Scholars Journal of Engineering and Technology
Abbreviated Key Title: Sch J Eng Tech
ISSN 2347-9523 (Print) | ISSN 2321-435X (Online)
Journal homepage: https://saspublishers.com

Optimising Warehouse Navigation: A Novel Two-Dimensional Grid Model for Robot Path Planning in Warehouse Logistics

Paarth Sonkiya[1*]

[1]Student Researcher, Neerja Modi School, Jaipur 302016

DOI: https://doi.org/10.36347/sjet.2024.v12i10.003 | **Received:** 11.09.2024 | **Accepted:** 18.10.2024 | **Published:** 21.10.2024

***Corresponding author:** Paarth Sonkiya
Student Researcher, Neerja Modi School, Jaipur 302016

Abstract **Review Article**

In practical warehouse scenarios, route optimization is an important factor due to its large impact on the cost and time efficiency of a warehouse. This directly affects the overall productivity of a warehouse. Many algorithms have been developed to address this issue- the most commonly used in the industry include the A* algorithm and Dijkstra's algorithm. While most provide appropriate usage in static ware house environments, many often fall short in dynamic warehouses, where they are unable to efficiently adapt to changing layouts and obstacles without compromising on time and cost efficiency. This study proposes a lattice-based two-dimensional algorithm designed to navigate warehouses while also avoiding obstacles efficiently. Employing this algorithm can result in substantial cost reductions, as it optimizes travel distances and resource allocation. Moreover, the algorithm enhances the time efficiency significantly by reducing order fulfillment. This research offers a practical solution to a persistent challenge in modern warehouse logistics. The effectiveness of the proposed algorithm suggests its potential to revolutionize the industry's approach to route optimization.

Keywords: Lattice Paths, Route optimization, Warehouse, AGVs.

1. INTRODUCTION

With the rise of the internet, there has been a rapid surge in the e-commerce sector. Online shopping in particular has grown exponentially due to factors like wider product selection, convenience and better pricing. This growth has increased the demand for more efficient and productive warehouse operations to meet the growing customer expectations for fast and reliable delivery. With this, new entrepreneurs have entered the industry, making warehouse logistics a highly competitive sector. Warehouses face constant pressure to optimize processes and reduce costs to remain profitable. Because of this rise in competitiveness, newer problems came to being. One of the most critical factors affecting the overall productivity of a warehouse is the storage and retrieval of goods. Many traditional methods are slow, inefficient, and prone to a lot of errors. Most modern warehouses now address these challenges by utilizing automated guided vehicles (AGVs) to streamline storage and retrieval operations. The integration of AGVs and warehouse robots has significantly enhanced transportation speed and precision. These automated systems can improve efficiency, reduce labor costs, and minimize picking errors. Therefore, optimizing path planning for these automated robots can greatly affect the operational efficiency of warehouses. However, navigating warehouse environments optimally remains a key challenge as they introduce complexities that traditional pathfinding algorithms struggle to handle. This can include obstacles- such as unpredictable inventory placement, forklifts and personnel-computational efficiency and time taken. Existing pathfinding algorithms often fall short in these dynamic environments. This paper proposes a novel two-dimensional grid model and an optimized algorithm specifically designed to address these challenges and enable efficient robot navigation in dynamic warehouses.

Researchers over the past decade have developed several methodologies to address this persisting problem. The most popular algorithms used today include the Dijkstra's and A* algorithm. Dijkstra's algorithm [1] works by transforming the warehouse layout into a graph. Each aisle intersection becomes a node, and paths between them become edges with weights representing travel time or distance. The algorithm then iteratively explores these connections, prioritizing unvisited nodes with the lowest total travel distance from the starting point. This efficiently determines the shortest path for a robot or picker to reach any destination within the warehouse. The algorithm guarantees finding the optimal path but can be computationally expensive for large and complex warehouses. Additionally, it struggles to adapt to dynamic changes like moving obstacles [5]. The A* search algorithm builds upon Dijkstra's algorithm by incorporating a heuristic function to prioritize exploration towards the goal. A heuristic function is an informed estimate of the cost (distance, time, etc.) to reach the goal from a particular point in the environment. These estimates help the algorithms prioritize exploring more promising paths that are likely to lead to the goal faster. For example, in a two dimensional grid representing a warehouse, a common heuristic function might be the Manhattan distance between a cell and the goal location [4]. This estimates the minimum number of horizontal and vertical steps required to reach the goal, ignoring obstacles. Heuristics play a crucial role in path selection by directing the search towards more efficient paths. Algorithms like A* use the total cost (combination of movement cost and heuristic estimate) to evaluate neighboring cells and prioritize those with a lower total cost. This strategy helps them focus their search on promising areas and avoid exploring irrelevant parts of the environment. Therefore, this is similar to the method the proposed algorithm builds up on. This leads to faster path finding, especially in complex environments. However, the traditional A* algorithm can still be computationally expensive for very large warehouses [3].

There have been many other approaches to this problem as well. For instance, the research by Yang *et al*., [6] introduces the concept of the largest convex polygon (LCP) to illustrate the shortest path to traverse all goods locations in an ideal condition. This involves getting an initial node, establishing a Cartesian coordinate system, and then adding nodes based on their positions relative to the initial node. This method could potentially improve vehicular navigation due to its shown reduction in time complexity and path length, but the method does not consider the real-world complexities which impacts its applicability in practical scenarios. In their study, Roodbergen *et al*., [2] propose a branch-and-bound algorithm adapted from the Travelling Salesman Problem (TSP) to identify the shortest path within a parallel aisle warehouse. This approach is specifically designed for warehouses with crossovers at both the ends and midpoints of aisles. The authors compare their algorithm's performance to established routing heuristics like S-shape, aisle-by aisle, largest gap, and a combined method. Additionally, they explore the impact of warehouse layout on efficiency, demonstrating that incorporating cross aisles can significantly reduce travel time during picking operations by offering more direct routes. However, the paper did not explore the impact of non-random storage assignment rules on heuristic performance, which could be crucial in real warehouse settings.

