# Ants and Reinforcement Learning Algorithms

Akanksha Paudyal, Alice Yang, Damon Gee, Lily Salem, Tom Fan
Department of Computer Science
Carleton University

December 7, 2025

**Abstract**

Ant Colony Optimization (ACO) algorithms have seen extensive research, inspired by ants' strategy of creating pheromone trails between food sources and their nest to optimize colonial foraging. In this project, we have implemented an ant-inspired agent that navigates an open world to retrieve an object, overcoming limited environmental awareness by following a path. We present implementations of Q-learning, SARSA, and Dyna-Q and compare their training performance. Our goal was to create an agent that can succeed in a multiagent setting alongside copies of itself sharing one policy, though due to time constraints this will have to be future research.

## 1 Introduction

### 1.1 What problem does this project focus on?

Many of the greatest innovations in the world have not involved humans at all. Through nothing more than natural selection, countless brilliant methods of survival have emerged, and those found in social species are among the most complex and fascinating. Ants, known for their strength in numbers, have gained attention not only among biologists, but computer scientists as well. An ant colony can be viewed as a multiagent system working towards a collective goal – and the sheer scale of some species' colonies is a testament to how successful this strategy is. Furthermore, one adaptation of theirs has gained particular attention: the use of pheromone trails while foraging. Regardless of colony size, nearly all species make use of this organized foraging behaviour.
In the typical case, worker ants begin by wandering randomly searching for food. Once an individual discovers an exploitable food source, it returns to the nest. As it returns, however, it lays down a chemical pheromone trail recognizable by others of its species. Other foraging ants prioritize following the trail, leading them to the food source without the discovering ant having to directly share the location. This is an immensely more efficient approach than fully random exploration [4]. As the chemical is not permanent and evaporates over time, the trail must be maintained if the food source is to be exploited. Every returning ant lays down more pheromone, strengthening it. This also enables another significant behaviour. As they make their way to the nest, some ants take different, and possibly shorter, paths. Without knowing the distances, an ant that chooses to use a shorter trail will complete its task more frequently, therefore reinforcing the pheromone more frequently as well. Over multiple runs, the stronger trails will be chosen by other ants more often, eventually leading to the identification of the shortest path for the whole colony [1]

### 1.2 Why should we care about this problem?

In contrast to honeybees, for example, which dance in their hives to directly communicate the distance and direction of food sources, the trail method is far more indirect. This natural strategy has been the focus of a significant amount of prior research, and for good reason. Ant Colony Optimization (ACO) algorithms date back to 1996, when the first one was used to solve the travelling salesman problem [1]. Besides mathematical theory, they have seen use in vehicle routing, nanoelectronic circuit design,

image edge detection, and more.
For this project, we have gone back to the roots of this field and taken inspiration from nature. We have challenged ourselves to design an agent that can navigate an open world to retrieve an object, overcoming limited environmental awareness by following a path. Like real ants, this system is most applicable to multiagent settings where agents must communicate locations to each other, but direct contact, remote messaging, or other methods of transferring detailed data are inconvenient or impossible. Rather than the more theoretical applications discussed above, our approach is more relevant to concrete applications like swarm intelligence robotics, perhaps in dangerous or obscured environments.

### 1.3 Environment Specification

For the environment we categorize cell types as empty cell (E), obstacle (O), food (F), queen (Q), the pheromone trail (T) and signal specifying if that cell is closest to the queen (S).

### 1.4 State Space

The state space is discrete consisting of 4 states: `hasFood` and the three cells in its vision. The `hasFood` state is a boolean indicating whether the worker ant is currently carrying the food. The three cells part of the state space represent the contents of the three forward-facing hexagonal cells relative to the ant's current position and orientation.

### 1.5 State Space Calculation

We calculate the total number of reachable states by considering the two cases based on whether the ant is carrying food.

#### *1.5.1 Case 1: `hasFood` = false*

When the ant is not carrying food, each of the three vision cells can contain one of four cell types: empty (E), obstacle (O), trail (T), or food (F). Therefore, the number of possible states is:

$$N_{\text{no food}} = 4^3 = 64 \tag{1}$$

#### *1.5.2 Case 2: `hasFood` = true*

When the ant is carrying food, each of the three vision cells can contain: empty (E), obstacle (O), signal (S), or queen (Q). However, not all combinations are reachable due to constraints on how signals and queens appear.

