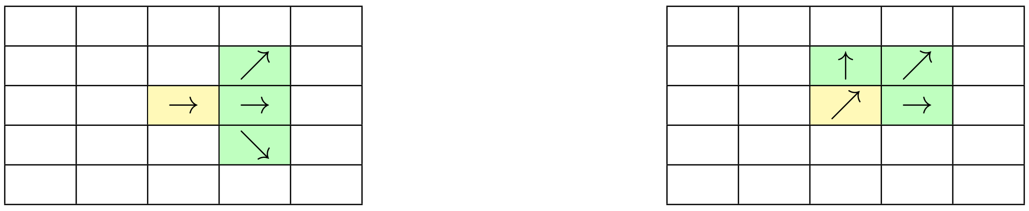


Figure 1: Diagram showing valid walker moves relative to its current position and facing. Similarly for the other six possible directions.

The choice of move that the walker will make is random, and controlled by a Probability Density Function that we will explicitly define in 4.4; that takes into account the concentration of the marker type that it is sensitive to at that cell. Once the walker has made its move, it will lay down a new marker corresponding to its job role.

4.3 Marker Concentration Sub-model

In order for the walker to be able to compare the cells in-front of it; we need to be able to evaluate the concentration/intensity from the relevant markers at a given cell.

For this, we settled on a system where the concentration of a cell at position $\vec{x} = (x, y)$ is calculated as the sum of the concentrations of each individual marker; and each markers concentration is proportional to some inverse-power of the distance[3] from the marker:

$$C(\vec{x}) = \sum_{i=1}^{n} = c_i(\vec{x}) \tag{3}$$

$$c_i(\vec{x}) = \frac{\alpha}{(||\vec{x} - \vec{p}_i|| + 1)^{\beta}} \tag{4}$$

The effect of a single marker in isolation is shown in Figure 2, which is equivalent to (4). Figure 3 shows how the concentration value for each cell is affected by adding a second marker. Note how the cells which contain each marker are now able to reach a higher value than they would otherwise be able to reach on their own.

0.17	0.25	0.17	0.10	0.06	0.04
0.25	1.00	0.25	0.11	0.06	0.04
0.17	0.25	0.17	0.10	0.06	0.04
0.10	0.11	0.10	0.07	0.05	0.03
0.06	0.06	0.06	0.05	0.04	0.03
0.04	0.04	0.04	0.03	0.03	0.02

Figure 2: Concentration grid values for a single marker placed at the blue highlighted cell, with parameters $\alpha = 1$, $\beta = 2$

0.19	0.27	0.20	0.12	0.09	0.06
0.27	1.03	0.28	0.15	0.10	0.08
0.20	0.29	0.22	0.15	0.12	0.10
0.13	0.16	0.16	0.16	0.16	0.13
0.10	0.12	0.15	0.22	0.29	0.20
0.08	0.10	0.15	0.28	1.03	0.27

Figure 3: Combined concentration values of two markers (blue highlighted cells), with parameters $\alpha = 1$, $\beta = 2$

[3]We add 1 to the distance in order to prevent the concentration from being undefined at the location of the marker itself, and to cap the value here to α

Where $\vec{p}_i$ is the position of the ith marker; α is a linear strength coefficient; and β as the falloff parameter. For example, a value of $\beta = 2$ would mean that the intensity would follow the inverse-square law.

In its current state, once deposited the markers will persist indefinitely. This isn't good as we want to have them gradually get weaker with time so that only the trails which are constantly traversed are refreshed. We can model this by having α as a function of t:

$$\alpha_i(t) = \begin{cases} 0 & \text{if } t < t_i \\ 1 - \frac{t-t_i}{T} & \text{if } t_i \leq t < t_i + T \\ 0 & \text{if } t \geq t_i + T \end{cases} \tag{5}$$

This function varies the linear strength of the ith marker; where $\alpha_i = 1$ when it is deposited at t_i, and decreases linearly to zero over a duration of T.

This is calculated on a per-type basis, so we keep track of a list of all the markers of type A, and all the markers of type B; and calculate their concentrations separately (i.e. $C_A(\vec{x})$ and $C_B(\vec{x})$). The marker concentration is then utilised in the aforementioned Probability Density Function:

4.4 Probability Density Function for Choosing a Move

What we want to achieve in this section is to define a Probability Density Function for a random variable X, that takes one of three options as inputs; and spit out the probability that the walker will choose that option.

We do this by first calculating a relative weight for each option as so:

$$w(i) = C(\vec{x}_i)^\lambda \tag{6}$$

With $i \in \{\textit{left}, \textit{front}, \textit{right}\}$ and $\vec{x}_i$ as the position of the referenced cell. We then normalise this as so:

$$P(X = i) = \hat{w}(i) = \frac{w(i)}{w(left) + w(front) + w(right)} \tag{7}$$

There are also a few extreme cases that we must consider:

- In the case where the walker is up against a wall, we consider any move that would place the walker outside the boundary of the grid as invalid; and force the value of $w(i)$ to 0. (Choose from remaining options)
- If all of the three options are invalid, the walker should be forced to turn around on the spot without moving or placing a marker.

