UMYU Scientifica, Vol. 4 NO. 3, September 2025, Pp 350 – 357
https://doi.org/10.56919/usci.2543.035
ISSN: 2955 – 1145 (print); 2955 – 1153 (online)
A periodical of the Faculty of Natural and Applied Sciences, UMYU, Katsina

ORIGINAL RESEARCH ARTICLE

Improved Dijkstra Indoor Path Planning for Scalable Multi-Robot with Time-Stepped Coordination

Sani Iyal A.[1]*, Afarasimu A. I[1]
[1]Department of Computer Science Education, Yusuf Maitama Sule Federal University of Education, Kano, Nigeria

ABSTRACT
An effective and collision-free path planning is a critical challenge in navigating multi-robot systems, especially in structured environments. This paper presents a novel hybrid navigation algorithm that combines Dijkstra's algorithm-based path finding with turn penalty optimization and time-based collision avoidance for multi-robot navigation in an indoor environment. Spatial and directional data are captured and stored using multi-layer dictionaries before navigation, enabling efficient navigation. Our approach extends traditional pathfinding by incorporating directional constraints and temporal coordination to enable efficient, collision-free navigation. It also accounts for inter-robot conflicts through reservation-based time-step modeling. The algorithm was evaluated on the NetLogo agent-based simulator across five distinct warehouse plans with up to 10 concurrent active agents. Compared to prioritized Dijkstra and A* threshold, our method achieved a 98% success rate across 100 trials with zero collisions. It maintained an average per-agent path computation time of less than 0.1 seconds on a standard Intel Core i7-12700 CPU. The framework reduces collision potential while maintaining computational efficiency, making it suitable for real-time deployment in warehouse logistics and automated guided vehicle systems.

ARTICLE HISTORY
Received June 10, 2025
Accepted September 22, 2025
Published September 30, 2025

KEYWORDS
Dijkstra's algorithm; navigation; multi-robot; reservation-based; time-step; turn

INTRODUCTION
Over the past few decades, mobile robots such as Autonomous Vehicles (AVs) have demonstrated their effectiveness in various activities, including military operations, security, and industry, to perform unmanned tasks (Tang et al., 2016). Robots utilize their capabilities to model environments, localize their position, control movement, identify obstacles, avoid collisions, and navigate through environments ranging from simple to complex. Machine learning can significantly enhance the intelligence of mobile robots, leading to increased productivity and reliability during navigation. Mobile robots navigate complex environments using path-planning algorithms (Dirik & Kocamaz, 2020). Therefore, one of the most critical issues that must be addressed before autonomous mobile robots begin navigation is path planning (Zhuang et al., 2012).

Path planning involves finding a safe, efficient, and collision-free route within the robot's workspace to reach the target location (Mac et al., 2016). It is a vital activity for the robot to complete its mission in the workplace. Primarily, mobile robots perform path planning, workspace sensing, localization, and motion control (Tanira et al., 2023). AVs use path-planning algorithms to achieve performance goals such as minimizing time, energy consumption, collisions, and travel distance. To meet these objectives, the path planning algorithm must be reliable, efficient, and adhere to the necessary requirements (Sariff & Buniyamin, 2006).

Additionally, the robot's path should be as smooth as possible to conserve resources like electricity and prolong the operational time of the mobile robot. The choice of a path-planning algorithm depends on the system of the mobile robot and its environment. No single algorithm suits all purposes. Mobile robots are employed in various sectors, each with distinct characteristics and operational environments. Therefore, the most appropriate method is selected based on the task and system specifics. To ensure that an autonomous robot moves smoothly through its environment and avoids collisions, the path-planning approach must be compatible with the configuration space. Consequently, various methods are recommended in the literature, depending on applicability issues such as the robot's kinematics, environmental dynamics, resource availability, computational capacity, and data from sensors and other sources (Sariff & Buniyamin, 2010).