**Non-queen states:** Each cell can be E, O, or S, giving $3^3 = 27$ combinations. However, due to the nature of signal cells (S), the following configurations are unreachable:

- SES (signal-empty-signal)
- SOS (signal-obstacle-signal)
- SSS (all signals)

This reduces the non-queen states to:

$$N_{\text{non-queen}} = 3^3 - 3 = 27 - 3 = 24 \tag{2}$$

**Queen states (terminal):** When the queen (Q) appears in the ant's vision, the state is terminal. The queen can appear in any of the three vision positions, with the remaining two cells being either E or O. This gives:

- Position 1: Q in first cell, $2^2 = 4$ combinations for remaining cells (Qxx)
- Position 2: Q in second cell, $2^2 = 4$ combinations for remaining cells (xQx)

- Position 3: Q in third cell, $2^2 = 4$ combinations for remaining cells (xxQ)

Therefore, the number of terminal queen states is:

$$N_{\text{queen}} = 3 \times 2^2 = 3 \times 4 = 12 \tag{3}$$

The total number of states when carrying food is:

$$N_{\text{has food}} = N_{\text{non-queen}} + N_{\text{queen}} = 24 + 12 = 36 \tag{4}$$

#### *1.5.3 Total State Space*

The total number of reachable states in the environment is:

$$N_{\text{total}} = N_{\text{no food}} + N_{\text{has food}} = 64 + 36 = 100 \tag{5}$$

### 1.6 Action Space

The action space is discrete consisting of 5 actions: move forward, move left, move right, pick up food and give food to the queen. We place the following constraints to the action space:

- The move actions only succeed if the destination cell is labeled E. Actions fail if cells are out of bounds or occupied.
- The pick up food action only succeeds if there is at least one F cell in the ant's vision and the ant is not currently carrying food.
- The give food to the queen action only succeeds if there is the Q cell in the ant's vision and the ant is currently carrying food.

### 1.7 Reward Function

The reward function returns integer rewards based on the action taken and its outcomes. The rewards for each action are as follows:

| Action/Outcome | Reward |
|---|---|
| Moving to a trail cell (T) or signal cell (S) | +1 |
| Moving to an empty cell (E) | 0 |
| Picking up food | +3 |
| Giving food to the queen | +10 |
| Invalid move or action | -1 |

Table 1: Reward function for different actions and outcomes.

## 2 Approaches

### 2.1 Previous Research

Our research on ant-inspired systems spans biological simulation to practical optimization problems. Source 1 [6] presents a biological-inspired approach where agents share neural network architectures interacting through pheromone-like chemical signals that evaporate over time. This is something we implemented in our environment as well. Source 3 [3] emphasizes how pheromone trails enable stigmergic communication, allowing relatively simple individual ants to collectively solve complex tasks through indirect coordination. The relationship between natural ant behavior and computational problem-solving has been explored through the Ant Colony Optimization (ACO) framework. ACO has proven effective across various applications. Source 2 [2] demonstrates human-in-the-loop extensions of the MAX-MIN Ant System for solving the Traveling Salesman Problem, showing

that hybrid human-machine intelligence can outperform purely algorithmic approaches. Source 4 [5] applies ACO principles to real-time game environments, specifically developing NPCs for Ms. Pac-Man, demonstrating the framework's adaptability to dynamic environments which is a crucial consideration for your simulation where food locations and ant positions constantly change.

### 2.2 Custom environment

Early on in our project, we invested a majority of our time developing a custom 2D hex-grid world with a 3D hexagonal coordinate system representation. This coordinate system is built in such a way that each hexagonal cell has 6 neighbours. Unlike a square grid which has more complexity regarding diagonal movement and vision, all neighbours in a hex grid are equidistant. During world generation, each cell is normalized before storing the cell coordinate. The cell coordinate is stored around the constraint that a valid hex grid cell must have at least one coordinate equal to 0. We achieved this using normalization to convert the coordinate representation to follow this constraint. To achieve normalization, we subtract the minimum value from all three coordinates, ensuring at least one coordinate is 0. This allows equivalent coordinates to map to the same cell. It is important to note that equivalent coordinates in a hex grid system are possible, and we can think of any given point (x,y,z) as equivalent to (x+c, y+c, z+c). Without normalization, coordinates like (5,3,8) would not map to a valid array index and consequently cause errors. Normalization also helps with more efficient storage. We avoid storing duplicated cells multiple times under different coordinates, otherwise creating unnecessary bloat. It also makes for efficient cell access and comparisons.