Most existing algorithms exhibit limitations in scalability and computational time as warehouse complex ity increases. Similarly, path planning methods for warehouse robots often struggle with slow convergence and neglect downstream impacts. These challenges highlight the need for advanced AGV scheduling and path planning algorithms that can adapt to dynamic environments and scale efficiently. Therefore, this research focuses on employing lattice pathfinding algorithms to optimise route planning in warehouse logistics, with a specific emphasis on effective obstacle avoidance. The objectives include developing and optimising a specialised lattice pathfinding algorithm, evaluating its performance against traditional methods, and providing practical recommendations for real-world warehouse navigation challenges.

2. Problem Description

This research specifically focuses on block stocking warehouses, also known as pile-type warehouses. Figure 1 shows a simplified model of the warehouse. The red squares represent the area that is occupied by a single block and the blue square shows the area an AGV can go to for the retrieval of goods.

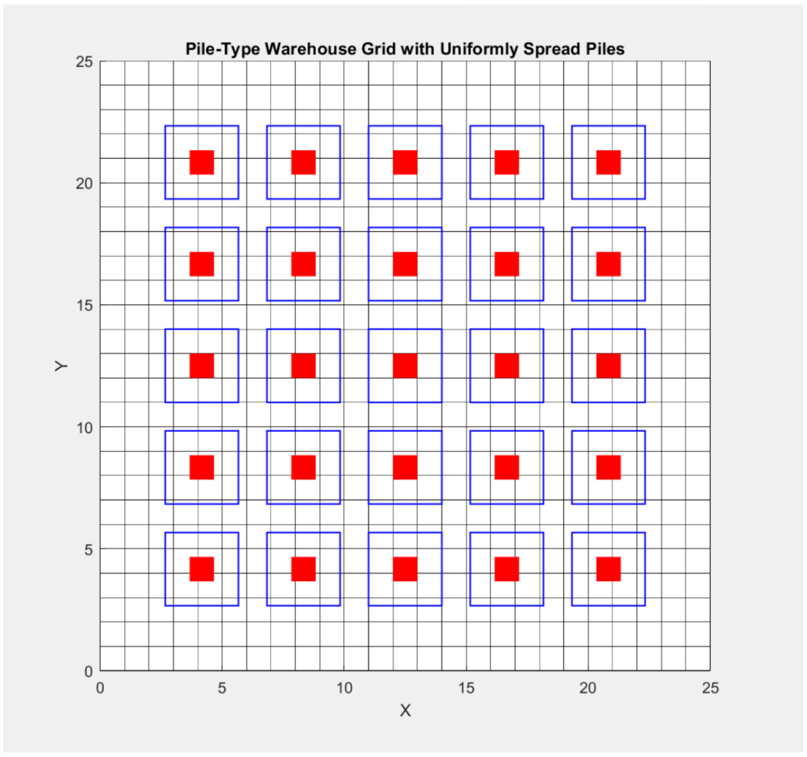


Figure 1: A Simple representation of a block stocking warehouse

For simplicity, a translated version of this warehouse image is considered in this paper, as shown in Figure 2. The intersection of the grid lines or nodes represent each block and the lines represent the path a robot can take. The advantage of using a simplified lattice path model allows the easy facilitation of pathfinding algorithms. This allows the algorithms to easily explore the grid, evaluating possible movement options between connected cells, and ultimately identify the optimal path for the robot to navigate within the warehouse.

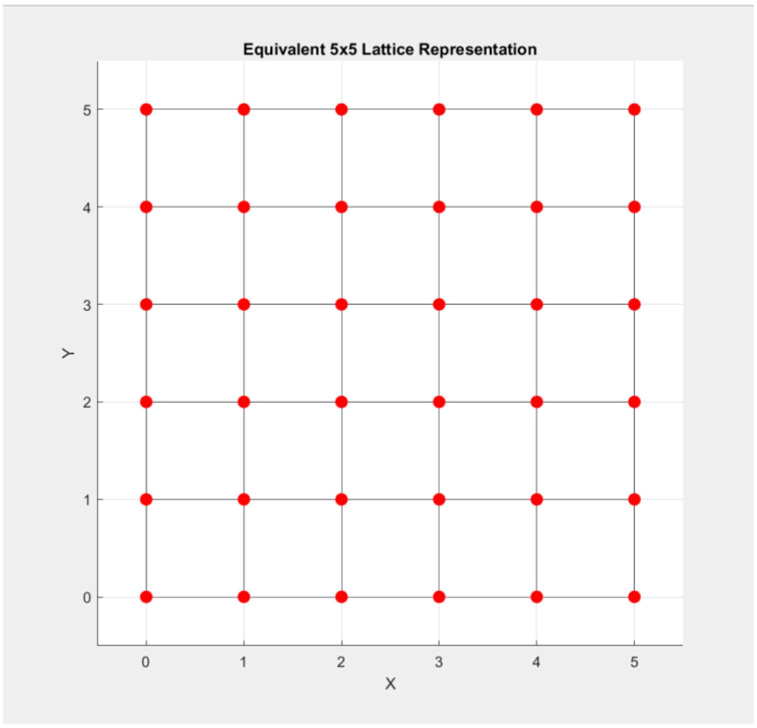


Figure 2: The equivalent simplified model

Figure 3 portrays a model of a real-life warehouse.