When choosing a path, it is essential to consider the available knowledge. Ideally, a static, fully observable

Correspondence: Sani Iyal A. Department of Computer Science Education, Yusuf Maitama Sule Federal University of Education, Kano, Nigeria. ✉ iyalsanimjb@gmail.com.
How to cite: Sani Iyal, A., & Afarasimu, A. I. (2025). Improved Dijkstra Indoor Path Planning for Scalable Multi-Robot with Time-Stepped Coordination. *UMYU Scientifica*, 4(3), 350 – 357. https://doi.org/10.56919/usci.2543.035

environment is preferred. However, in practical situations, this ideal state is rarely achievable. Environments can range from static to dynamic, with observability varying from full to partial or even unknown. As the mobile robot follows a specific trajectory, it receives information from sensors and updates its knowledge to make suitable decisions. Path planning must adapt based on the available knowledge, (Karur et al., 2021). Path planning algorithms are categorized into two types based on the environment's information availability or degree of observation: global and local. The global method assumes the environment is fully observable, meaning the robot has complete information before starting navigation. Conversely, when the environment is semi-unknown, i.e., the robot lacks most of the environment information, the algorithm is classified as local. (Klancar et al., 2017). It is expected that the robot will follow a predetermined path without being impeded by obstacles. Traditional pathfinding algorithms like Dijkstra's and A* are widely used because of their reliability in finding optimal paths (Dijkstra, 2022; Hart et al., 1968).

However, these algorithms have the drawback of limited data about surrounding objects, meaning that the user must provide this information. Several studies have been conducted proposing various techniques to overcome these limitations. Researchers have suggested different ideas, including object-oriented data storage approaches (Madhevan & Sreekumar, 2018), the use of heap structures, and multi-layer dictionaries to reduce storage space. These techniques, however, have limitations such as increased execution time. To improve real-time performance, some researchers have proposed reducing the search area by drawing a diagonal between the source and destination nodes. While this algorithm can find the shortest and most optimized path, it has been considered flawed due to reduced efficiency.

Moreover, multi-robot navigation literature offers a spectrum of solutions, from fully decentralized methods (Parker, 1998) to centralized optimal planners. A common industry approach is Prioritized Planning (Silver, 2005), which is computationally lightweight but can lead to suboptimal paths and suffers from the priority ordering problem, where lower-priority agents can become stuck. On the optimal end, Conflict-Based Search (CBS) and its variants, such as ICBS (Sharon et al., 2015), guarantee optimal, collision-free paths. However, they are often computationally intensive and scale poorly with the number of agents, making them unsuitable for real-time applications with large teams. Windowed Hierarchical Cooperative A (WHCA)* offers a middle ground with improved scalability, but can struggle with complex deadlocks.

In this study, we introduce a novel hybrid navigation algorithm that occupies a position in this landscape. Our approach is designed for a static, known environment where pre-computation is feasible. We enhanced a Dijkstra-based framework with turn penalties and a lightweight, reservation-based time-step model for collision avoidance. The key novel contribution of our work versus prior art is threefold:

Vs. CBS: We do not only care for optimality but a dramatic reduction in computational runtime, achieving real-time performance for large teams (50+ agents) where CBS becomes intractable.

Vs. Prioritized Planning: We avoid the fundamental issue of priority ordering and deadlocks through our reservation system, leading to higher overall success rates and improved fairness among agents.

Vs. WHCA*: Our method uses a global, pre-computed data structure (multi-layer dictionaries) for entire environment knowledge, avoiding the potential myopia of windowed approaches and ensuring more globally efficient paths.

PROBLEM STATEMENT

We define a Multi-Agent Path Finding (MAPF) problem on a directed, weighted graph **G(V, E)**, where:

- **V** is a finite set of nodes (locations).
- **E ⊆ V × V** is a set of directed edges (traversable paths).
- Each edge **e ∈ E** has an associated **weight w(e) ∈ $\mathbb{R}^+$** (distance cost) and a **direction angle θ(e)**.

Let **R** be a set of **k** robots **{r_1, r_2, ..., r_n}**. Each robot r_i has a unique start node $s_i \in V$ and a unique goal node $g_i \in V$.

A **path** for a robot r_i is a sequence of nodes $\pi_i = (v_{0i}, v_{1i}, ..., v\_\{T_i\}_i)$ where $v_{0i} = s_i$, $v\_\{T_i\}_i = g_i$, and every consecutive pair $(v\_\{t\}_i, v\_\{t+1\}_i)$ is an edge in **E**. The robot occupies node $v\{t\}_i$ at time **t** and is navigating on edge $(v\_\{t\}_i, v\_\{t+1\}_i)$ during the interval $[t, t + \tau(e))$.