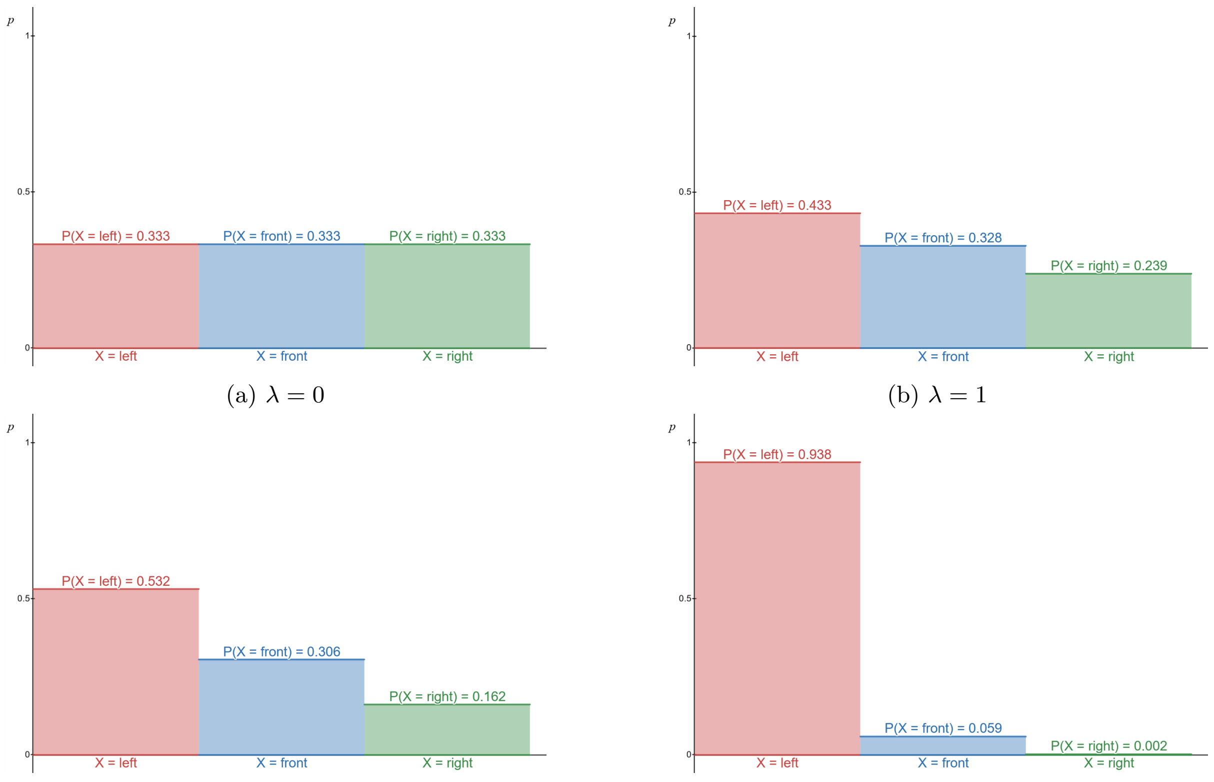
What this gives us is a PDF that we can control how biased towards higher marker concentrations it is, by varying the parameter λ.

As an example, we will place a walker into the space such that it is in the configuration found in Figure 4; with $C(\vec{x}_{\textit{left}}) = 0.29$, $C(\vec{x}_{\textit{front}}) = 0.22$, $C(\vec{x}_{\textit{right}}) = 0.16$.

If we were to use $\lambda = 0$, there would be an equal probability for each of the options, as $w(i) = 1 \ \forall i \in \{\textit{left}, \textit{front}, \textit{right}\}$. However, with larger values of λ, the probability that the walker chooses the largest concentration increases. See Figure 5.

0.19	0.27	0.20	0.12	0.09
0.27	1.03	0.28	0.15	0.10
0.20	0.29	0.22	0.15	0.12
0.13	0.16	0.16	0.16	0.16
0.10	0.12	0.15	0.22	0.29
0.08	0.10	0.15	0.28	1.03

Figure 4: Diagram showing what options (green) a walker placed in the yellow highlighted cell would have, when facing North-East



(a) $\lambda = 0$

(b) $\lambda = 1$

(c) $\lambda = 2$

(d) $\lambda = 10$

Figure 5: Comparison of λ values and how it affects the probability density function. $C\left(\overrightarrow{x}_{left}\right) = 0.29$, $C\left(\overrightarrow{x}_{front}\right) = 0.22$, $C\left(\overrightarrow{x}_{right}\right) = 0.16$, from Figure 4.

