Article | Received 9 April 2025; Revised 23 July 2025; Accepted date 19 September 2025; Published date
https://doi.org/10.55092/rl20250008

A lightweight and rapid bidirectional search algorithm

Momodou Bah, Ioanna Giorgi* and Giovanni Luca Masala

School of Computing, University of Kent, Canterbury, UK

* Correspondence author; E-mail: i.giorgi@kent.ac.uk.

Highlights:

- Scalable and robust path generation in high-density maps via adaptive bidirectional merging.
- Handles non-central merges with dynamic frontier attraction.
- Rapid path recomputing in real-time in dynamic or partially known maps.
- Outperforms baseline methods in node expansion (> 40%), timing (> 83%) and overhead (> 79%).
- Low memory footprint optimal for resource-limited robotic platforms.

Abstract: Path planning in mobile robotics is critical for efficient navigation in complex environments. To date, grid-based planning remains a popular choice due to its simple spatial representation, integration with sensor data and the ability to encode motion constraints. This work contributes to this direction by proposing a novel and complete grid-based algorithm, LiteRBS (Lightweight and Rapid Bidirectional Search), optimised for computational efficiency and scalability. The algorithm balances aggressive bidirectional, forward search heuristics with a fallback strategy to reserve queues. Evaluated extensively against well-known algorithms like A*, bidirectional A*, Jump Point Search (JPS) and Shortest Path Faster Algorithm (SPFA), LiteRBS demonstrates statistically and practically significant reductions in memory usage (79%–96%), node expansion (40%–92%) and runtime (83%–98%), the latter remaining density-invariant across increasing spatial and environmental complexity. It handles non-central merges by dynamically adjusting search targets in each direction to "attract" nodes rapidly towards convergence. This yields flat search overhead as the problem scales to large and crowded maps. Real-world deployment on a Turtlebot3 robot demonstrates its responsiveness under partial observability and dynamic obstacle conditions. Overall, LiteRBS offers a scalable, lightweight and practical solution for the navigation of terrestrial robots in complex, resource-constrained environments.

Keywords: bidirectional search algorithm; grid-based search algorithm; LiteRBS; mobile robots; path-finding algorithm

1. Introduction

Path planning is a fundamental and extensively researched aspect of the navigational control of mobile robots. It is commonly defined as the process of identifying a collision-free trajectory from an initial position to a designated target within a space populated by obstacles, based on optimisation criteria

such as distance, time and cost [1]. The aim, typically, is to identify the 'optimal' path or, at a minimum, an admissible approximation of it. A standard definition of algorithmic optimality is "An algorithm is optimal if no other search algorithm uses less time or space or expands fewer nodes, with a guarantee of a solution quality" [2]. However, an optimal path can vary based on the application and may refer to that which minimises one or more objective optimisation functions [3]. For example, search and rescue missions necessitate minimising the time required to find or traverse a viable path [4]. Another important optimisation criterion could be the energy consumption of the robot required to identify a path for those applications that are energy resource-limited, e.g., planetary explorations [5]. Other constraints for path-generation may be restricting the number of manoeuvres the robot can execute, like the number of sharp turns throughout the path, to preserve tyres or avoid delays caused by the manoeuvre [6]. Certainly, the above can coincide with the shortest path; however, each optimisation function may simultaneously constrain the set of feasible paths, potentially including that which is the shortest. Thus, in mobile robots, a collision-free optimal path is defined within the context of a usage scenario [7], and the algorithmic choice is ultimately dependent on the specific requirements of the application [8].

To date, classical path planning solutions remain a popular choice for robotics, largely due to their ability to reliably guarantee a path given sufficient resolution and accurate information about the environment [9]. These methods typically operate on grid configurations [10], which are particularly well-suited to ground mobile robots due to their compatibility with discrete, structured representations of 2D environments. A common example is occupancy grid mapping, which discretises continuous space into uniform cells, allowing robots to navigate environments using straightforward computational models [11]. This structured approach is characterised by simplicity and aligns well with the constraints of wheeled or tracked robots, where motion is typically planar and governed by limited kinematic flexibility [12]. Moreover, they integrate well with simultaneous localisation and mapping (SLAM) outputs for real-world deployment. For such configurations, well-established algorithms like Dijkstra and A* remain commonly applied in mobile robots, and other applications like Google maps [13] due to their ability to balance optimality and completeness. In particular, A* heuristic search strategy makes it attractive for time-sensitive or computationally constrained applications, as it significantly reduces the search space while maintaining optimality [14]. Nevertheless, these algorithms may not scale well to complex environments, and they typically assume static global knowledge of the world. Moreover, their computational cost can increase rapidly when increasing the granularity of the grids, due to finer details [15], compromising both efficiency and path smoothness [16].