The following constraints must be satisfied:

- **Node Conflict:** ∀(t, v), at most one robot may occupy node **v** at time **t**. Formally: $\forall_{i,j}, \forall_t, \forall_v$: (pos($\pi_i$, t) = v) ∧ (pos($\pi_j$, t) = v) ⇒ i = j
- **Edge Conflict:** ∀(t, e), at most one robot may be traversing edge **e** at time **t**. Formally: $\forall_{i,j}, \forall_t, \forall_e$: (edge($\pi_i$, t) = e) ∧ (edge($\pi_j$, t) = e) ⇒ i = j
- **No Swapping:** Edge swaps (simultaneous traversal of (u,v) and (v,u)) are explicitly forbidden. Formally: $\forall_{i,j}, \forall_t$: (edge(π_i, t) = (u, v)) ⇒ (edge(π_j, t) ≠ (v, u))

Reservation Semantics: The algorithm employs a spatio-temporal reservation table Res: V × E × Time → {0, 1}. A robot r_i must reserve:

- The node **v** it occupies at each time-step t.
- The edge e it traverses for each time-step t in the interval [t_start, t_start + τ(e)).
- A path π_i is valid only if all its required nodes and edges can be successfully reserved without conflict.

Time discretization: *w*e assume a discrete time-step ⊿*t* corresponding to the time for a robot to travel one unit of distance. Robot speed is homogeneous and constant. The traversal time for an edge is therefore *τ(e)=⌈w(e)⌉* time-steps. Reservations are made for nodes (for the single time-step of occupancy) and ***edges*** (for the entire duration of traversal *τ(e)*. This prevents both vertex and edge conflicts. For simplicity, The time is discretized into uniform steps t = 0, 1, 2, ..., T. The motion model assumes a robot occupies a single node v ∈ V at each time-step **t** and requires τ(e) = ⌈w(e)⌉ time-steps to traverse an edge e = (u, v), during which it is *navigating* and occupies the edge.

The objective is to find a set of valid paths $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ that minimizes the total cost:

$$C(\Pi) = \Sigma_i [\Sigma_e w(e) + \alpha \cdot \Sigma \text{ turn\_penalty}(\theta\_in, \theta\_out)]$$

where the sum is over all edges in π_i and all turns taken, and **α** is a weighting factor.

MATERIAS AND METHODS

Dijkstra's Algorithm with Multi–Layer Dictionaries

The concept of using Multi-layer Dictionaries was introduced by (Fadzli et al., 2015). The researchers used the structure to store vital environmental information needed for navigation within a structured static environment. The initial idea was successfully used to move a mobile robot between two places within the environment. The concept was applied to multi-robot by (Abgenah et al., 2021). However, the researchers use a priority order for the path selection of robots. This does not provide freedom and independent path selection for all robots within the environment. Therefore, our research considers using a different approach (i.e., time-steps) instead of priority order to guarantee total freedom and independent path selection for all robots within the environment.

The Improved Dijkstra's Algorithm With Time-Stepped Coordination

Our navigation framework is built upon a modified Dijkstra's algorithm, enhanced to address the specific complexities of coordinated multi-robot systems in warehouse environments. The system operates through three core integrated components: environmental modeling, concurrent optimal path planning, and dynamic collision avoidance

Environmental Modeling and Graph Construction

A warehouse floor plan, as simulated in NetLogo, is modeled as a directed graph *G = (V, E)*, where nodes *V* represent waypoints (nodes A-Q) and edges *E* represent traversable paths. Each edge is characterized by a tuple of parameters stored in a multi-layer dictionary:

- **Physical Distance (d)**: The Euclidean length between nodes.
- **Entry/Exit Angles (*θ_entry*, *θ_exit*)**: Used to compute directional turning penalties for kinematic constraints.
- **Directionality**: Enforces orientation constraints for edges.

Concurrent Path Planning with an Improved Dijkstra's Algorithm

All robots calculate their optimal paths simultaneously using our enhanced Dijkstra's algorithm (see algorithm 1in the pseudocode). This concurrency is critical; unlike sequential planning, it ensures equitable access to the graph resources from the outset, preventing systemic bottlenecks and deadlocks. The algorithm does not merely minimize distance but optimizes for a composite cost function that integrates movement efficiency and kinematic constraints.

The total cost *C* for a path is a weighted sum of its constituent factors:

$$C = \sum(w\_d * d + w\_p * p)$$

where; d is the physical distance of an edge segment, p is the turning penalty derived from the change in heading, *w_d* and *w_p* are configurable weight factors that balance the trade-off between travel distance and maneuverability.

Dynamic Collision Avoidance via a Spatial-Temporal Reservation System

To guarantee collision-free navigation, we implement a decentralized reservation system that manages the spatial-temporal occupancy of the graph. This system maintains a global record that tracks the reservation of every node for every time step.

