# Comparative Investigation Between Model-Based Heuristics and State-Based Learning Methods for Intelligent Agent Pathfinding in Generalised Warehouse Environments

CHUTIYADA CHEEPCHIEWCHARNCHAI, 20421271

## 1 Introduction

### 1.1 Background

In modern logistics and supply chain operations, the role of autonomous mobile robots in warehouse environments has become increasingly significant. These agents must navigate efficiently and retrieve items with minimal human input, while avoiding collisions and bottlenecks. A central challenge in deploying such systems is the multi-agent pathfinding (MAPF) process. This is the process of computing optimal, conflict-free paths for multiple agents operating simultaneously.

Traditional model-based heuristics like A* algorithm rely on explicit environmental models and predefined rules to calculate optimal paths. These methods are deterministic and interpretable but may struggle with adaptability and scalability in dynamic multi-agent systems.

In contrast, non model-based approaches like genetic algorithms and neural networks do not require explicit environmental models, and can rely on learning from their immediate surrounding factors. i.e., their states. This offers them adaptability and robustness in changing environments. These methods can infer complex navigation strategies from environmental feedback, or historical data, making them particularly promising for decentralised or emergent coordination in robotic swarms.

### 1.2 Aim

This project investigates and compares the performance of model-based and state-based learning strategies to solve the MAPF problem in warehouse environments. The project specifically aims to evaluate the following pathfinding methods:

- **Model-based heuristics**: Rule-based Algorithms and A* Heuristics for deterministic path planning.
- **State-based learning methods**: Genetic Algorithm and Neural Network-based Multilayer Perceptron for adaptive behaviours.

The primary research question driving this investigation is:

> "Between model-based and state-based strategies, which intelligent agent approach is most effective in pathfinding and collision avoidance for autonomous agents in warehouse environments?"

By implementing and comparing these strategies under identical simulated conditions, I aim to identify the most robust and scalable solution for intelligent agent coordination in warehouse logistics.

## 2 Related Work

### 2.1 Relevant Learning Approaches

Classical pathfinding strategies such as A*, Theta*, and conflict-based search (CBS) form the fundamental basis of classical pathfinding. These methods can provide near-optimal paths without the requirement of the full environment knowledge. However, they often lack flexibility in dynamic, multi-agent contexts. Prioritised planning variants such as HCA* and WHCA* developed by Silver [1] offer improved scalability by assigning temporal or spatial priorities among agents but still exhibit limitations in environments with frequent agent interactions.

Neural network-based strategies are increasingly used for policy learning in navigation tasks. These models generalise over state representations and approximate complex decision boundaries, making them suitable for dynamic environments. Supervised or imitation learning methods can train such networks on optimal trajectories derived from planners like A*.

Genetic algorithms have been applied to many path optimisation problems, due to their robustness in larger, high-dimensional spaces. Kordos et al. [2] demonstrated how genetic algorithms can effectively improve warehouse operational efficiency by evolving agent movement strategies to minimise travel distances and avoid collisions. Using Cost = $\sum_{n=1}^{N_{\text{ord}}} \text{Route}_{\min}(n)$ as their warehouse produce placement optimisation algorithm, where $N_{ord}$ is the order number and $\text{Route}_{\min}(n)$ is the shortest route found for the n-th order, highlighting genetic algorithms' robustness and adaptability towards route optimisation in high-dimensional logistic problems.

### 2.2 Comparison Studies

Comparative research has traditionally focused on reinforcement learning versus classical heuristics (e.g., A*, CBS). Frameworks like PRIMAL2 has demonstrated superiority over such methods in lifelong multi-agent pathfinding settings through metrics like makespan and throughput. LNS2+RL [3] extended this work by incorporating curriculum learning and adaptive switching, showing improved coordination in challenging warehouse-like scenarios.

However, most genetic algorithm studies are isolated, focusing on optimisation rather than heuristic benchmarking. Systems like MAPPER [4] show genetic algorithm's strengths in parameter tuning and design space exploration but are not evaluated side-by-side with deterministic or learned policies in navigation contexts.

This lack of comprehensive benchmarking between rule-based, genetic, and neural strategies represents a clear research gap, which this project directly addresses.

### 2.3 Contribution

This project contributes by offering a side-by-side comparison of two different data-driven approaches, model-based and state-based, under the same simulated warehouse conditions. The results aim to clarify:

- Which approach yields the best pathfinding and collision avoidance agents?
- How scalable are these methods with increasing agent count and environment complexity?

By providing direct performance comparisons, this work clarifies the practical strengths of each data-based approach in autonomous warehouse operations and lays the groundwork for future understanding of data collection and usages.