4.5 Objectives & Performance Metrics

The objective of our walkers will be to transport all of the resources (food etc.) from the resource piles in the environment, to a centralised base. The resource piles contain multiple units of food that cannot be carried by a single walker in a single trip, so the walkers will benefit from using their markers to signal to the other walkers to help move the resource. When a scout walker touches a resource pile (given that it doesn't already have food), it will collect a unit of the resource; turn 180°; and swap to the carrier job role. Similarly when a carrier walker reaches their nest.

This allows us to quantify the performance of the walker system and the parameters we supply them; as we can measure the total amount of resource at the base, how many resources are currently in-transit, and how many units of resource are still located in the environment. We can then plot these values over time to see how well the walkers are performing.

4.6 Implementation

Agent-based modelling is well suited to Object Oriented Programming approaches, where the agent itself is defined by a class, and its rule-set is executed by a method of that class. The language that we used for our implementation was Python 3

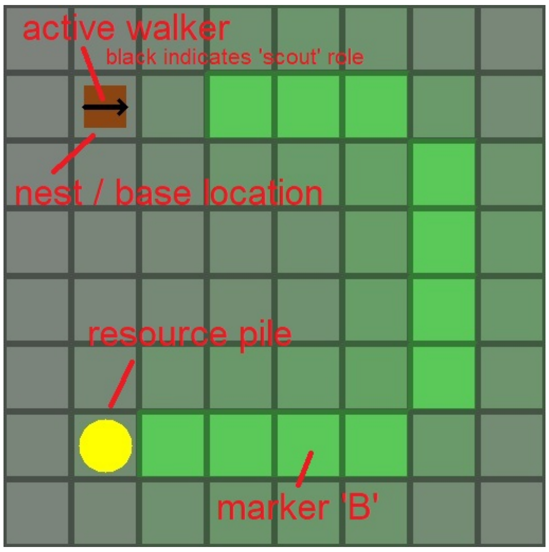
I will keep the exact details of the implementation to the Appendix; and keep the remainder of this section to a rough process list of what the program does every time it processes a timestep, to calculate the future state of the system:

1. For each of the marker types (namely A and B); generate the concentration value for each possible position $C(\overrightarrow{x})$. We do this for every cell, not just the ones near the walkers as these values are used for the visual renderer.
2. All the objective pieces (nest, resource piles) check if there are any walkers nearby that they can take/give resources to; allowing the scouts to collect from the piles, and carriers to deposit at the nest.
3. Make all the Walkers pick their next move:
   (a) Determine what its job role should be. If it is carrying resources it will update to a 'carrier', otherwise a 'scout'.
   (b) Read the marker concentration corresponding to its job role, and construct the PDF from (7)
   (c) If it has at least 1 valid choice, pick a new position and facing direction from the PDF; if no valid choices, turn around.
   (d) If it was able to make a move, create a new marker (type corresponding to job role) and place it in the position it used to be in. Note that this marker will not have any effect until the next timestep.
4. Render the state of the system for visualisation and export the telemetry for plotting later.

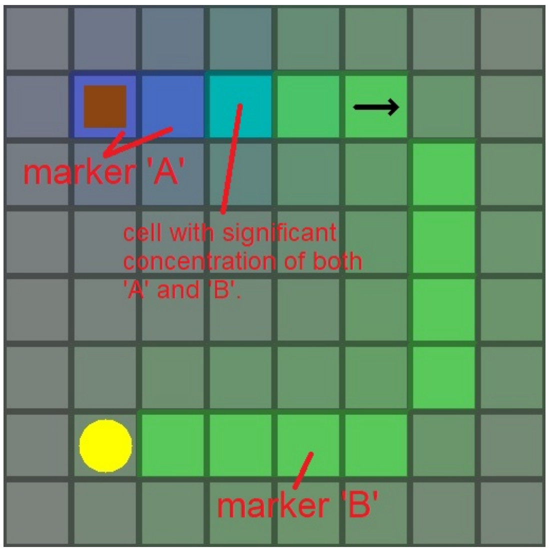
5 Observation & Experimentation

5.1 Basic Behaviour

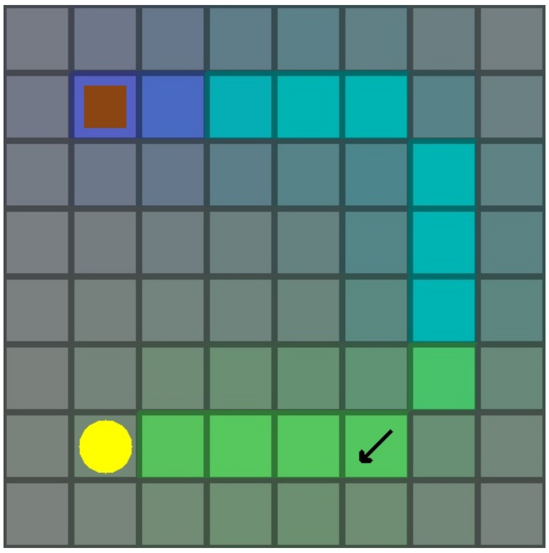
We will now demonstrate typical behaviour of how a single walker operates, which should tie up everything discussed in the previous section.