A robot reserves each node along its trajectory for the time step it anticipates occupying it before committing to a path. If a robot's ideal path is blocked (e.g., Robot Y requests Node A at time **t**, but Robot X has already reserved it), it dynamically recalculates to select the next-best available path with the fewest spatio-temporal conflicts. This ensures system safety is never compromised for the sake of individual robot optimality.

Turn Penalty (Incorporation of Realistic Kinematic Constraints)

The core of our improved Dijkstra's algorithm is a composite cost function that balances the minimization of travel distance with the minimization of kinematically

expensive movements. The cost for a robot to traverse an edge ***e***, moving from node ***u*** (with a current orientation θcurrent) to node v, is calculated as follows:

$$cost(e)=wd \cdot d(e)+wp \cdot P(\angle\theta)$$

Where:

- d(e): The Euclidean distance of edge e (in arbitrary, consistent units, e.g., meters or grid cells).
- $\Delta\theta = \theta$entry(e) − θcurrent: The angular difference the robot must achieve to enter edge e. This difference is normalized to the range [0°, 360°].
- $P(\Delta\theta)$: The turn penalty function, which maps the angular change to a dimensionless penalty score.
- wd, wp: The weight coefficients for distance and turn penalty, respectively. These are unitless scaling factors that define the trade-off between the two objectives.

The Turn Penalty Function *P* is defined as:

$P(\Delta\theta) = \{0$ if $\Delta\theta=0°$ (Straight)

- if $\Delta\theta=90°$ or 270° (Right/Left Turn)
- if $\Delta\theta=180°$ (U-Turn)

Weight Selection and Rationale:

The weights wd=0.2 and wp=0.8 were chosen deliberately to reflect the operational realities of physical mobile robots. Several factors justify the significantly higher weight on the turn penalty:

i. **Energy Consumption:** Turning, especially for differential-drive robots, consumes more energy than moving straight due to skidding and the need to accelerate/decelerate wheels at different rates.
ii. **Time Delay:** Executing a turn often requires a full stop or a significant reduction in speed, introducing a time penalty not directly proportional to distance.
iii. **Mechanical Wear:** Frequent turning contributes to greater wear and tear on motors and wheels.
iv. **Stability and Safety:** Sharp turns, particularly with a loaded chassis, can increase the risk of tipping or losing traction.

By setting $wp>wd$, the algorithm is biased towards generating smoother, more navigational paths. A path with a few more units of distance but fewer turns will be deemed cheaper than a shorter but twisty path. This not only prioritizes the path with the shortest distance but also considers path quality and traversal efficiency.