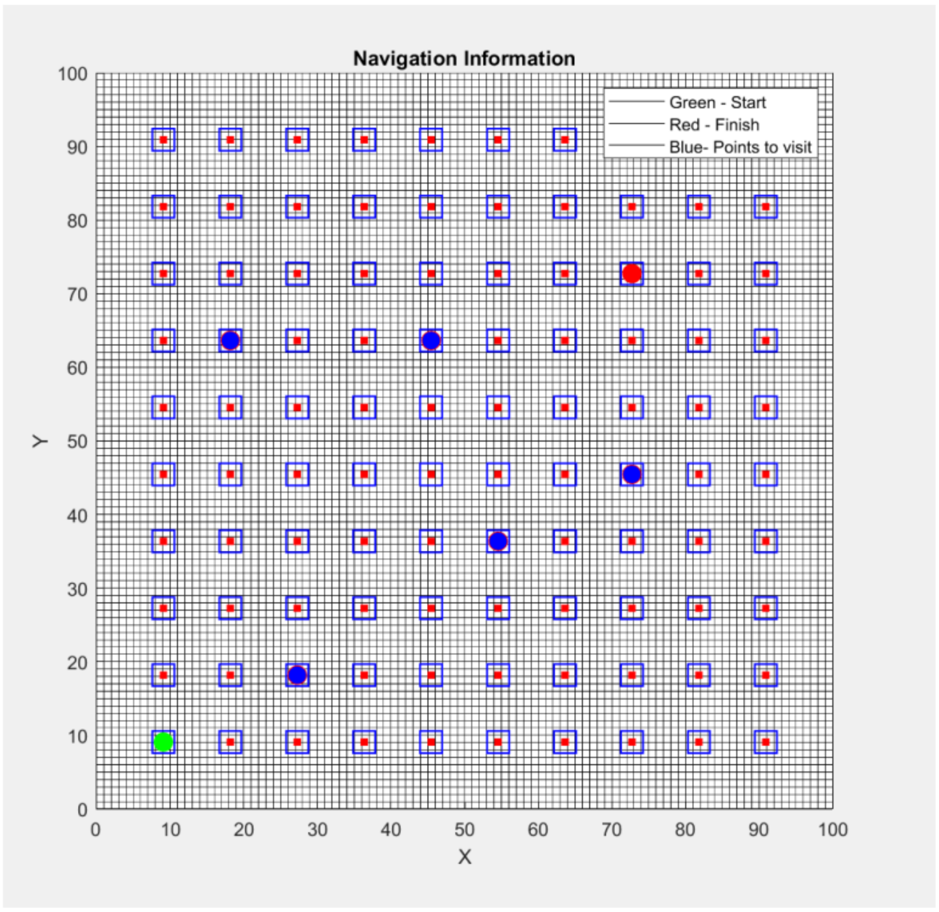


Figure 3: Model of a realistic block warehouse with Navigation information

This model depicts a more expansive and complex environment, similar to a modern warehouse, where there are hundreds of blocks and isles. The figure visually depicts the path a robot needs to take for the retrieval of goods within the warehouse environment. This illustration provides the robot's navigation process and the key elements involved.

The green dot indicates the starting point for the robot's journey. This can be the robot's charging station or a designated starting location within the warehouse. The blue dots represent the specific locations, or blocks, within the warehouse that the robot needs to visit to retrieve goods. These blocks could correspond to individual storage locations, picking stations, or designated areas where specific items are stored. The sequence and order of visiting these blue dots therefore determine the efficiency of the overall retrieval process. The red dot signifies the final destination for the robot after completing its goods retrieval task. This point could be a designated drop-off location or the robot's charging station, depending on the specific workflow. Figure 4 translates this large-scale warehouse model into a corresponding simplified lattice path model. Similar to the previous lattice model, this representation abstracts the physical layout into a two-dimensional grid. This model incorporates a larger grid size to accommodate the increased complexity of the real world warehouse. Translating real-world warehouse layouts into simplified lattice path models is crucial for pathfinding algorithms for both simplicity and effectiveness. These algorithms operate more effectively within the grid structure, allowing them to determine optimal paths for robots navigating the actual warehouse environment while reducing the complexity. The figure also represents obstacles in the path, where the solid black squares denote the obstacles through which robots cannot pass. This scenario emphasises the challenges faced by robots performing tasks like navigation and path planning within warehouses.