Therefore, multiple improved planners have been proposed to address different needs, like speed, adaptivity, or robot-specific constraints, which can be layered over grid meshes or adapted to work with them. For example, to improve the efficiency of A* for large-scale maps, a bidirectional A* has been proposed, which reduces computation time and memory consumption [17]. Another popular variant is the Jump Point Search (JPS) algorithm, which accelerates A* by skipping over symmetric paths in uniform-cost grids, making it particularly efficient in sparse environments [18]. However, in dense environments, identifying jump points significantly compromises computation efficiency [19], and efforts to optimise this, like polygon-based pruning (PO-JPS), may introduce rigid paths or only local optima [20]. Bidirectional variants of JPS have also been proposed, which reduce search times and node expansions, but often at the cost of the two search instances failing to converge under certain conditions [19,21].

Further to approaches that emphasise speed and computation efficiency, the optimised Shortest Path Faster Algorithm (SPFA) [22] introduces a forward-star structure and smart queue prioritisation, resulting in strong average-case performance gains and memory efficiency, though its validation is limited to random graph testing. Despite being less commonly used than A* and Dijkstra variants in terrestrial mobile robots, this method holds value in scenarios requiring frequent updates to path costs.

Grid-based planners are often complemented with sampling-based methods, like the Rapidly-exploring Random Tree (RRT) and its variants [23] for planning in complex and high-dimensional spaces. RRT is often adapted to work in grid configurations by constraining its sampling and expansion to grid cells. However, applying RRTs to discretised grid configurations is not without limitations. RRTs are inherently designed for continuous state spaces; thus, their exploratory efficiency diminishes when constrained to grid cells, resulting in redundant node expansions and compromising guarantees of completeness and optimality. This makes them less suitable than deterministic and systematic grid planners like A*. As a result, RRTs are not typically used directly on grids but rather in combination with traditional grid searches, offering improved speed and adaptability to challenging environments.

Ultimately, when it comes to path planning of mobile robots, no single algorithm can solve all planning problems in practical applications. While the field has made striking progress, for example, in analytic learning (e.g., [24]) and adaptive control (e.g., [25,26]), classical grid-based planning remains foundational for terrestrial robotic navigation. In particular, these planners provide consistent and repeatable paths with well-understood theoretical guarantees of completeness and optimality, but also serve as an effective backbone for advanced hybrid planning approaches [27]. They offer clear representations of the environment and integrate well with sensor inputs like LIDAR and cameras without complex transformations. With ongoing optimisations, they can provide low-latency, memory-efficient planning suitable for embedded robotic systems with constrained resources.

Our work contributes to this ongoing line of research by proposing a novel, lightweight and rapid bidirectional search algorithm (LiteRBS). Our approach combines a systematic wavefront expansion with a bidirectional heuristic strategy to improve computational speed and memory efficiency while maintaining determinism in grid-based planning. The proposed algorithm, along with the protocol to validate its performance against baseline methods, is described in Section 2. Section 3 presents validation results comparing LiteRBS to state-of-the-art algorithms. In section 4, we demonstrate the algorithm's implementation in a low-cost mobile robot to dynamically handle real-world scenes with varying obstacles or incomplete maps due to sensor constraints. Concluding remarks follow in Section 5.

2. Methods

2.1. The LiteRBS algorithm

The proposed LiteRBS algorithm is partially inspired by Lee's pathfinding algorithm [28] with modifications to the wave propagation and the integration of a bidirectional heuristic search. The traditional Lee approach begins traversal at a specific source node and propagates outward to all unblocked adjacent nodes, one node at a time, while maintaining both a propagation list, *i.e.*, p-list and a neighbour list, *i.e.*, n-list. Although this router finds the shortest path if one exists, it incurs a significant search space.