## 3 Design and Implementation

### 3.1 System Architecture

This project is built upon the Model-View-Control (MVC) architectural pattern, as it best fits the requirement of the project, allowing the separate development of the environment (model) and the agents (control). As the MVC structure promotes modularity, it allows for ease of testing and the comparison of multi-agent learning approaches within the same simulation framework.

**Model** - The model is used to manage all information on the simulation, including the grid-based environment, placement of different types of blocks (see Section 3.2). It is considered to be the agent of the project. To help standardise the implementation, all control (agent) types will inherit from a shared base class "`baseAgent.py`", which defines common methods such as `moveAgents()`, `_validate_move()`, and `_get_valid_neighbours()`. This ensures consistent testing conditions.

**View** - The view provides the visualisation for the simulated warehouse and the agents.

**Controller** - The control provides instructions on how the agents should move. Each control class represents a different type of heuristic approach, which can be easily swapped out, and trained individually.

```
main.py
  |
  ⊢ control
  |    ⊢ baseAgent.py
  |    ⊢ astarAgent.py
  |    ⊢ GAAgent.py
  |    ⊢ NNAgent.py
  |    ⊢ rulebasedAgent.py
  |
  ⊢ model
  |    ⊢ model.py
  |    ⊢ modelAgent.py
  |
  ⊢ view
       ⊢ baseView.py
       ⊢ textView.py
       ⊢ tkView.py
```

Fig. 1. The project's MVC structure.

The project also implements base class and inheritance for view and control classes, allowing for faster development and conforming the classes to the same sets of functions so that they can actually be swapped between each other. Fig. 1 shows the top-down diagram of the MVC implementations. This figure excludes other training files.

## 3.2 Environment Design

### *3.2.1 Pre-existing Environments*

As the focus of this project is on autonomous agents and pathfinding, initial attempts were carried out to integrate existing environments. Gymnasium[1] was chosen, as it provides both the environment and training functionalities. Specifically, the environment "`Taxi-v3`" was integrated into the project.

However, an issue with implementing this API is that Gymnasium is developed specifically for reinforcement learning, but does not natively include others. Not only that, but the training functions are bounded to the environment, meaning the project could not be developed in the modular MVC model as intended. As I wanted to be able to benchmark the different strategies in the same environments/circumstances, I decided against using Gymnasium for this project.

Other solutions were also researched (PettingZoo[2] and Flatland[3]). However, those solutions, and many other open-source materials, are all built on top of Gymnasium, and therefore hold the same issues. Therefore, the decision was made to develop an environment from scratch.

[1]https://gymnasium.farama.org/
[2]https://pettingzoo.farama.org/
[3]https://flatland.aicrowd.com/intro.html

### 3.2.2 *Developed Environment*

The plan of the environment design is to produce a grid-style representation to allow customisation into different blocks of objects, while maintaining enough abstract to for the intelligent agents to properly perform in.
The environment has a total of five different types of objects:

- Pathway - Blocks where the agents are allowed to walk on. Represented as "`0`".
- Agent - The autonomous robots tasked with retrieving items from the shelves. Represented as "`1`" on the text view, and as "`A_`" on the Tkinter rendered interface.
- Charging station - Blocks where agents stand on, to recharge their energy. Represented as "`2`".
- Drop off point - Blocks where agents drop off the retrieved items. Represented as "`3`".
- Shelf - Blocks where agents retrieve item from. Represented as the value of "`4`" onward.

These blocks are generated based on a layout that simulates a real warehouse, maximising shelve positions, while still generating enough pathway to allow efficient agent movement and access to all shelves. As the project requires multiple warehouse sizes for training, the environment is developed to be able to generate at different sizes. Appendix A shows how this logic is implemented.

The final design generates columns of shelves in groups of three, with the middle columns set as pathways. Generated column of shelves in the final row are removed if it will generate shelves that won't be accessible by the agents. The columns are also broken up to a maximum of five rows, to enable more efficient movements for agents. The top and bottom two rows are set to generate full rows of shelves, next to a full row of pathways. Finally, the charging and drop-off points are placed at the top and bottom rows, replacing some shelves. Fig. 2 shows a visual example of this generated environment.