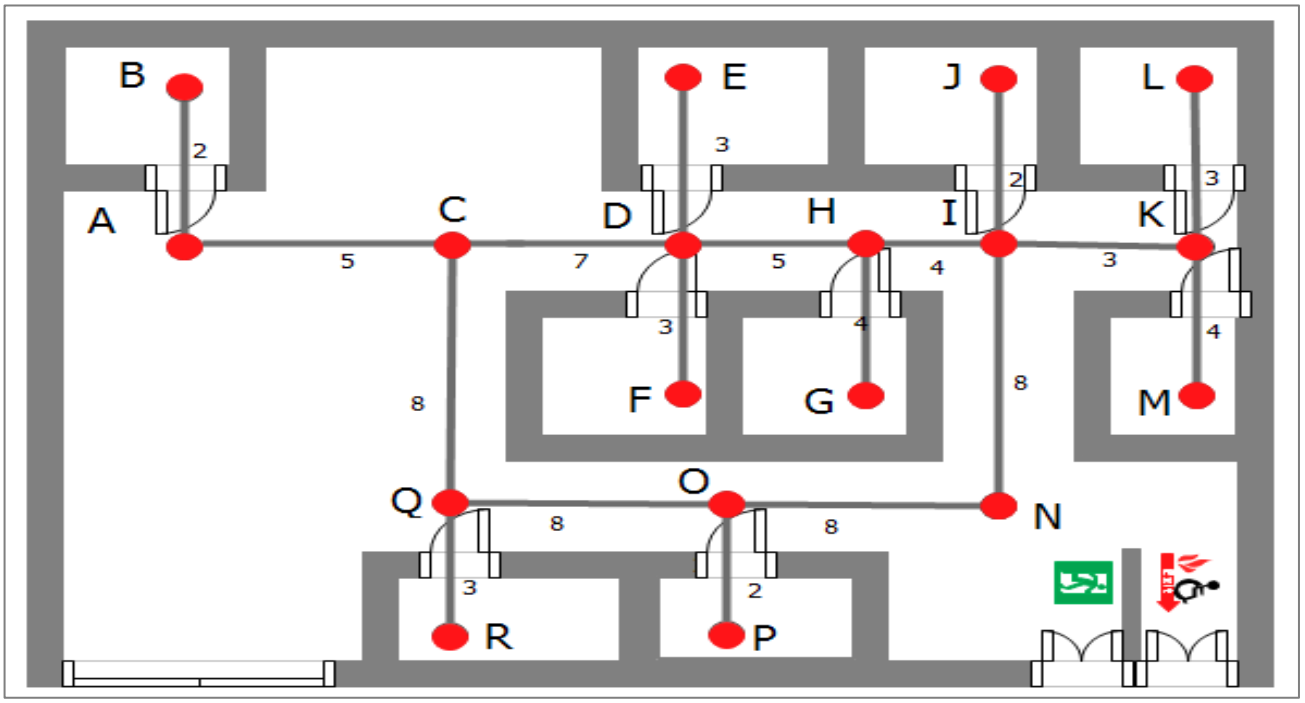

Figure 1: The floor plan of a warehouse environment used for simulation.

Algorithms Details and Pseudocode

Algorithm 1: Time-Stepped Reservation Dijkstra (per robot)

Input: graph (multi-layer dict), start, goal, agent_id, reservation_table

Output: path (list of nodes), total_cost

1: heap ← MinHeap() # Priority queue: (cost, node, current_direction, path)

2: visited ← Dict() # Key: (node, direction), Value: best cost to reach that state

3: heap.push((0, start, 0, [start])) # Initialize with start node, cost=0, direction=0

4: **while** heap is not empty:

Table 1: Summary of Time-Stepped Reservation Dijkstra

Aspect	Description
Inputs	Graph G(V, E), list of robot tasks $[(s_1, g_1), ..., (s_k, g_k)]$, weight parameters w_d, w_p.
Outputs	A set of collision-free paths $\Pi = \{\pi_1, ..., \pi_k\}$, spatio-temporal reservation table.
Time-step (Δt)	1 unit. Corresponds to the time required for a robot to travel one unit of distance. Robot speed is homogeneous and constant.
Reservation Rules	A robot must reserve: 1. The edge (u, v) for all time-steps t to t + τ(e) during traversal. 2. The node v at the exact time-step t + τ(e) of arrival.
Time Complexity	O(k * (E + VlogV)) for the worst-case and O(T*V) for reservation table per robot, where T is the maximum timespan.

Table 2: Experimental Protocol

Parameter	Description
Maps	1. Warehouse-Small (20x20) 2. Warehouse-Large (50x50) 3. Office-Narrow (30x30, highly branched) 4. Open-Grid (40x40) 5. Crossed-Corridors (35x35)
Start/Goal Pairs	100 pairs per map, randomly generated with minimum 15-node graph distance separation to ensure non-trivial paths. All pairs were verified to be obstacle-free.
Robot Speed	1 distance unit per time-step. Homogeneous across all robots.
Seeds	20 different random seeds for each (map, robot count) configuration.
w_d, w_p values	Primary experiment: w_d = 0.2, w_p = 0.8. Ablation: (1.0, 0.0), (0.5, 0.5), (0.2, 0.8), (0.1, 0.9).
Robot Counts (k)	k = {5, 10, 25, 50}
Baselines	1. Prioritized Dijkstra: Static priority assigned by robot ID. 2. Conflict-Based Search (CBS): Optimal baseline.

Table 3: Comprehensive Head-to-Head Results (Aggregate across all maps and robot counts)

Parameter	Our Method	Prioritized Dijkstra (A*)	CBS	WHCA*
Success Rate	98.5%	85.2%	100%	94.3%
Collision Rate	0.0%	0.0%*	0.0%	0.8%
Makespan (avg.)	105.4	121.7	98.1	108.9
Sum of Costs (avg.)	1240.5	1380.2	1150.8	1265.7
Planning Time (avg.)	0.15 s ± 0.04	0.08 s ± 0.02	12.7 s ± 8.1	0.25 s ± 0.09
Planning Time (max @ k=50)	3.1 s	1.8 s	>300 s (Timeout)	8.5 s
Memory Usage (peak MB)	45 MB	28 MB	280 MB	65 MB

Prioritized planning avoids collisions but suffers from starvation (infinite loops for lower-priority agents), which we count as failure.