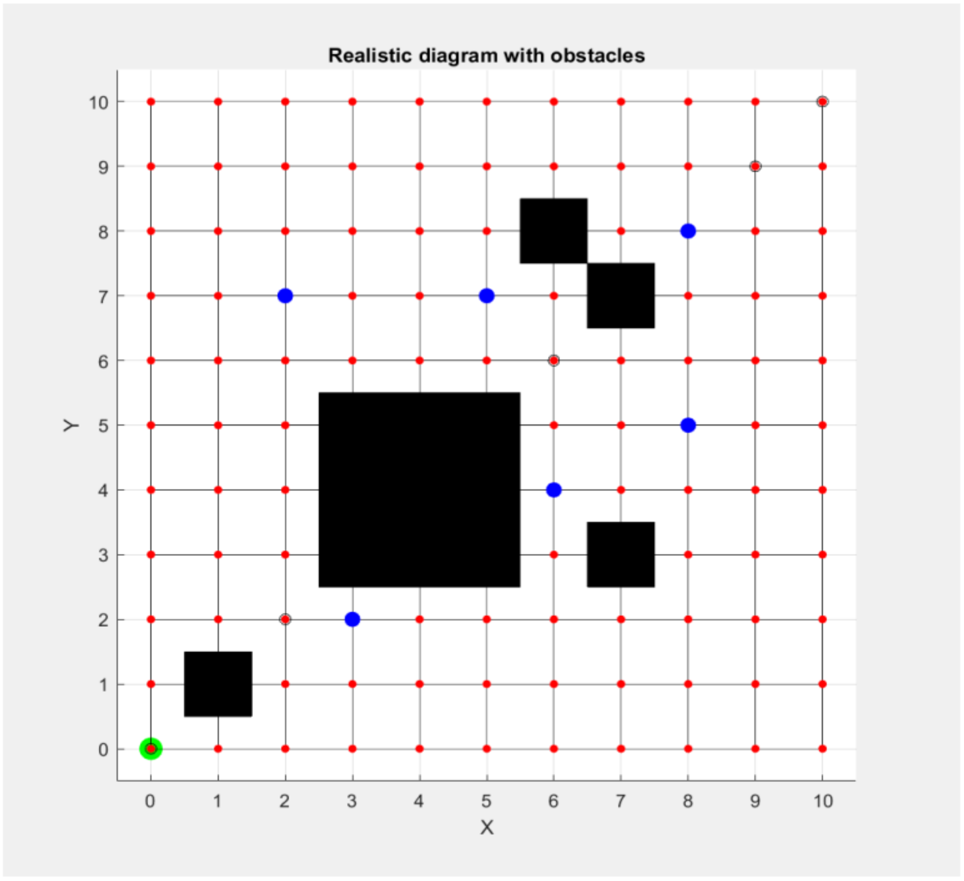


Figure 4: Simplified model of a realistic block warehouse with obstacles

The previous models discussed provide a foundational understanding of block warehouse layouts. How ever, real-world warehouses present a more complex environment filled with various obstacles that robots must navigate around. These obstacles pose significant challenges for robot path planning algorithms. The obstacles can be static or dynamic. Static obstacles are permanent fixtures within the warehouse that robots cannot move through, which can include support pillars or walls, inventory storage racks and shelves and also designated no-go zones due to maintenance, safety reasons, or specific operations that require human intervention. Dynamic obstacles include elements that can change within the warehouse environment, creat ing temporary blockages for robots. This reseacrh does not address actively moving obstacles but obstacles that may change positions but remain stationary during run-time. This research proposes a novel algorithm that addresses these challenges by modifying the heuristic function while taking effective obstacle avoidance into consideration.

3. Algorithm Design

This section describes our pathfinding algorithm designed for robots navigating a grid-based environment with obstacles. The algorithm modifies the A* search, a well-known technique for finding optimal paths in graphs or grids. A* search balances exploration of the search space with an informed prioritisation of promising paths. Our specific implementation focuses on finding the shortest path for a robot visiting multiple designated points within the warehouse. The problem can be formulated within the framework of a graph, where the warehouse layout is represented as a directed graph $G = (V, E, D)$, comprising vertices V , edges E, and distances D.

Consider a path path P as $P_1, P_2, ..., P_n$ as shown in Figure 5, where each P_i denotes a coordinate on the x-y plane or the two-dimensional grid.

Vertices in the graph correspond to distinct locations within the warehouse, such as aisles, racks, inter sections, and loading docks. Formally, $V = \{v_1, v_2, ..., v_n\}$, where n denotes the total number of vertices in the warehouse layout. Edges represent permissible paths or connections between vertices, denoting feasible routes that can be traversed by the warehouse vehicles or personnel. For any pair of vertices v_i, v_j in V , if there exists a direct path from v_i to v_j , then an edge e_{ij} is present in E. Mathematically, $E \subseteq V \times V$. The variable d represents the Manhattan distance between two nodes P_1=(x_1, y_1) and P_2=(x_2, y_2), which is simply given by:

$$d(P_1, P_2) = |x_2 - x_1| + |y_2 - y_1|$$

This dentoes the length or cost associated with traversing an edge in the warehouse graph. For any edge e_{ij} in E, the distance d_{ij} signifies the distance or cost to travel from vertex v_i to vertex v_j . Formally, $D = \{d_{ij}\}$,