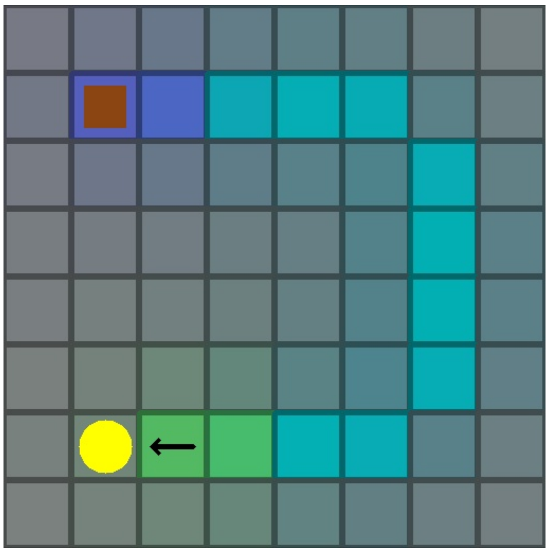
(a) $t = 0$



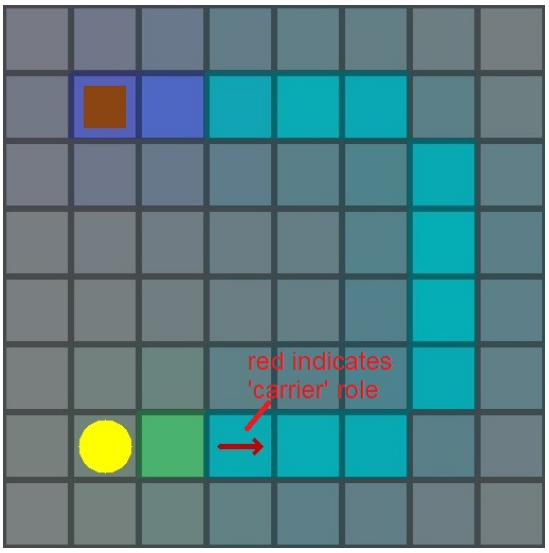
(b) $t = 4$



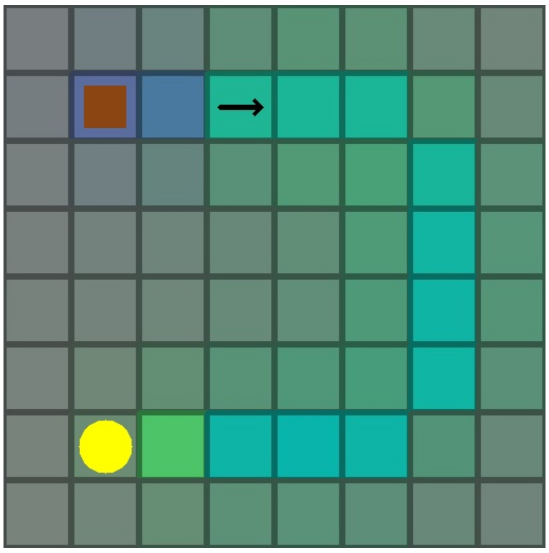
(c) $t = 9$



(d) $t = 12$



(e) $t = 13$



(f) $t = 24$

Figure 6: Simple progression of the model as t increases. Initialised with a single walker and resource pile. $\lambda = 3$, $\beta = 3$, $T = 40$. Initial trail at $t = 0$ placed manually for demonstration purposes.

Figure 6 showcases how the system could progress with time, as instantiated in sub-figure 6a. It shows how a walker with the role of 'scout' will follow a trail of marker 'B' (green highlighted cells) in order to find resources. Once it has collected resources ($t = 13$, sub-fig 6e), it will start refreshing the trail of marker 'B' by following the trail of marker 'A' (blue highlighted cells) that it deposited on its way out. Once it reaches its base, it will revert back the scout role to grab more resource.

Due to the relatively high λ value, it does not deviate much from the trail. This is something that we will look into later.

5.2 It's Not a Bug it's a Feature!

One of the types of phenomena that we would encounter in our simulations was one where all of the walkers in the simulation would get stuck in a state where they could not transport any of the resources to the base, and where the quantity of resources in transit by the walkers stays constant; shown in 7:

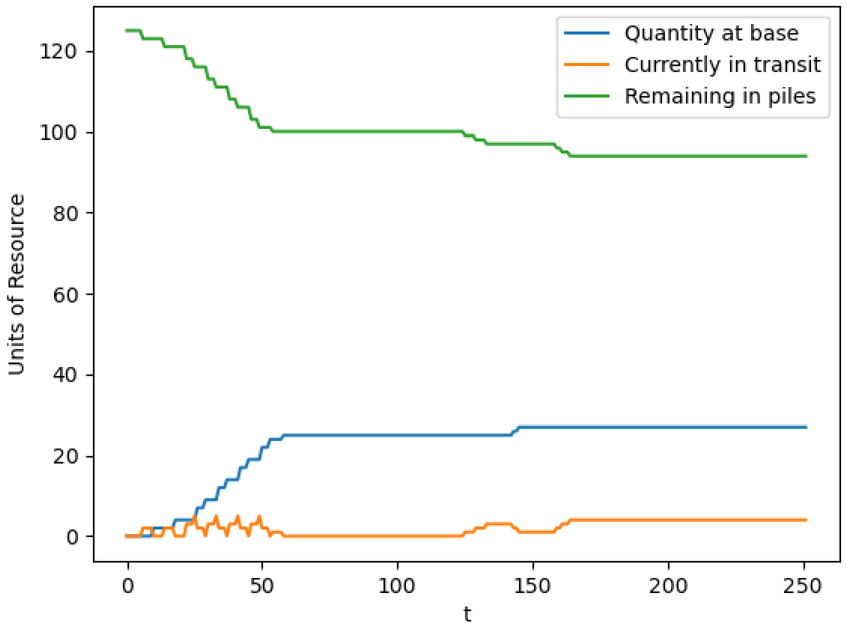
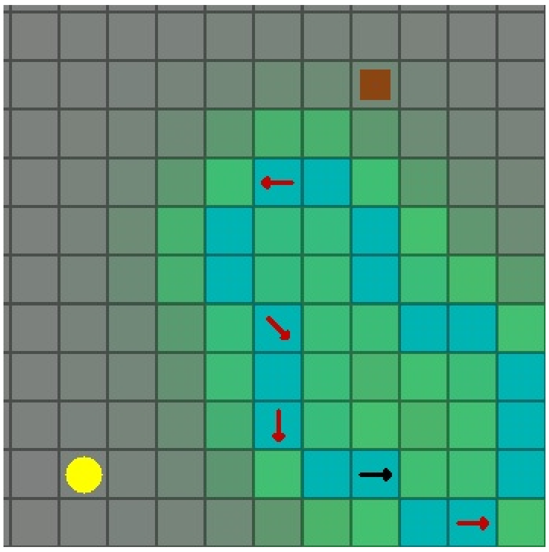


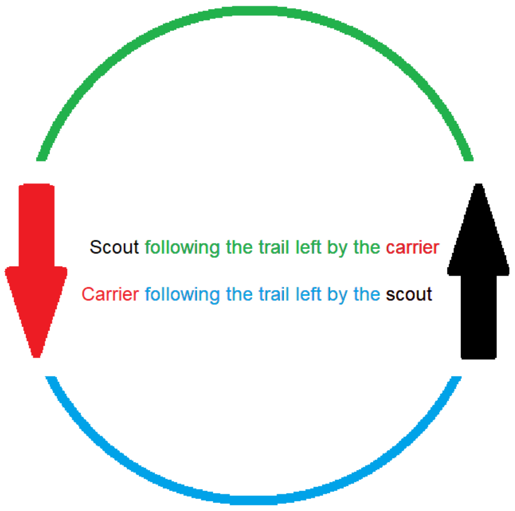
Figure 7: Walkers that are unable to complete their task of moving all the resources to base, due to getting stuck.

During normal behaviour, the orange plot is either constant at zero, or rapidly oscillating. This corresponds to the walkers either exploring for food, or travelling back and forth between the base and a resource location. However, taking a look at the system (Figure 8) quickly reveals what the issue is:

These walkers are stuck following the trails of one another, where the criteria for forming this phenomena are simply that a collection of walkers are following a set of trails, where the trails are being continually refreshed by the affected walkers, reinforcing the problem. This is not just a byproduct of our model, but can also be found in similar models (Erhard, Franco, and Reis 2020) and even in nature (Schneirla et al. 1944); where it has been labelled as an 'ant mill'.



(a) Rendering of system from Fig. 7 at $t = 251$, cropped to relevant area.


Scout following the trail left by the carrier
Carrier following the trail left by the scout

(b) Explanation diagram

Figure 8: Walkers stuck in an 'ant mill', where they are following the trails of walkers which in turn are following their trails; forming an inescapable loop.

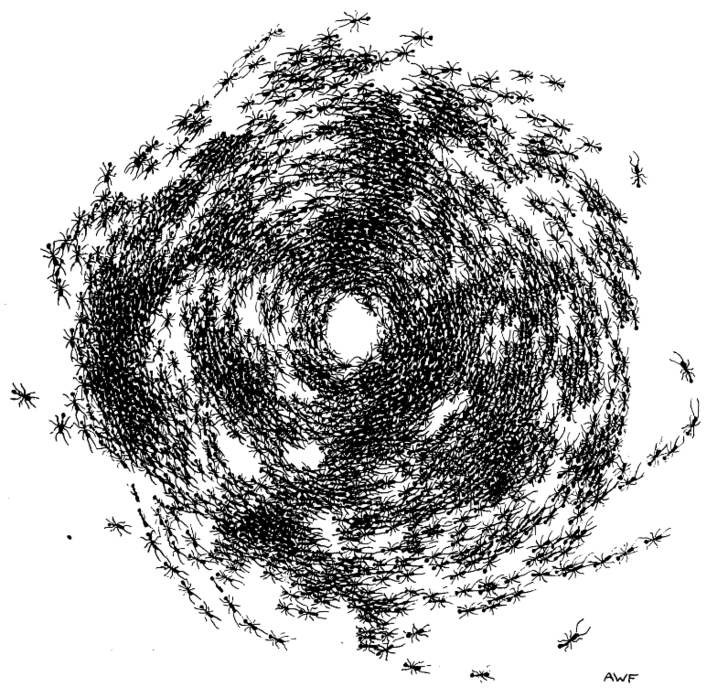


Figure 9: Illustration of an ant-mill; taken from Figure 1 of (Schneirla et al. 1944).