### 2.3 Algorithm implementations & justifications

We implemented 3 temporal difference control algorithms. First, we implemented an off-policy Q-learning algorithm that updates Q-values using the maximum Q-values of the next state. Through this process the agent learns the optimal policy by leveraging epsilon-greedy exploration. We started with this algorithm as our base since off-policy learning helps discover optimal paths even when exploration is not always optimal, this is critical for when good choices are rare (given our reward structure has few actions that yield a positive reward). Even if through exploration the ant does not pick up any food, Q-learning can still learn that picking up food is valuable by considering the best possible future actions. The ant agent learns to balance finding food (the exploration) and following trails back to the queen( exploitation). Second, we implemented the SARSA on-policy algorithm which uses the Q-values of the next selected action from the epsilon-greedy policy. An on-policy algorithm allows us to compare whether a conservative on-policy algorithm that makes safer updates performs better or worse than a constrained off-policy learning. Our third and final algorithm we implemented is Dyna-Q, which builds off of Q-learning. It keeps track of a learned model of the environment and performs planning updates by replaying simulated experiences from that model, giving way to potentially improving the sample efficiency. Since the ant agent can only see the three hex cells in front of it and has limited observations, we sought out to determine whether a model-based planning approach would help the agent learn better from past experiences. We thought this could help the ant agent revisit similar states such as approaching food and following pheromone trails, and planning would help reinforce those behaviours.

## 3 Empirical Studies

In this section, we compare the three temporal-difference control algorithms implemented in this project: Q-learning, SARSA, and Dyna-Q. Evaluate overall task performance, training efficiency, and hyperparameter sensitivity, and investigates whether Dyna-Q's model-based planning offers advantages under the environment's limited observability.

### 3.1 Experimental Design

For the experiment, we used a consistent training set up to ensure a fair comparison across algorithms. Each agent trained for 500 episodes, with a limit of 600 steps per episode. Exploration followed an

$\epsilon$-greedy policy with exponential decay, and episodes terminated early when the agent successfully found food and delivered it back to the queen. For each run, we recorded per-episode reward, episode length, and food deliveries, and computed average reward, average steps, average deliveries over the last 50 episodes to help us analyze the results. We also performed greedy evaluations ($\epsilon = 0$) over 50 rollouts to measure final policy quality independently of exploration. We explored a small but systematic hyperparameter grid: learning rate $\alpha \in \{0.001, 0.01\}$, discount $\gamma \in \{0.9, 0.99\}$, initial $\epsilon \in \{0.3, 0.5\}$, $\epsilon$-decay $\in \{0.99, 0.995\}$. For Dyna-Q, we also tested different planning_steps $\in \{3, 5\}$, to compare whether additional simulated experience improves learning efficiency.

### 3.2 Comparative Performance of Algorithms

| Algorithm | 50-ep. avg. return | Train avg steps | lr / $\gamma$ / $\epsilon$ / decay | Deliveries |
|---|---|---|---|---|
| Q-learning | 56.9 | 100.9 | 0.001 / 0.99 / 0.3 / 0.99 | 1.0 |
| Dyna-Q (p5) | 49.0 | 137.9 | 0.01 / 0.99 / 0.5 / 0.995 | 0.99 |
| Dyna-Q (p3) | 47.6 | 81.0 | 0.01 / 0.9 / 0.3 / 0.995 | 1.0 |
| SARSA | 47.3 | 73.0 | 0.01 / 0.9 / 0.3 / 0.995 | 1.0 |

Table 2: Best-performing configurations per algorithm. Rewards are averaged over the last 50 training episodes; deliveries $\approx 1.0$ indicates nearly every episode achieved a food return.

**Q-learning.**

**Q-learning**

Overall, Q-learning achieved the highest average reward among all methods. The best configuration ($\alpha$=0.001, $\gamma$=0.99, $\epsilon$=0.3, decay=0.99) reached:

- 56.9 reward (average over last 50 episodes)
- ~101 steps per episode
- Delivery success $\approx 1.0$ (meaning nearly all episodes achieved the goal once)

These results demonstrate that off-policy bootstrapping is effective in this environment, where positive rewards (pickup $+3$, delivery $+10$) are sparse. As discussed in Section 2 (Approach), Q-learning's optimistic value backups allow it to propagate high rewards even from rare successful trajectories. However, these high-reward policies usually lead to longer episodes, indicating that Q-learning often selects longer but high-value paths.