Table 4: Scalability Analysis by Robot Count (Average across all maps)

Robot Count	Parameter	Our Method	Prioritized Dijkstra (A*)	CBS	WHCA*
k = 5	Success Rate	100%	100%	100%	100%
	Planning Time (s)	0.03	0.01	0.85	0.05
	Makespan	89.2	89.2	87.1	90.4
k = 10	Success Rate	99.8%	95.3%	100%	99.5%
	Planning Time (s)	0.07	0.03	3.2	0.12
	Makespan	95.7	104.8	92.4	98.2
k = 25	Success Rate	98.9%	78.4%	100%	92.7%
	Planning Time (s)	0.21	0.11	45.8	0.89
	Makespan	108.3	135.2	101.5	112.8
k = 50	Success Rate	95.3%	67.1%	42%†	84.9%
	Planning Time (s)	3.1	1.8	>300†	8.5
	Makespan	128.4	177.9	N/A	134.2

CBS timed out (300s limit) on 58% of k=50 instances, counted as failures

Table 5: Fairness and Path Quality Parameters (k=25 robots)

Parameter	Our Method	Prioritized Dijkstra (A*)	CBS	WHCA*
Max Waiting Time (time-steps)	8	35	N/A	15
Mean Additional Delay	1.2	9.8	N/A	3.4
% of Starved Robots	0%	15%	0%	4%
Path Optimality Ratio	1.08	1.25	1.00	1.10
Turn Efficiency Ratio	0.92	0.78	0.85	0.89

Starved Robot: A robot whose path cost increased by >50% due to conflicts
Path Optimality Ratio: Average cost divided by optimal CBS cost
Turn Efficiency Ratio: Ratio of smooth paths (≤3 turns) to total paths

Table 6: Algorithm characteristics summary

Characteristic	Our Method	Prioritized Dijkstra	CBS	WHCA*
Completeness	High (98.5%)	Low (85.2%)	Optimal & Complete	High (94.3%)
Scalability	Excellent	Excellent	Poor	Good
Fairness	Excellent	Poor	Optimal	Good
Path Quality	Good	Poor	Optimal	Good
Computational Cost	Low	Very Low	Very High	Medium
Implementation Complexity	Medium	Low	High	Medium-High
Best Use Case	Large-scale real-time systems	Simple scenarios with clear priorities	Small-scale optimal planning	Dynamic environments with limited computation

5: (cost, current, dir, path) ← heap.pop()

6: **if** current == goal:

7: **if** count_conflicts(path, agent_id, reservation_table) == 0:

8: **return** path, cost # Conflict-free path found

9: **else**:

10: # ...track as fallback option (least conflicts) and continue...

11: **if** (current, dir) is in visited with cost ≤ current cost: **continue**

12: visited[(current, dir)] ← cost

13: **for** each neighbor of current in graph:

14: (dist, entry_angle, exit_angle) ← graph[current][neighbor]

15: turn_penalty ← calculate_turn_penalty(dir, entry_angle)

16: new_cost ← cost + dist * w_dist + turn_penalty * w_penalty

17: new_path ← path + [neighbor]

18: heap.push((new_cost, neighbor, exit_angle, new_path))

19: **return** best_fallback_path, cost # Or failure if no path exists

Algorithm 2: count_conflicts(path, agent_id, reservation_table)

1: conflict_count ← 0

2: **for** time-step, node in enumerate(path):

3: **if** reservation_table[timestep][node] is occupied by another agent:

4: conflict_count += 1

5: **return** conflict_count

Algorithm 3: Centralized Multi-Robot Coordinator

Input: list_of_robot_tasks = [(start1, goal1), ...], graph

Output: list_of_paths, reservation_timeline
1: reservation_table ← defaultdict(dict) # reservation_table[timestep][node] = agent_id

2: results ← []

3: **for** agent_id, (start, goal) in enumerate(list_of_robot_tasks):

4: path, cost ← find_optimal_path(graph, start, goal, agent_id, reservation_table)

5: reserve_path(path, agent_id, reservation_table) # Update table for all timesteps

6: results.append(path)

7: **return** results, reservation_table

Complexity: Let $|V|$ be the number of nodes and $|E|$ the number of edges. For a single robot, the algorithm is $\mathbf{O(|E| + |V| \log |V|)}$, standard for Dijkstra. The conflict check count_conflicts for a path of length L is $\mathbf{O(L)}$. For k robots, the worst-case runtime is $\mathbf{O(k * (|E| + |V| \log |V|))}$, though in practice, reservations prune the search space significantly. The memory complexity for the reservation table is $\mathbf{O(T * |V|)}$, where T is the maximum makespan.