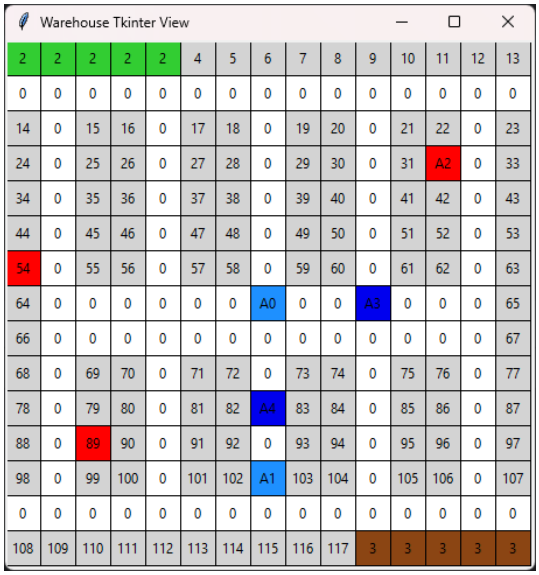


Fig. 2. An example of a warehouse environment, generated at size (13,14).

Two types of views were developed for this project. Initially, the "`TextView`" was created to ensure correctness of the environment implementation, such as populating blocks to the correct coordinates, and spawning agents to randomised location. This view is based on text, using "`print()`" to show each snapshot of the simulation. However, this view was difficult to use for tracking changes between each agent's movement. Therefore, a

graphical interface was created using Tkinter, to help visually follow these movements. This view is called "TkView", and includes dynamic colour-coding for better tracking.

## 3.3 Agent Design

### *3.3.1 Introduction of Agents*

The agents created for this project aim to most efficiently retrieve items from warehouse environments, avoiding collisions with other agents, and diverting from a bottleneck traffics at drop off areas. Another aspect of the agents are their tasks to ensure activity by heading back to their charger stations when their energy levels are low. This section will explore how each strategy tries to implement these goals onto the agents.

### *3.3.2 Model-Based Agent*

The model-based agents in this project are built on classical AI planning principles, which rely on predefined representations of the environment, and heuristics calculation, to compute optimal paths. These agents are explicitly programmed to use the knowledge of the spatial layouts of environments to inform their transition decisions.

Two distinct model-based strategies were implemented:

- **Rule-Based Algorithm**: This approach uses predefined rule-based calculations of the environment information to guide agents through the warehouse. A total of four simple rules were implemented:
  (1) Immediately chooses the neighbouring block if that block is the agent's current target.
  (2) If no target block is adjacent, use the Manhattan Distance Heuristics given by $h(n) = |x_1 - x_2| + |y_1 - y_2|$ to help calculate a score for each neighbouring block, as it best aligns with the grid-based structure of the simulated warehouse. The full formula for the score is then given as:

$$\text{score} = -h(n) \times (1 + \text{round}(\text{rand}(-\alpha, \alpha), 2)) \tag{1}$$

  - $h(n)$: Manhattan distance between the agent and the target.
  - $\alpha$: Alpha factor for introducing randomness when equal scores are found.

  (3) Discourage back-tracking and staying still by reducing the move score by 2 points if one of those paths are being considered.
  (4) Select the highest scoring move.
- **A* Search Algorithm**: A* is a classic pathfinding algorithm which combines the actual cost to reach a node, g(n), and an estimated cost to reach the goal, h(n), to find the shortest possible path:

$$f(n) = g(n) + h(n) \tag{2}$$

  This implementation will also use the Manhattan distance as the heuristic h(n). The A* algorithm will enable agents to make informed and optimal navigation decisions, and dynamically recalculates paths as needed when confronted with other agents blocking the path.

### *3.3.3 State-Based Agent*

The state-based agents in this project are designed using artificial intelligence techniques that rely on learning from immediate surrounding states rather than explicit environmental models. These agents adapt their behaviours based on local observations without a predefined map or rules. This approach allows for a more flexible and scalable solution, especially in dynamic or partially observable environments.

In order to develop these algorithms, however, a lot of predetermined data must be collected. This was done through using the A* algorithm on randomly generated environments. The A* algorithm used on the unweighted grid map becomes a breadth first search algorithm. For this project a total of 30 maps were generated to retrieve

a total of 103,622 move instances between all agents, which is then used for training state-based models. The implementation to simulate this dataset can be found at "`control/training_data/DataCreator.ipynb`".
The two learning-based strategies were implemented in this projects are:

- **Genetic Algorithm**: This learning algorithm uses the principles of evolutionary computation to discover effective navigation policies over successive generations. Each agent's policy is encoded as a chromosome and evaluated with a fitness function to calculate navigation outcomes. The project uses the "`PyGAD`" library and uses the following GA features:
  (1)*Initialisation*: A population of agents with randomly generated weight.
  (2) *Evaluation*: Each state/optimal move pair from the dataset is compared against the weight, where the weight is then evaluated and changed.
  (3) *Selection*: The 8 best performers of each generation is chosen to reproduce in the next round. Four of those parents are also chosen to stay onto the next generation, ensuring the best solutions are being kept alive.
  (4) *Crossover and Mutation*: Single point crossover and 10% random mutation is chosen at the algorithm's genetic operators, in order to introduce small random changes to promote diversity.
- **Neural Network Agent**: The neural network agent leverages deep learning to map environmental states to navigation actions, through training the neural network. As a viable dataset has been produced, this algorithm will be learning through supervised learning, in order to help generalise to unseen situations. For this project,the neural network is implemented using the "`scikit-learn`" library to design for the following implementation:
  (1)*Architecture*: A fully connected multilayer perceptron (MLP)
  (2) *Hidden Layers*: 4 hidden layers, each containing 20 neurons.
  (3) *Activation Function*: ReLU (Rectified Linear Unit)
  (4) *One-Hot Encoding*: Use one-hot encoding to separate the different categories of visible neighbouring blocks, increasing the input to 35 nodes.

These two algorithms receive a total of 11 basic perceivable information as input features: x position, y position, target x position, target y position, value of block above, value of block below, value of block at left, value of block at right, item carrying boolean, amount of energy left, and charging required boolean. The features are preprocessed before being fed into the alearning agents. The output of these agents are a label representing one of five possible movement decisions: 0 (up), 1 (down), 2 (left), 3 (right), 4 (stay),

This structured state representation will allow agents to learn navigation strategies based on local conditions and task context, enabling robust decision-making in complex warehouse environments without relying on a global model.

## 4 Methodology

### 4.1 Experiment Design

To evaluate the performance of different intelligent agent strategies, a series of controlled simulations will be conducted across warehouse simulations of varying sizes. Agents of all four types (rule-based, A*, genetic algorithm, and neural network) will be tested under the same sets of conditions to ensure fairness.

A total of three sets of experiments will be conducted, each with different amounts of agents: 5, 10 and 20. Each set will run ten autogenerated warehouses simulations.

For each simulation, agents are tasked with retrieving a set number of randomly generated items and delivering them to drop-off points while avoiding collisions and managing their energy levels by returning to charging stations as needed.

Each simulation will allow a maximum of 1,200 time-steps and request a total of 100 orders. It will run four times to deploy the four pathfinding algorithms.

## 4.2 Evaluation Metrics

The following metrics were selected to capture key aspects of agent performance in pathfinding, task completion, and survival:

- **Percentage of Order Completion**
  The percentage of orders completed portray the ratio of successfully delivered items per simulation. This directly relates to the main objective of the agents, and reflects each strategy's effectiveness in completing the assigned tasks under real-time constraints. Therefore, this metric is the most important in the experiment
- **Real-Time Performance**
  This metric measures the average time required by each strategy to compute each course of action. Lower computation times indicate higher algorithm efficiency and real-time responsiveness, two critical factors in time-sensitive warehouse operations.
- **Number of Dead Agents**
  This metric tracks agents that fail to complete their tasks due to running out of energy before recharging, or getting stuck in a blockage that they cannot get out of. Fewer dead agents indicate better planning and avoidance of collisions.

Together, these metrics provide a comprehensive view of agent performance by balancing operational success (order completion), algorithmic efficiency (real-time responsiveness), and robustness (agent survivability).

# 5 Results

## 5.1 Percentage of Orders Completed

Fig. 3 shows the average percentage of completion for each type of pathfinding strategies for 5, 10 and 20 deployed agents. The graph was calculated by dividing the total number of orders dropped off by agents per simulation by the total amount of orders given to the agents. This result helps assess each agent's effectiveness to complete the assigned tasks.