While it is reassuring that our model can replicate behaviour found in other models; we decided to develop an amendment to prevent this behaviour in the future.

The solution we came up with was to vary the initial strength of the markers that get deposited, relative to how many steps the walker has taken from :

$$c_i(\overrightarrow{x}) = \frac{\alpha e^{-\gamma s_i}}{(||\overrightarrow{x} - \overrightarrow{p}_i|| + 1)^\beta} \tag{8}$$

Where s_i is the step count of the walker at the point where the marker was created, and governed by the parameter γ. The walker's step count is a number that ticks up with each turn, but is reset to zero whenever it encounters a resource pile or its base[4]. The effect that this has is negligible if the walker is continuously shuttling resource between a pile and base; but if it ventures further from an objective, the strength of marker it places will decrease until it resets its counter by encountering an objective. Ant-mills will still form with this modification, but eventually dissipate as its members cannot reset their step counter so the marker trail fades.

For example, with $\gamma = 0.05$, a marker initially placed by a walker as it leaves the base will have an initial strength of 1; but two steps later, the marker it deposits will only have an initial strength of $e^{-2\gamma} = 0.905$.

5.3 The effect of the λ parameter

One of discussion points from our project brief was to inspect if there was an optimal amount of randomness in the walkers' decision-making that would allow for the walkers to complete their objective as efficiently as possible. The rationale behind this is that more deterministic walkers will strictly follow a trail, even if it is not the most time-efficient route between the base and resource; while increasing the amount of randomness (decreasing λ) might allow for the walkers to deviate from the path and discover a faster route; improving efficiency in the long run.

We would vary the parameter λ and measure how long it would take a group of walkers to transport all the resources from a set scenario back to their base; averaged over a series of individual runs. This scenario can be summarised with the following list, and visualised in Figure 10.

- 25 by 25 sized grid.
- 10 walkers start from a base at the center of the grid, each facing in a random direction.
- A pile of resources in each corner of the grid, with 20 units of resource at the start.
- The scenario is terminated and the duration recorded once all 80 units of resource have been transported.
- Recorded value will be averaged over 10 trials for each unique λ value.

The results from this experiment are shown in Table 1; and are plotted in Figure 11.

	Duration, t			
Value of λ	Best (quickest)	Worst (slowest)	Average (μ)	Std. Dev. (σ)
0	1,759	2,502	2,076	240
.5	1,274	1,751	1,487	141
1	1,019	1,489	1,279	141
2	629	973	815	111
3	456	766	575	82
4	484	700	548	58
5	432	752	569	91
6	432	682	508	77
7	499	904	621	113
8	488	696	569	66
9	424	781	545	122
10	367	656	526	96

Table 1: Results obtained after 10 trials for each unique value of λ

[4]This still occurs if it isn't making a transaction of resources, i.e. bumps into a resource pile when carrying something as the carrier role.

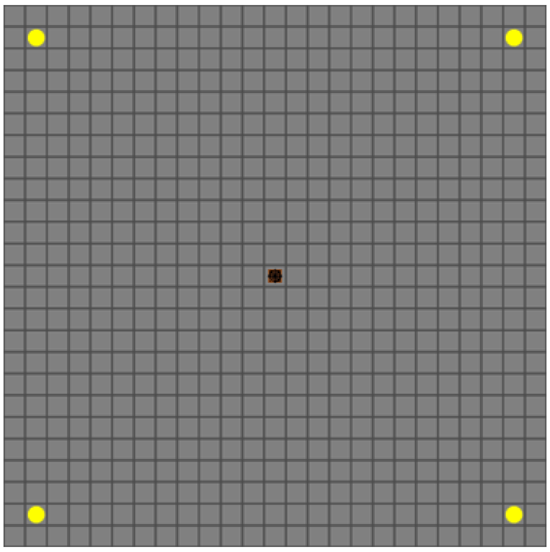


Figure 10: Rendering of the scenario used to generate results for λ comparison.