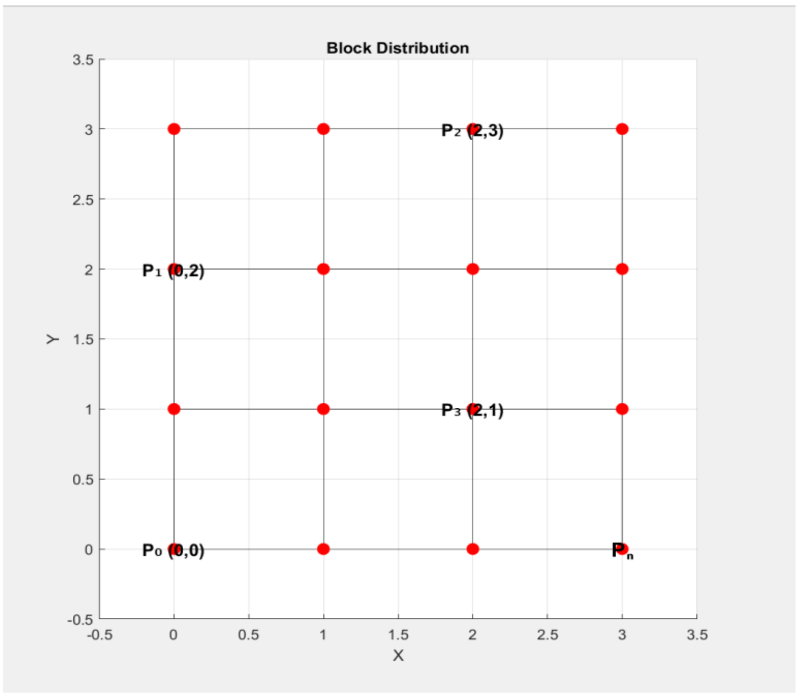


Figure 5: Simplified model of a realistic block warehouse with obstacles

Where d_{ij} denotes the distance between vertices v_i and v_j. Given this graph representation of the warehouse layout, the optimization problem can be formally defined as finding the most efficient path or sequence of vertices to navigate from a designated source location to a target destination, subject to various constraints and objectives.

Mathematical Formulation

Let f denote the objective function, which quantifies the efficiency metric to be optimized. This metric may vary depending on the specific objectives of the warehouse management system. The objective function f can be expressed as a function of the path traversed through the warehouse graph, represented by a sequence of vertices, where $f: V \rightarrow \mathrm{R}$, where R represents the set of real numbers. The optimization problem may be subject to various constraints imposed by the warehouse environment, vehicle characteristics, safety regulations, and operational requirements. These constraints may include limitations on vehicle speed, maximum load capacity, aisle width, aisle congestion, and restricted access zones. The total cost associated with traversing a given path P in the warehouse graph can be expressed as the sum of distances between consecutive vertices along the path. Mathematically, the total cost $C(P)$ can be represented as:

$$C(P) = {}^{k}\mathrm{X}^{-1}\ i=1\ x_{i,i+1} \cdot d(v_i, v_{i+1})$$

Where k represents the number of vertices in the path P, and $d_{i,i+1}$ denotes the distance between the i-th and $(i+1)$-th vertices along the path. The variable $x_{i,i+1}$ is a binary variable: it takes the value 1 if the edge between vertices v_i and v_{i+j} is included in the graph, representing that the point is part of the path P; otherwise, it takes the value 0.

The core principle behind the algorithm lies in the A* search algorithm, a well-established method for optimal pathfinding in graphs and grids. In our specific implementation, the algorithm aims to find the shortest path for a robot that needs to visit multiple designated points sequentially. The algorithm maintains a priority queue (heap) data structure. This queue stores potential paths, each represented as a tuple containing the total cost incurred so far (distance travelled by the robot), the current cell coordinates of the robot, and the path history, which tracks the sequence of cells visited to reach the current cell. The algorithm then iteratively explores the neighbours of the cell with the lowest total cost according to the priority queue. This prioritisation ensures that the algorithm focuses on paths that are most likely to lead to the goal efficiently.

To further guide exploration, the algorithm employs a heuristic function. This function estimates the remaining distance from the current cell to the final destination. In our case, we utilise the Manhattan distance heuristic. As stated above, it calculates the absolute difference in x and y coordinates between the current cell and the final destination, providing a simple and efficient estimate of the remaining distance. The estimated distance is then added to the actual cost to create the total cost for each path. This combined value guides the prioritisation within the heap, favouring paths that are geographically closer to the goal. The two key data structures utilised by the algorithm are the priority queue and the visited set structure. The priority queue prioritises elements based on a key value. In our implementation, the key value represents the total cost of a path, which is the sum of the actual cost traversed so far and the estimated remaining distance calculated by the heuristic function. The heap efficiently retrieves the cell with the lowest total cost for exploration at each step. This ensures that the algorithm explores promising paths with potentially lower overall costs first, leading to a faster discovery of the optimal path. This Visited Set stores the coordinates of all cells that have already been explored by the algorithm. Including a cell's coordinates in the visited set after it has been explored ensures the algorithm doesn't revisit previously explored areas. This prevents redundant exploration and focuses the search towards unexplored territories within the warehouse environment.

The algorithm incorporates obstacle detection to ensure the robot navigates only on valid paths. Before considering a neighbouring cell for exploration, the algorithm checks two conditions - the cell must be within the defined grid boundaries and the cell's value in the grid representation must not be 1, which signifies an obstacle. By adhering to these conditions, the algorithm ensures that the robot only explores and utilises valid paths that are free of obstacles.

Once the robot reaches a designated point, the algorithm needs to reconstruct the complete path taken so far. This path reconstruction is done by the information stored within the priority queue. Each cell in the queue stores its parent cell in the path, indicating the cell from which it was explored. By backtracking through this parent-child relationship stored in the queue, the algorithm can reconstruct the complete path taken by the robot from the starting point to the current designated point. This backtracking process continues for each designated point the robot needs to visit. The algorithm finds the next closest unvisited point using the heuristic function and repeats the exploration process until all designated points are visited. By accumulating the reconstructed paths for each point, the algorithm obtains the final complete path for the entire robot navigation task. Figure 6 below shows a simplified flowchart indicating the basic principles of the algorithm.