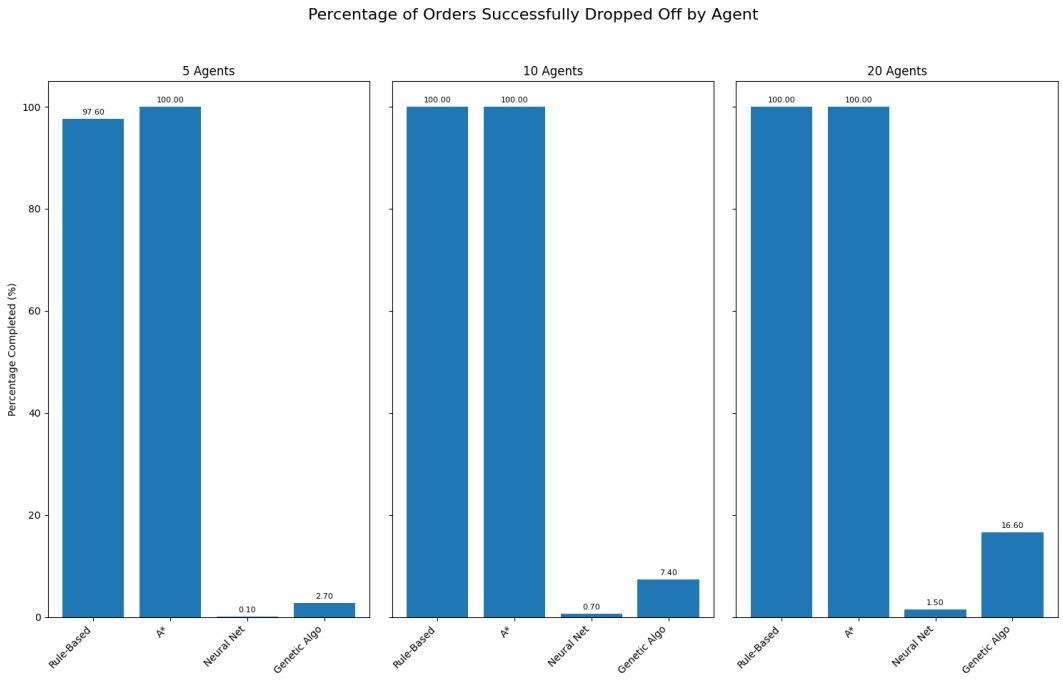


Fig. 3. Average orders completion rate per simulation for 5, 10 and 20 agents per warehouse environment..

## 5.2 Real-Time Performance

Fig. 4, 5, 6 presents the times taken for the simulation to be completed. This result will be used to understand the efficiency of each strategy.

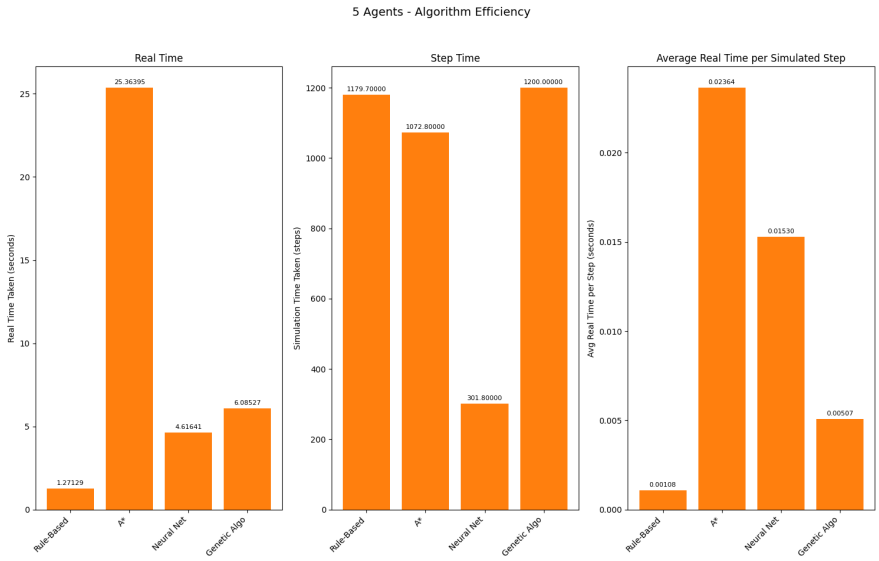


Fig. 4. Average times taken during simulations with 5 agents.

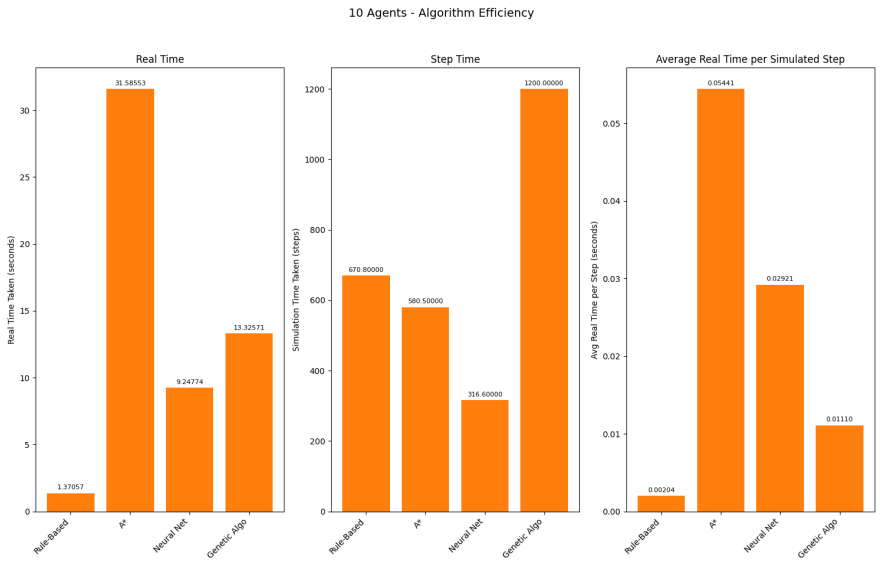


Fig. 5. Average times taken during simulations with 10 agents.