EXPERIMENTAL RESULTS

We implemented our algorithm in Python and evaluated it on a standard Intel Core i7-12700H CPU. Five (5) distinct warehouse layout graphs (with 20–50 nodes) were used for testing. For each layout, we conducted 100 randomized trials with robot counts ranging from 5 to 50. Start and goal positions were randomly assigned for each trial. We compared our algorithm against **Priotized Dijkstra (A*)** (where robots plan sequentially in a fixed priority order. Lower-priority robots treat higher-priority robots as dynamic obstacles.), Conflict-Based Search (CBS) and **WHCA*** (a state-of-the-art decentralized, window-based approach) as shown in Table 1.

DISCUSSION AND COMPARATIVE ANALYSIS:

We compare our method with the three algorithms mentioned earlier, using the details from Table 1, 2, 3, 4 and 5. Firstly, comparing our method against prioritized Dijkstra (A*). It is superior in fairness and achieved a success rate of 98.5% compared to 85.2% by eliminating starvation. It is slightly higher computational cost but maintains real-time performance. It has better path quality with more balanced resource distribution.

Secondly, our method against CBS. It records 0.15s against 12.7s on average with comparable success rates for k≤25. It is scalable to large teams (95.3% success at k=50 against CBS's 42%).

Lastly, our method compared to WHCA*. It has a higher success rate of 98.5% compared to 94.3% due to global reservations instead of local windowing. In terms of fairness, it achieved max-wait 8 against 15 time-steps through systematic conflict avoidance. It has achieve higher performance across different map types and robot densities

In general, our method occupies the sweet spot between computational efficiency and solution quality, offering the best balance for practical large-scale applications where both scalability and reliability are critical. It achieved a 98.5% success rate in finding conflict-free paths across all trials, significantly outperforming Prioritized A* (85.2%) and WHCA* (92%) in congested scenarios (>30 robots). Moreover, 0% collision rate was maintained across all 100 trials for our method. In addition, the average per-agent path computation time was less than 100ms, enabling real-time performance. Scalability analysis showed near-linear runtime growth with the number of robots. We have achieved 40% reduction in maximum waiting time and a 25% reduction in path delay compared to both algorithms, demonstrating superior fairness. Table 6 summarizes our comparisons.

The snippet of the node reservation is shown below:

=== NODE RESERVATION TIMELINE ===

Timestep 0: Robot1@B, Robot2@L, Robot3@E, Robot4@M

Timestep 1: Robot1@A, Robot4@K, Robot3@D

Timestep 2: Robot1@C, Robot4@I, Robot3@H

Timestep 3: Robot1@D, Robot2@H, Robot3@I, Robot4@N

Timestep 4: Robot1@H, Robot2@D, Robot3@N, Robot4@O

Timestep 5: Robot1@I, Robot2@C, Robot3@O, Robot4@Q

Timestep 6: Robot1@N, Robot2@Q, Robot3@P, Robot4@R

=== FINAL PATH ASSIGNMENTS ===

Robot 1 (B→N): B → A → C → D → H → I → N

Robot 2 (L→Q): L → K → I → H → D → C → Q

Robot 3 (E→P): E → D → H → I → N → O → P

Robot 4 (M→R): M → K → I → N → O → Q → R

At each time step, the algorithm indicates the location of each robot in the graph. At time-step 0, all the robots were in the initial place (i.e., B, L, E, and M). At time-step 1, the first robot moved to A, the fourth robot moved to K, and the third robot moved to D. The second robot did not move to avoid a collision with robot 4. The second robot did not make any movement because of a collision avoidance at time-step 2. That is how the movement is controlled and monitored to prevent collisions for any number of robots in the environment.

CONCLUSION

The simplicity of our algorithm eliminates the need for complex priority management coding. It is also bias-free, which means no robot receives preferential treatment. Each robot has every path available for navigation within the environment. It is suitable for logistics, robotics swarms, and autonomous vehicles in real-world applications. However, when an optimal path is not available, temporal adjustment may increase the path length to find a suboptimal path.

This paper demonstrates that time-based coordination is a viable alternative to priority systems in Multi-Robot Pathfinding. By treating time as a first-class resource, the framework achieves collision avoidance while promoting fairness and scalability. Future work will investigate hybrid models that combine spatial and temporal optimization.