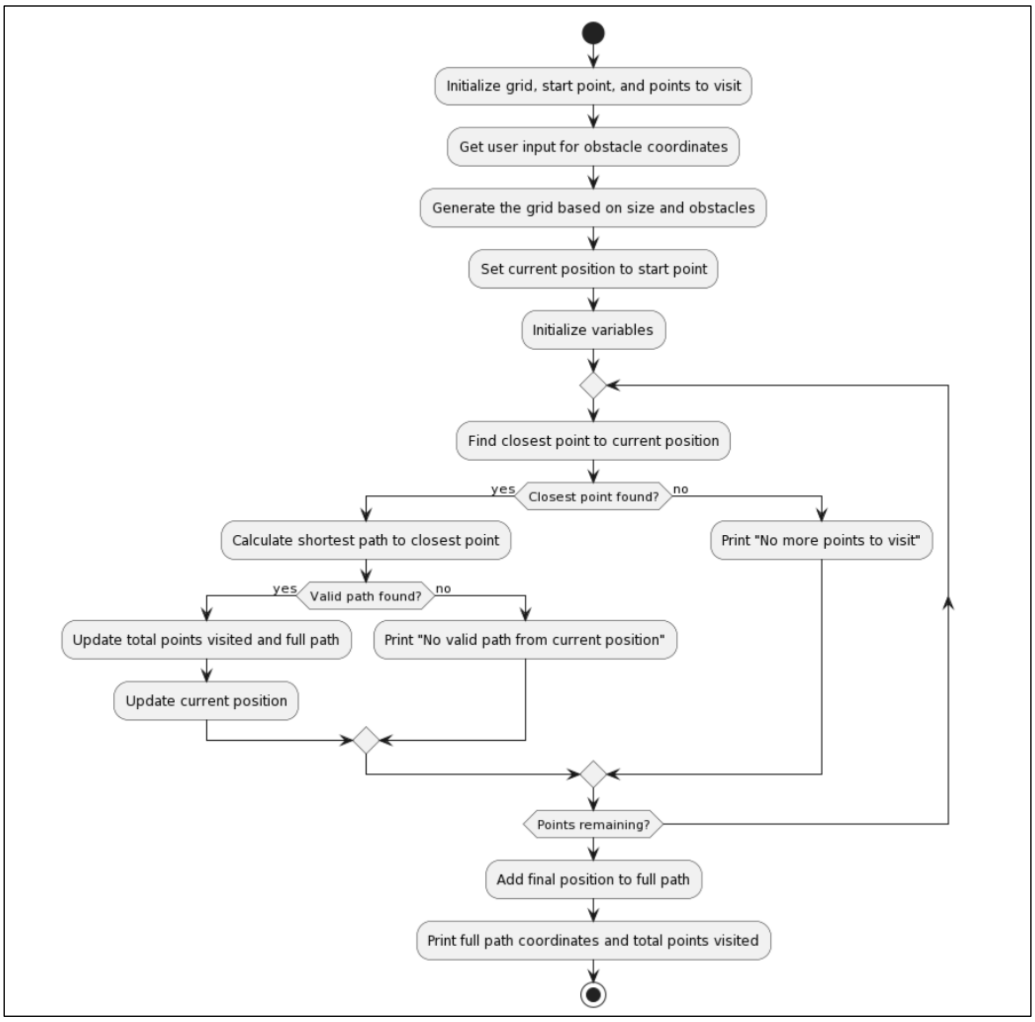

Initialize grid, start point, and points to visit
Get user input for obstacle coordinates
Generate the grid based on size and obstacles
Set current position to start point
Initialize variables
Find closest point to current position
Closest point found?
yes
no
Calculate shortest path to closest point
Print "No more points to visit"
Valid path found?
yes
no
Update total points visited and full path
Print "No valid path from current position"
Update current position
Points remaining?
Add final position to full path
Print full path coordinates and total points visited

Figure 6: A basic understanding of the key principles of the algorithm

4. Comparative Analysis

This section presents a comparative analysis of our proposed algorithm with Dijkstra's algorithm, a popular pathfinding algorithm widely used today. Our aim is to evaluate the strengths and weaknesses of each approach in the context of robot pathfinding within warehouse environments with obstacles. This comparison will discuss the suitability of our algorithm for practical applications in warehouse navigation tasks.

4.1 Time Complexity

This section analyses the time complexity of the proposed algorithm and Dijkstra's algorithm. Time com plexity refers to the amount of time an algorithm takes to execute as the size of the input grows. In the context of robot pathfinding within a warehouse environment, the main factors affecting the input size are - the number of grid cells (V), the number of obstacles and the number of designated points (P).

4.1.1 Proposed Algorithm

The time complexity is analysed by taking the average and the worst case scenarios into consideration.

Average Case In the average case scenario, where the heuristic function provides a good estimate of the remaining distance, the time complexity of the proposed algorithm is expected to be:

$$O((\log b) * V)$$

Where log b represents the repeated operations on the priority queue (heap) used for exploration, b represents the branching factor, which is the average number of neighbours a cell has in the grid. The logarithmic term reflects the efficient retrieval and update operations within the heap data structure and V represents the total number of grid cells.

Worst Case In the worst case scenario, where the heuristic function provides poor estimates, the time complexity of the algorithm can approach:

$$O(W * \log b * V)$$

Here, W represents a factor dependent on the specific grid layout, obstacle distribution, and the starting and goal locations. This factor accounts for the additional exploration required due to the heuristic's inef ficiency in the worst case. However, the logarithmic term (log b) due to the heap operations and the linear term (V) representing the total number of cells are likely to dominate the complexity even in the worst case.