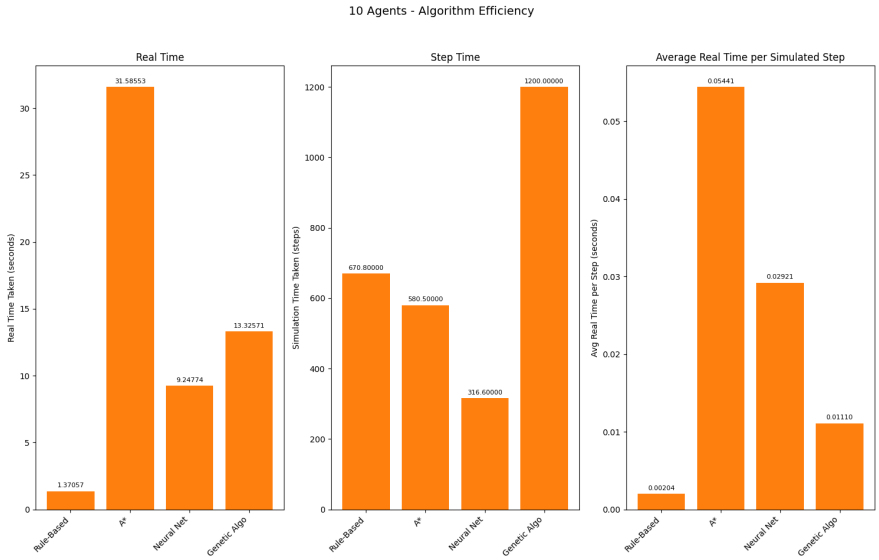


Fig. 6. Average times taken during simulations with 20 agents.

## 5.3 Number of Dead Agents

Fig. 7 shows the average amount of agent deaths by the end of each simulation. This information can give us an insight to how each pathfinding strategy avoids collisions or getting blocked by other agents.

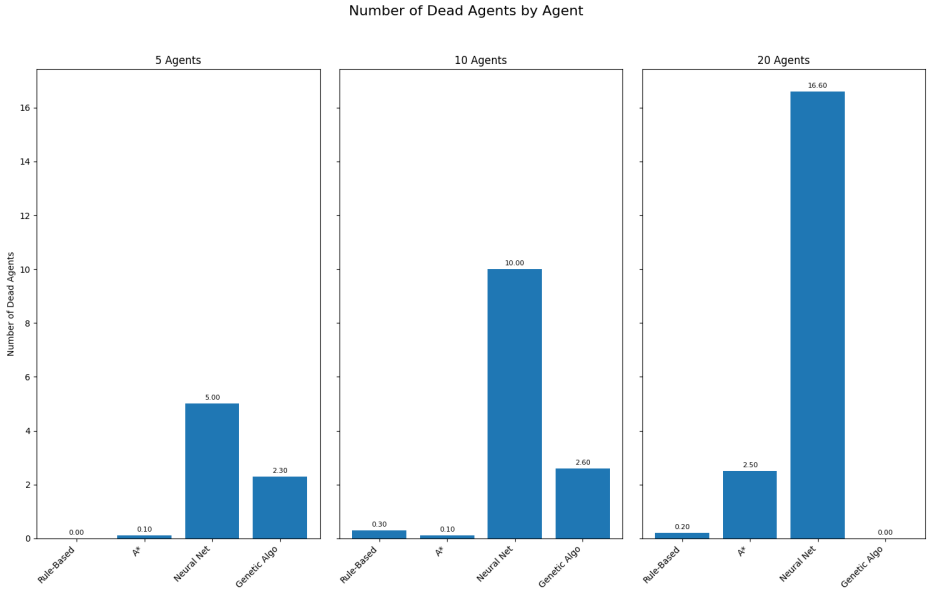


Fig. 7. Average total dead agents by the end of each simulation.

# 6 Discussion

## 6.1 Introduction

This section presents a comparative analysis between model-based and state-based (non-model) strategies for pathfinding in a warehouse setting. The results of the experiments will be discussed with respect to the three core performance metrics: percentage of order completion, real-time performance, and number of dead agents.