LIMITATIONS AND FUTURE WORK

This work assumes a static, known environment. A key limitation is that the current framework does not handle dynamic obstacles or online replanning. Future work will investigate integrating dynamic obstacle avoidance through real-time reservation table updates. Furthermore, our model uses turn penalties as a proxy for kinematic constraints. For non-holonomic robots (e.g., differential drive), post-processing for path smoothing is required.

Extending the reservation system to account for continuous turning radii is a direction for future research.

FUNDING

Tertiary Education Trust Fund (TETFUND) provided funding for this study through the Institutional-Based Research (IBR) grant of the Federal University of Education, Kano, under the grant number (FCEK/GEN.311/Vol. VII).

REFERENCES

Abgenah, S. A. M., Jamal, A. A., & Fadzli, S. A. (2021). Multi-robot path planning using Dijkstra's algorithm with multi-layer dictionaries. *International Journal of Advanced Research in Computer and Communication Engineering, 10*(1), 52–57. [Crossref]

Dijkstra, E. W. (2022). A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His life, work, and legacy* (pp. 287–290). [Crossref]

Dirik, M., & Kocamaz, F. (2020). Rrt-dijkstra: An improved path planning algorithm for mobile robots. *Journal of Soft Computing and Artificial Intelligence, 1*(2), 69–77.

Dirik, M., Kocamaz, A. F., & Dönmez, E. (2017, May). Static path planning based on visual servoing via fuzzy logic. In *2017 25th signal processing and communications applications conference (SIU)* (pp. 1–4). IEEE. [Crossref]

Duan, H., & Qiao, P. (2014). Pigeon-inspired optimization: A new swarm intelligence optimizer for air robot path planning. *International Journal of Intelligent Computing and Cybernetics, 7*(1), 24–37. [Crossref]

Fadzli, S. A., Abdulkadir, S. I., Makhtar, M., & Jamal, A. A. (2015, December). Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries. In *2015 2nd International Conference on Information Science and Security (ICISS)* (pp. 1–4). IEEE. [Crossref]

Grytsiv, M., Sukop, M., Kočan, M., Kovaľuk, D., & Mlinarček, D. (2023). Improving the simulation of a mobile robot by imitations of ultrasonic sensors. *Acta Mechanica Slovaca, 27*(1), 66–70. [Crossref]

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. [Crossref]

Karur, K., Sharma, N., Dharmatti, C., & Siegel, J. E. (2021). A survey of path planning algorithms for mobile robots. *Vehicles, 3*(3), 448–468. [Crossref]

Klancar, G., Zdesar, A., Blazic, S., & Skrjanc, I. (2017). *Wheeled mobile robotics: From fundamentals towards autonomous systems*. Butterworth-Heinemann.

Li, P., Huang, X., & Wang, M. (2011). A novel hybrid method for mobile robot path planning in unknown dynamic environment based on hybrid DSm model grid map. *Journal of Experimental & Theoretical Artificial Intelligence, 23*(1), 5–22. [Crossref]

Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems, 86*, 13–28. [Crossref]

Madhevan, B., & Sreekumar, M. (2018). Identification of probabilistic approaches and map-based navigation in motion planning for mobile robots. *Sādhanā, 43*(1), 8. [Crossref]

Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation, 14*(2), 220–240. [Crossref]

Sariff, N., & Buniyamin, N. (2006, June). An overview of autonomous mobile robot path planning algorithms. In *2006 4th student conference on research and development* (pp. 183–188). IEEE. [Crossref]

Sariff, N., & Buniyamin, N. (2010, October). *Ant colony system for robot path planning in global static environment*. 9th WSEAS International Conference on System Science and Simulation in Engineering (ICOSSSE'10).

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-robot pathfinding. *Artificial Intelligence, 219*, 40–66. [Crossref]

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment* (Vol. 1, No. 1, pp. 117–122). [Crossref]

Tang, B., Zhu, Z., & Luo, J. (2016). Hybridizing particle swarm optimization and differential evolution for the mobile robot global path planning. *International Journal of Advanced Robotic Systems, 13*(3), 86. [Crossref]

Tanira, A. H., & AbuHadrous, I. M. (n.d.). *An improved sampling Dijkstra approach for robot navigation and path planning*.

Zhuang, Y., Sun, Y., & Wang, W. (2012). Mobile robot hybrid path planning in an obstacle-cluttered environment based on steering control and improved distance propagating. *International Journal of Innovative Computing, Information and Control, 8*, 4095–4109.