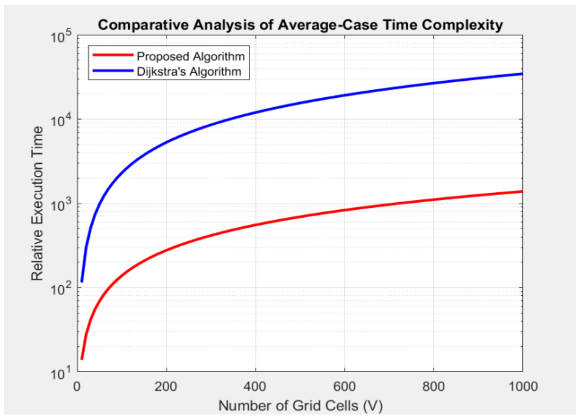
4.1.2 Dijkstra's Algorithm

Dijkstra's algorithm has a time complexity of:

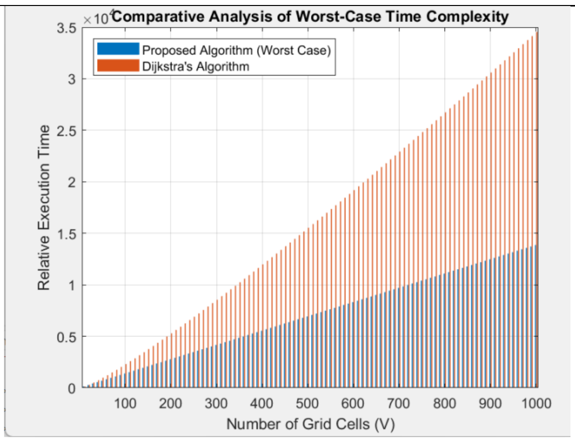
$$O(V + E * \log V)$$

Where, E Represents the total number of edges in the grid. In a warehouse environment, this translates to the number of valid connections between neighbouring cells.

5. RESULTS



(a) The Time Complexity analysis of the proposed algorithm and Diksarta'a algorithm



(b) The Time Complexity analysis shown on a line graph

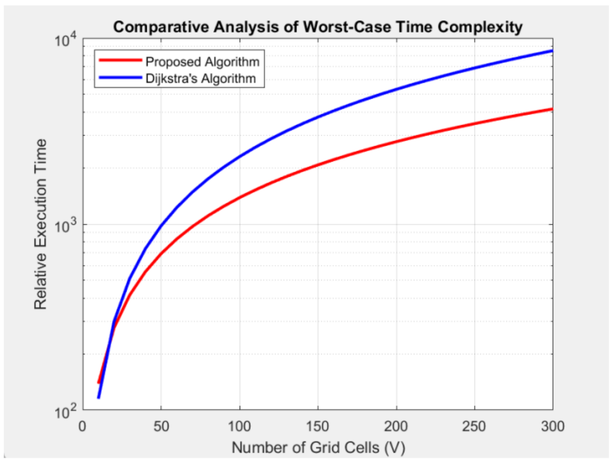
Figure 7: Average Case Time Complexity Comparison

In the average case, the proposed algorithm outperforms Dijkstra's algorithm as shown in Figures 7 a. and b. due to the logarithmic term (log b) in its complexity. This logarithmic term stems from the efficient use of a priority queue (heap) data structure for exploration. The heap operations, like inserting and retrieving elements, take logarithmic time with respect to the number of elements in the heap. In the context of our algorithm, the number of elements in the heap corresponds to the number of promising paths being explored. As the warehouse environment (grid size) grows, the number of paths to explore increases. However, due to the logarithmic nature of heap operations, the time spent managing the exploration queue scales proportionally less significantly compared to Dijkstra's algorithm.

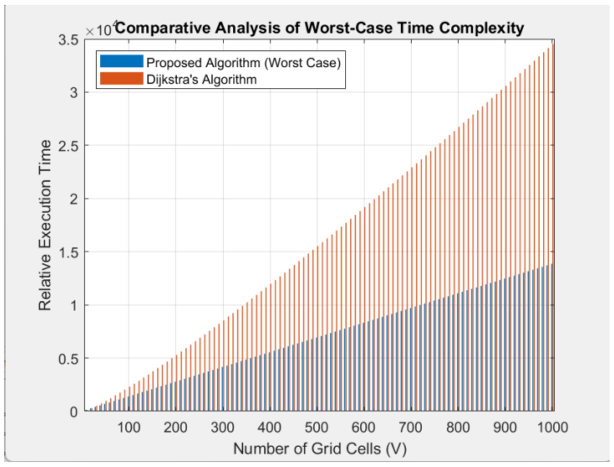
Dijkstra's algorithm, on the other hand, has a time complexity that includes a term linear in the number of grid cells (V) and edges (E). As the warehouse environment becomes larger, the number of cells and edges increases proportionally. This translates to a more significant growth in the execution time for Dijkstra's algorithm compared to the proposed algorithm in the average case. Therefore, the logarithmic term in the proposed algorithm's complexity signifies that the exploration process scales more efficiently with the size of the warehouse grid compared to Dijkstra's algorithm. This efficiency advantage becomes more pronounced for larger warehouses, making the algorithm a more suitable choice in such scenarios.

While both algorithms can theoretically exhibit exponential dependence in the worst case, there are many potential advantages for the proposed algorithm in handling complex warehouse environments. Dijkstra's algorithm's dependence on the total number of grid cells (V) and edges (E) can lead to significant exploration overhead, especially in scenarios with dense obstacles or unfavourable start and goal locations. In such cases, the exhaustive exploration strategy of Dijkstra's algorithm might struggle to efficiently navigate the environment.