## 6.2 Percentage of Orders Completed

The Model-based agents shows a near perfect consistency of delivering the required items for both rule-based and A* algorithms. The result also showed no signs of issues with a crowd of up to 20 agents. This strongly highlights the importance of deterministic path planning in the reliability of task execution. As warehouses are often designed with predictable layouts, this result sheds light into how adaption of less complicated strategies may still provide a viable option, especially in settings with fixed setups.

Both state-based agents performed badly. Particularly, neural network had <2% order completion, despite its 95.37% training accuracy using a train/test dataset. It also performed worse than the genetic algorithm, which only achieved 0.8310 (83.10%) in fitness value during training. The most likely cause of this is overfitting, which can be caused by multiple factors, including not having enough variations in the generated data, or using too many/few input nodes. As the generated data used for training was also from smaller warehouse environments, this can restrict the agent's ability from learning to explore larger spaces.

However, one surprising result on the completion rate of state-based agents is that, as the number of agents doubled, their completion percentages increase at a slightly higher rate. This may lead to a possible indication of emergence behaviours. Furthermore, a more populated environment may allow for more different "input perception" than the repetitive neighbouring shelves blocks. As these state-based strategies can only rely on their instant perceptions, a repeating and structured layout may be a hindering factor to their learning ability. This reflects on the difficulty/disadvantage of state-based strategies in comparison to model-based.

## 6.3 Real-Time Performance

As both real-time and time-step values can be influenced by the simulation finishing early, their standalone values cannot be used as a metric for judging strategy performance. However, using $realTime \div stepTime = AverageDecisionTime$, the average response times can be calculated (see most-right graphs of Fig. 4, 5, 6).

In model-based agents, rule-based and A* has a significant difference between their decision response rate, with A* taking over twenty times longer than rule-based. The rule-based algorithm also stayed relatively small when agents were doubled to 10, and 20. This is due to its lightweight nature when it comes to decision making. As described in section 3.3.2, the rule-based algorithm is based on only 4 rules at its core. On the other hand, heuristics such as A* that can offer better, near-optimal solutions can take significantly more time to create a new path in dynamic settings like the experiment's environment. This confirms the strength of simple heuristics in speed critical, low complex environments.

Although the difference between state-based agents are much smaller, genetic algorithm is considerably more efficient, with all results showing response times more than halved when compared to the neural network. This may also be contributed to the architecture of the two learning strategies. With neural network having 4 hidden layers, it is possible that scaling down its complexity may lead to more comparable response time. However, this will likely also reduce performance capabilities.

Thus, in terms of real-time responsiveness, the rule-based method is the most efficient, with the genetic algorithm offering a suitable state-based alternative.

### 6.4 Number of Dead Agents

When an agent cannot travel back to its charger station in time, or became blocked off by other agents, it runs out of energy and becomes dead. Therefore, a strategy's ability to keep their agents alive portrays its capacity to plan effective path and avoid collisions.

Again, the model-based agents led in this category, with near-zero agent failures across all conditions. This means that both A* and rule-based agents are able to successfully avoid deadlocks and plan returns to charging stations effectively.

For state-based agents, the neural network struggles to find its way back to the charging station. Combined with low retrieval accuracy, this further suggests a major issue with pathfinding using only its states. Therefore, it can be suggested that neural network does not suit a state-based pathfinding environment.

The Genetic Algorithm showed intermediate results. It outperformed the neural network in every condition, and had minimal dead agents even in high-density tests. This suggests a possible emergent behaviour and collision avoidance, and should be investigated further, as it can become an important feature in a more complex environment.

### 6.5 Overall

The outcome of the experiments show a strong affirmation for model-based heuristics for pathfinding in generalised warehouse environment. With all but one result showing top score, the simplicity in architecture and capability in pathfinding makes them the optimal solution.

On the other hand, the results also showed how state-based agents may require more robust training or more environmental input of their states to ensure capabilities under scaled/dynamic environments.

## 7 Conclusion

### 7.1 Summary

This study conducted a comparative investigation on two core perception paradigms, model-based heuristics and state-based learning methods, represented by four intelligent agent strategies: rule-based, A*, neural network, and genetic algorithm. These agents were deployed in generalised warehouse environments, and evaluated based on three key performance metrics: percentage of order completion, real-time computational efficiency, and number of agent failures.

The results show a clear dominance of model-based heuristics, particularly the rule-based, across all tested scenarios. These agents not only demonstrated near-perfect order completion rates, but also maintained low decision-making times and minimal agent failures, even as the number of agents scaled up. Their simplicity, determinism, and awareness of the full environment enabled robust, scalable, and effective navigation. This is especially useful in structured layouts typical of warehouse operations.