**SARSA.** SARSA produced slightly lower peak reward but with substantially shorter episode lengths, demonstrating the more conservative nature of on-policy updates. The best SARSA configuration achieved around 47.3 reward, 73 steps and the shortest observed SARSA episode is 55 steps. SARSA is more stable because it focuses on the actual next action rather than the maximum possible future value, limiting over-estimation and encouraging efficient paths. Compared to Q-learning, SARSA consistently yielded more compact and direct routes to the goal.

**Dyna-Q**

Dyna-Q was evaluated to determine whether planning through model replay provides measurable benefits given the ant's limited vision. Performance depended strongly on planning depth:

- Best p5 configuration: ~49.0 reward, ~138 steps
- Best p3 configuration: ~47.6 reward, ~81 steps

Although higher planning improved reward in some cases, it frequently increased episode length, suggesting that replay amplified the effect of aggressive updates or high $\epsilon$. Dyna-Q therefore did not surpass Q-learning's peak reward, and it often balanced between Q-learning and SARSA in terms of reward and efficiency. This aligns with our expectation (Section 2) that model-based planning may be limited by the small state space and modest movement penalties.

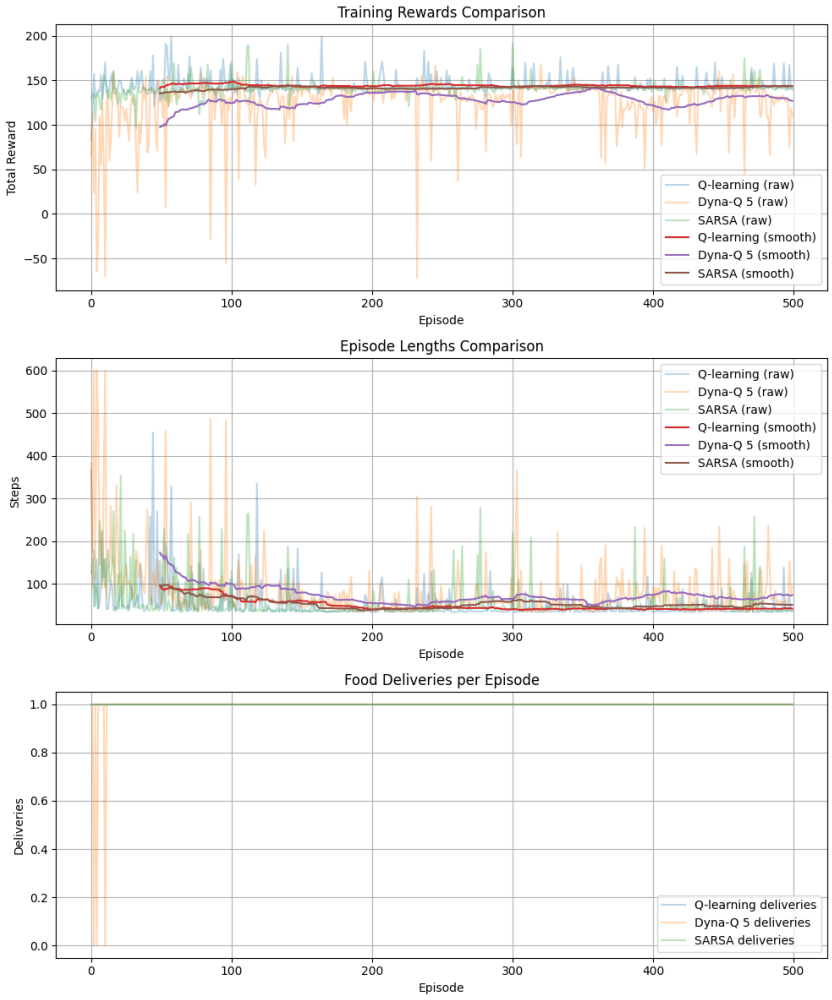


**Figure 1:** Training rewards and episode lengths comparing Q-learning, Dyna-Q and SARSA (with shared hyperparameters).

This graph shows all three methods converge to a close reward level range, but Q-learning and Dyna-Q show slighly higher early variance, while SARSA remains more stable. Episode lengths decrease fastest for SARSA, reflecting its tendency to learn shorter, more direct trajectories. Dyna-Q exhibits greater fluctuation in both rewards and steps, consistent with planning amplifying noisy updates. The delivery plot shows that all algorithms quickly achieve reliable one-delivery episodes, indicating that in this simple environment, differences lie mainly in training stability and efficiency rather than final task success.