From these results, it is clear that as the amount of randomness is decreased through λ; the efficiency of the walkers generally improves. Additionally, it is entirely expected that as we make the system more deterministic, the performance between trials gets more consistent (lower standard deviation). However, for the extreme values there are comments to be made:

- The trials where $\lambda = 0$ (equal odds for all 3 choices) were able to still finish in a somewhat reasonable amount of time; and far exceeded our expectations. This comes from the fact that the walkers have a tendency to move forwards due to only being able to choose from the three frontal cells; are turned around once they reach their objective; and that they will eventually bump into their destination due to the relatively small grid and boundary conditions.
- Diminishing returns to both the average time to complete and the consistency occur past $\lambda = 3$

6 Conclusion

To summarise:

- We have produced a model containing simple Walkers which are able to efficiently complete a search & retrieve objective; with only a simple ruleset and
- There is no significant efficiency gain for walkers completing their objective with more randomness; and are instead able to complete the objective a lot faster when moving 'more deterministicly'.
- Our model can be used to model the behaviours of ant species which use pheromones to navigate; and will replicate the same emergent behaviour that comes from the ant's decision making
- Have implemented an amended model which stops our walkers from encountering the same errors that the ants do; and are able to break free from scenarios where they would otherwise be trapped.

In terms of future exploration; there are a few avenues that can be looked into:

- Introducing obstacles such as solid walls to the environment, as something for the walkers to path-find around. This would have applications in allowing the walkers to solve mazes; and in real life applications

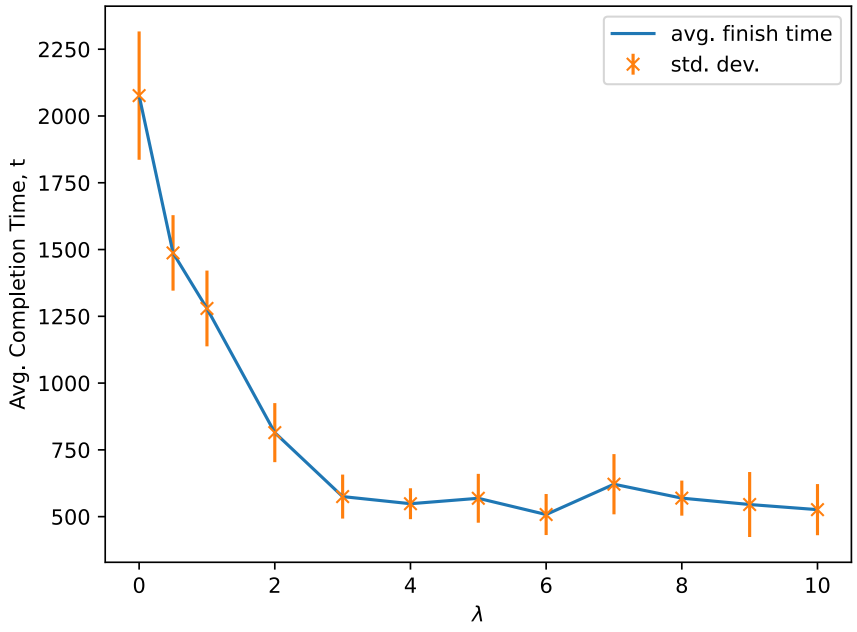


Figure 11: Plot of average time to complete, along with std. dev.

- Experiment with the effect of the γ parameter and its effect on efficiency when completing the objective.
- A continuous version of the model, that operate within a continuous space with small continuous-approximate time-steps; instead of the discrete 2D grid.
- A more optimised implementation that would allow for experiments to take place on a larger grid size; as currently the initialisation of our model operates at $O(n^4)$ time-complexity. This, along with possible remedies are discussed in the Appendix.
- Diminishing returns for increasing λ within our scenario. It may be worth calculating an analytical lower-bound for the minimum duration it would take to complete the objective.
- Actual physical implementation of the system; with small physical robots

7 Acknowledgements

I would like to acknowledge my teammates on this project:

- **Marlene Hudler**; m.hudler-20@student.lboro.ac.uk
- **Clara Noguerol**; c.v.noguerol-18@student.lboro.ac.uk
- **Nathan Vincent**; n.vincent-19@student.lboro.ac.uk

References

Crooks, Andrew T and Alison J Heppenstall (2011). “Introduction to agent-based modelling”. In: *Agent-based models of geographical systems*. Springer, pp. 85–105.

Erhard, Dirk, Tertuliano Franco, and Guilherme Reis (2020). *The Directed Edge Reinforced Random Walk: The Ant Mill Phenomenon.* arXiv: `1911.07295 [math.PR]`.

minutephysics (Mar. 2015). *Computer color is broken.* URL: `https://www.youtube.com/watch?v=LKnqECcg6Gw&amp;t=160s`.

Schneirla, Theodore Christian et al. (1944). "A unique case of circular milling in ants, considered in relation to trail following and the general problem of orientation". In.

Schweitzer, Frank, Kenneth Lao, and Fereydoon Family (1997). "Active random walkers simulate trunk trail formation by ants". In: *BioSystems* 41.3, pp. 153–166.