On the other hand, our algorithm's complexity includes a logarithmic term (log b) that stems from the efficient management of the exploration queue using a heap data structure. This logarithmic term helps to mitigate the impact of a growing number of potential paths on the processing time. Additionally, the proposed algorithm's inherent prioritisation mechanism, guided by the heuristic function restricts the exploration to a more focused search space around promising paths. This focus can significantly reduce the number of irrelevant paths explored compared to Dijkstra's exhaustive approach, potentially preventing the worst-case complexity from becoming extremely slow in complex warehouse environments. The results of the above can be seen in Figures 8 a. and b. below.



(a) The Time Complexity analysis of the pro posed algorithm and Diksarta'a algorithm



(b) The Time Complexity analysis shown on a bar graph

Figure 8: Worst Case Time Complexity Comparison

Overall, by considering both average and worst-case scenarios, this comparative analysis highlights the potential of the proposed algorithm for efficient robot pathfinding in warehouse environments. The logarithmic term in its complexity signifies efficient exploration management using a priority queue. This structure allows our algorithm to scale more efficiently as the warehouse size increases, making it a preferable choice for real-world scenarios. While there might be a slight trade-off for very small warehouses, the overall analysis suggests that our proposed algorithm offers a significant time complexity advantage for larger and more realistic warehouse environments. The algorithm was developed with Python 3.12 and all simulations were done with MATLAB R2024a.

6. CONCLUSION

In this research, a critical aspect of our analysis was the time complexity of the proposed algorithm. Optimizing warehouse operations requires efficient navigation with the ability for robots to complete tasks in a timely manner. By comparing the time complexity of our algorithm to Dijkstra's algorithm, we were able to demonstrate the efficiency gains achieved by our method, particularly in scenarios with larger warehouse layouts. This efficiency translates to faster retrieval and storage times, ultimately contributing to increased warehouse throughput. While this research focused on time complexity, future work can explore the energy efficiency of the proposed algorithm. Investigating the relationship between pathfinding strategies and robot energy consumption could pave the way for even more optimized warehouse operations that minimize energy use without compromising efficiency. Morevover, We highlighted the limitations of traditional pathfinding algorithms in these warhouse settings, particularly not taking obstacle avoidance into consideration and having higher time complexity. Further research can explore the integration of machine learning techniques to continuously learn and improve the algorithm's performance in dynamic environments. Additionally, in vestigating methods for collaborative path planning between multiple robots operating within the warehouse could further optimize overall throughput and efficiency. By addressing the challenges of dynamic warehouse navigation, this research contributes to the development of more efficient and reliable automated systems for modern warehouses. This will play a vital role in supporting the ever-growing demands of the e-commerce sector and ensuring the smooth flow of goods within the supply chain.

REFERENCES

1. Liu, X., Cao, J., Yang, Y., & Jiang, S. (2018). CPS-Based Smart Warehouse for Industry 4.0: A Survey of the Underlying Technologies. *Computers, 7*(1), 13.
2. Roodbergen, K. J., & De Koster, R. (2001). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research, 39*(9), 1865-1883.
3. Sanei, O., Nasiri, V., Marjani, M. R., & Moattar Husseini, S. M. (2011). A heuristic algorithm for the warehouse space assignment problem considering operational constraints: with application in a case study. Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management, Kuala Lumpur, Malaysia, January 22-24, 2011.
4. Shen, X., Yi, H., & Wang, J. (2021). Optimization of picking in the warehouse. *Journal of Physics: Conference Series*, 1861.
5. Sun, Y., Fang, M., & Su, Y. (2021). AGV Path Planning and Obstacle Avoidance Using Dijkstra's Algorithm. *Journal of Physics: Conference Series*, 1746.
6. Yang, B., Li, W., Wang, J., Yang, J., Wang, T., & Liu, X. (2020). A Novel Path Planning Algorithm for Warehouse Robots Based on a Two-Dimensional Grid Model. *IEEE Access*, 8, 80347-80357.
7. Liu, R. (2022). Research on Optimization of the AGV Shortest-Path Model and Obstacle Avoidance Planning in Dynamic Environments. *Mathematical Problems in Engineering*, *2022*(1), 2239342. doi.org/10.1155/2022/2239342.
8. Shetty, N., Sah, B., & Chung, S. H. (2020). Route optimization for warehouse order picking operations via vehicle routing and simulation. *SN Applied Sciences*, *2*, 1-18. doi.org/10.1007/s42452- 020-2076-x.
9. Tai, R., Wang, J., & Chen, W. (2019). A prioritized planning algorithm of trajectory coordination based on time windows for multiple AGVs with delay disturbance. *Assembly Automation*, *39*(5), 753-768. doi.org/10.1108/AA-03-2019-0054.
10. Chen, J., Zhang, X., Peng, X., Xu, D., & Peng, J. (2022). Efficient routing for multi-AGV based on optimized Ant-agent. *Computers & Industrial Engineering*, *167*, 108042. doi.org/10.1016/j.cie.2022.108042
11. Zhou, Y., & Huang, N. (2022). Airport AGV path optimization model based on ant colony algorithm to optimize Dijkstra algorithm in urban systems. *Sustainable Computing: Informatics and Systems*, *35*, 100716. doi.org/10.1016/j.suscom.2022.100716.
12. Meysami, A., Cuillière, J. C., François, V., & Kelouwani, S. (2022). Investigating the impact of triangle and quadrangle mesh representations on AGV path planning for various indoor environments: With or without inflation. *Robotics*, *11*(2), 50. doi.org/10.3390/robotics11020050.