**Greedy Evaluation.** When evaluated without exploration ($\epsilon = 0$), all three algorithms converged to highly similar deterministic policies: approximately 45 reward, 34–36 steps, and consistent delivery success. This indicates that algorithmic differences primarily affect learning dynamics rather than the quality of the converged policy.

### 3.3 Analysis of Results

#### *3.3.1 Hyperparameter Sensitivity*

Learning rate had the strongest impact: Q-learning performed best with $\alpha = 0.001$, while $\alpha = 0.01$ destabilized updates, especially for Dyna-Q; SARSA handled larger $\alpha$ more reliably. Q-learning also favored moderate exploration ($\epsilon = 0.3$ with fast decay), whereas Dyna-Q sometimes benefited from higher $\epsilon = 0.5$ at the cost of longer episodes, and SARSA remained robust. A higher discount factor ($\gamma = 0.99$) improved Q-learning's reward, with more configuration-dependent effects for SARSA and Dyna-Q. Increasing planning depth improved Dyna-Q's reward in limited cases but generally lengthened episodes, with $p = 3$ more stable than $p = 5$.

#### *3.3.2 Time and Space Considerations*

Computationally, all methods have similar per-step cost except Dyna-Q, which adds an extra $O(\text{planning\_steps})$ update. With planning_steps $\in \{3, 5\}$, this overhead is small, but our results show no clear sample-efficiency gain to justify it. The environment's simple state representation likely limits the benefit of deeper planning.

### 3.4 Interpretation.

These results indicate that, in this environment, reward structure and limited vision restrain the benefits of model-based planning. Q-learning excels in extracting reward from sparse high-value transitions, while SARSA remains the most step-efficient. Dyna-Q offers intermediate performance and could demonstrate clearer advantages in environments with higher penalties for wandering, richer sensory inputs, or more complex dynamics.

## 4 Conclusions

In this paper, we introduced our custom reinforcement learning environment and an agent designed to operate within it. Evaluating the agent with multiple common RL algorithms including on-policy and off-policy methods, we found that it was able to reliably improve its per-episode performance over the course of a 500-episode training epoch. Notably, this is despite the state perception constraints we placed on the agent, having only given it information on 3 squares immediately adjacent to its current position in order to inform its action selection.

However, some intended features, testing, and stretch goals remain incomplete. Given more time, we would prioritize completing these stretch goals that we had set previously, including:

- Experimenting with the state space and state awareness of the agents - deliberately limiting the observation space of the agent was part of our experimental design. However, we also wanted to test the effects of increasing this observation space on the behavior of the agents using various algorithms.
- Multi-agent training - ants display a remarkable level of collective intelligence [4]. While we tested a single agent using predesignated pheromone trails, we had a grander vision of many agents collaborating in the environment in order to forage for resources.
- Batch and offline training - in addition to the previous goal, we would also aim to do some batch or offline training, as agents would be able to appear and disappear within the course of an episode.

In conclusion, our results from testing the reinforcement learning algorithm showed that a simulated ant-like agent can learn to act effectively in gathering resources in a complex gridworld environment using common RL techniques. It can even overcome constraints like an extremely limited observation space. This demonstrates that our project is a good proof-of-concept for a broader simulation system for ant simulation as it stands, and can yield even more significant observations and results given further development.

## References


[1] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):1–13, 1996.

[2] Andreas Holzinger, Markus Plass, Michael Kickmeier-Rust, Katharina Holzinger, Gloria Cerasela Crişan, Camelia-Mihaela Pintea, and Vasile Palade. Interactive machine learning: Experimental evidence for the human in the algorithmic loop. *Applied Intelligence*, 49:2401–2414, 2019.

[3] Paulo E. Merloti and Joseph Lewis. Simulation of artificial ant's behavior in a digital environment. In *Proceedings of the International Conference on Artificial Intelligence*, 2005.

[4] E. David Morgan. Trail pheromones of ants. *Physiological Entomology*, 34(1):1–17, 2009.

[5] Gustavo Recio, Emilio Martin, César Estebanez, and Yago Saez. AntBot: Ant colonies for video games. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):295–308, 2012.

[6] Ryosuke Takata, Yujin Tang, Yingtao Tian, Norihiro Maruyama, Hiroki Kojima, and Takashi Ikegami. Evolving collective AI: Simulation of ants communicating via chemicals. In *Proceedings of the Artificial Life Conference*, volume 35, page 112. MIT Press, 2023.