8 Appendix

8.1 Implementation Specifics

In this section of the Appendix, I will detail specifics of how we implemented the model in python using an object oriented approach; in order to make it easy for someone to follow in our footsteps and recreate our model. Alternatively; you can grab a copy of our implementation from `https://github.com/jacob-toller/mathematical-modelling-4`

8.2 Classes

For this, we made hefty usage of Object Oriented Programming principles. I will now detail the explicit objects that we defined in our code:

- Walker - one instance of this class for every unique walker.
  - Position
  - Direction
  - Job Role
  - Step Counter
  - λ parameter for decision making.
  - γ parameter for combining with step counter when creating new markers.
- MarkerTracker - this processes all of the markers of a specific type; we had one of these for A and another for marker B
  - β falloff parameter
  - T as temporal decay parameter
  - Lookup dictionary/JSON that tracked each individual marker. For each individual marker the following info was stored:
    - Marker position
    - t_i, time deposited at.
    - Initial strength, calculated by the walker when depositing and stored here.
- FoodTracker - similar to the marker tracker, where one object looks after multiple things.
  - Lookup dictionary/JSON that tracked each food pile. For each individual resource pile, the following info was stored:
    - Position of the food pile.
    - Amount of food in the food pile.
- Nest - this functions very similar to an individual food pile; except that there is only one of them to keep track of.
  - Position
  - Amount of food returned.

8.3 Renderer & Telemetry

The renderer was an important part of this project, as it is what allowed us to visually inspect the state of the model when performing preliminary testing and experiments; and is what allowed us to create the intuitive visuals that we used in our presentations to help the viewers grasp the concept of what was going on.

The drawing pipeline is as follows:

1. Create a blank image, labeled as a canvas.
2. Draw the cells on the canvas. This is done in a specific way such that we can still infer the information about the marker concentration of all marker types in an area. For each marker type, we calculate how strong its colour should be (Strong blue for a high concentration of A; pale blue for a low concentration of A; similarly for marker B and green); and then blend[5] these colours resulting in the teal-coloured cells that can be seen in Sub-Figure 6d
3. Draw the food piles and the nest on-top of the grid of cells

In terms of the results obtained; its even more important that we can export numerical results from the system so that they can be plotted as graphs. For each timestep, the system exports the following information.

- The timestep the information was generated on
- The amount of units of resource that are currently stored in the nest.
- How many units of resource are currently being transported by the Walkers.
- How many units of resourcea are still located in the resource piles of the system.

8.4 Optimisations

In our implementation, we have to calculate the concentration map for each of the marker types at the start of every single timestep.

In our first version; this was done by iterating through all of the grid cells and calculating the value of $C\left(\overrightarrow{x}\right)$ directly. For a 20 by 20 grid size, this has to be done 200 times; and then another 200 times for the other type of marker; and took about 0.160 seconds per timestep. Additionally, this was done within a for loop, so each of these calculations is performed in isolation through normal python arithmetic. This is important for our second version.

The methodology we took for our second version was to instead pre-calculate the values that done change before the simulation starts. Let us assume that we have a marker in the position $\overrightarrow{p}_i = (4, 5)$; the distance between this marker and the cells $(3, 3)$, $(19, 16)$ or any other position will always stay the same; so for the marker position $(4, 5)$, we can store the distances from that cell to all of the other cells in a look-up table. Not only that, but we can also raise this distance to the power of β such that we can pre-calculate and store the whole quotient of (8); $\frac{1}{\left(||\overrightarrow{x}-\overrightarrow{p}_i||+1\right)^{\beta}}$. We store one of these lookup-tables for each possible grid position. For a 20 by 20 grid; this means pre-calculating 160,000 values. The second improvement that we made was to use NumPy's array arithmetic when applying these lookup tables; as they allow for significantly faster processing compared to the for-loops that we used in the first version. These modifications take the processing time for a 20x20 grid down to 0.001 seconds per tick; while only taking an extra few seconds to start up. The downsides to this system is that, for an n by n grid; the startup pre-calculations operate with $O\left(n^4\right)$ time-complexity; so a 100x100 grid takes 6 minutes to start up compared to the few seconds of a 20x20 grid. However once startup has finished, the 100x100 grid runs almost as quickly as the 20x20 grid; a massive improvement.

From this point, we refrained from making further improvements to this system, as now the biggest bottleneck in the system for processing a tick was the rendering system. Therefore, when repeating trials for generating the average results; we disabled the renderer and were able to generate all of our required data in under 10 minutes.

That being said; we did identify room for future improvement. Currently, we are pre-calculating an entire grid for every single possible marker location p_i. However; this is simply redundant. A marker in the position $(4, 5)$ will have the exact same pre-calc grid as a marker in $(4, 4)$, except shifted over by

[5]This is done by taking the root of the average luminosity-squared for the colour, as seen in (minutephysics 2015)

one. Hence it would be possible to generate a pre-calculation grid for a space that is roughly twice as wide and high; and then just shift that for the appropriate local marker position.

Another point to be made is that this is an easily parallelisable task; with thousands of numbers that need calculating but can be done independently of each-other. This makes it a great candidate for multi-threading and GPU based acceleration.