In contrast, state-based agents underperformed significantly. Despite high evaluation accuracy during offline training, neural network struggled during the experiment, resulting in low task completion, frequent failures, and inconsistent behaviours. This highlights a fundamental limitation of learning models, as their performance in unseen environments is heavily dependent on the quality and diversity of training data, as well as the suitability of their perception inputs.

The genetic algorithm, though also a state-based strategy, showed more promise. It outperformed the neural network in both task success and survivability, and offered a better real-time efficiency. Its evolutionary nature suggests potential for emergent behaviours, especially in more complex or stochastic environments. There may be merits to further study this strategy as a lightweight adaptive strategy.

## 7.2 Limitations & Future Directions

While this project provides meaningful insights, several limitations must be acknowledged. The warehouse environments used were highly structured, regular grids, which do not fully capture the conditions of real-world logistics settings. Furthermore, state-based agents, performance was closely tied to training quality. Their behaviour is sensitive to training diversity, input design, and architecture. Time constraints had hugely limited the scope to test and refine this.

Future research should focus on expanding the diversity of training environments to help learning agents generalise to new layouts. Enhancing state perception, such as through incorporating path history or distance to nearby agents may also improve decision quality. Hybrid approaches, combining global model-based planning with local adaptive learning, can also represent a promising direction. Finally, evaluating agents in dynamic, irregular environments with task switching and inter-agent dependencies would further validate their real-world applicability.

## 7.3 Reflection

In conclusion, the results of this project suggest that model-based heuristics remain highly effective for structured, high-reliability logistics environments, while state-based learning agents, though promising, require significant development, particularly in training, perception, and adaptability. Despite their current limitations, learning agents could ultimately enable more flexible, decentralised coordination in complex systems where deterministic planning may fall short.

This project highlights the comparison between the two perception paradigms. Its results gave suggestion on hybrid systems and lays the foundation for future work bridging the gap between robust traditional planners and adaptive intelligent learning agents.

## References

[1] D. Silver, "Cooperative pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, no. 1, pp. 117–122, 2021. [Online]. Available: https://doi.org/10.1609/aiide.v1i1.18726

[2] M. Kordos, J. Boryczko, M. Blachnik, and S. Golak, "Optimization of warehouse operations with genetic algorithms," *Applied Sciences*, vol. 10, no. 14, 2020. [Online]. Available: https://doi.org/10.3390/app10144817

[3] Y. Wang, T. Duhan, J. Li, and G. Sartoretti, "Lns2+rl: Combining multi-agent reinforcement learning with large neighborhood search in multi-agent path finding," 2025. [Online]. Available: https://arxiv.org/abs/2405.17794

[4] Z. Liu, B. Chen, H. Zhou, G. S. Koushik, M. Hebert, and D. Zhao, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11 748–11 754, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2007.15724

## A Grid Population Algorithm

**Algorithm 1** Initialise Grid with Shelves, Chargers, and Drop-offs

**for** $i = 0$ to $grid\_size[0] - 1$ **do**
  **for** $j = 0$ to $grid\_size[1] - 1$ **do**
    **if** $1 < i < grid\_size[0] - 2$ **then** ▷ Setting Middle Section Shelves
      **if** $j = 0$ **then**
        $grid[i, j] \leftarrow shelf\_id$
        $shelf\_id \leftarrow shelf\_id + 1$
      **else if** $grid\_size[1] - j > 1$ **then**
        **if** $j \bmod 3 \neq 1$ and $i \bmod 7 \neq 1$ and $i \bmod 7 \neq 0$ **then**
          $grid[i, j] \leftarrow shelf\_id$
          $shelf\_id \leftarrow shelf\_id + 1$
        **end if**
      **else if** $grid\_size[1] \bmod 3 = 0$ **then**
        $grid[i, j] \leftarrow shelf\_id$
        $shelf\_id \leftarrow shelf\_id + 1$
      **end if**
    **else if** $i = 0$ **then** ▷ Setting First Row
      **if** $num\_charger > j$ **then**
        $grid[i, j] \leftarrow 2$ ▷ Setting Charging Station
      **else**
        $grid[i, j] \leftarrow shelf\_id$
        $shelf\_id \leftarrow shelf\_id + 1$
      **end if**
    **else if** $i = grid\_size[0] - 1$ **then** ▷ Setting Final Row
      **if** $grid\_size[1] - 1 - num\_dropoff < j$ **then**
        $grid[i, j] \leftarrow 3$ ▷ Setting Drop-off
      **else**
        $grid[i, j] \leftarrow shelf\_id$
        $shelf\_id \leftarrow shelf\_id + 1$
      **end if**
    **end if**
  **end for**
**end for**