Algorithm 1. The LiteRBS algorithm.

```
1:  visitedList = ({start}, {goal})
2:  queue = ([start], [goal])
3:  reserveList ([], [])
4:  currentNode = [{value:start}, {value:goal}]
5:  while queue[0] and queue[1] are not empty and distance(currentNode[0].value, currentNode[1].value) > 1 do
6:      get neighboursS of currentNode[0].value
7:      if neighboursS is empty then
8:          if reserveList[0] is empty then
9:              return NO_PATH
10:         end if
:
11:         reserveNodeS, reserveNodeSParent ←dequeue(reserveList[0])
:
12:         if reserveNodeS is in visitedList[1] then
:
13:             return MERGE_FROM_TRIAL (reserves)
:
14:         end if
:
15:         add reserveNodeS to queue[0]
:
16:         VisitedList[0][reserveNodeS] = {reserveNodeS, reserveNodeSParent}
:
17:     else
:
18:         for each neighbour Ps in neighboursS do
:
19:             get distance (fitness) from currentNode[1].value
:
20:         end for
:
21:         minNeighbourS←BEST_FITNESS(neighboursS)
:
22:         add remaining neighbours to reserveList[0]
:
23:         add minNeighbourS to queue[0]
:
24:         visitedList[0][minNeighbourS] = (minNeighbourS, neighboursS)
:
25:         if minNeighbourS is in visitedList[1] then
:
26:             return MERGE_FROM_TRIAL
:
27:         end if
:
28:     end if
:
29:     ***repeat lines 6-28 for the other search direction, currentNode[1].value***
:
30:     currentNode[0].value ← dequeue(queue[0])
:
31:     currentNode[1].value ← dequeue(queue[1])
:
32: end while
:
33: if distance(currentNode[0].value, currentNode[1].value) ≤ 1
:
34:     return MERGE_DIRECT
:
35: end if
:
36: return path cannot be found
:
```

LiteRBS is designed with an emphasis on computational overhead. Our approach leverages priority-based node expansion, which optimises the exhaustive traversal process in Lee's algorithm to expand only to the most promising node. In each direction, the algorithm propagates forward aggressively to the node with the smallest distance, Manhattan or Euclidean, from the counterpart node. It maintains respective visited lists to track nodes evaluated in each direction, thus preventing redundant exploration. The remaining candidate nodes are held in reserve lists without traversing them immediately as a fallback strategy should the selected best-fitness node fail to yield a valid path. To reduce complexity, the reserved nodes are stored in simple queues and retrieved using a LIFO logic. The search continues until the path is found, or all nodes and the reserves are exhausted, concluding that no path exists in finite time (Algorithm 1, Figure 1).

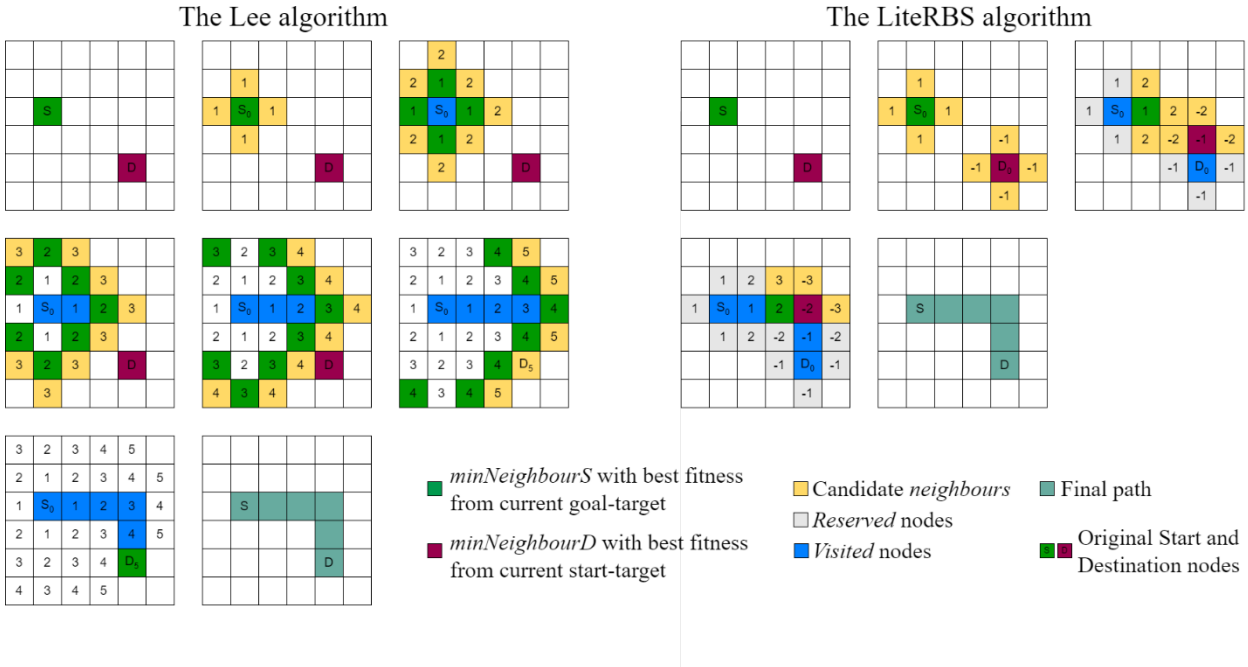


Figure 1. Propagation phases in Lee's algorithm (left) and the LiteRBS algorithm (right). Lee's algorithm traverses a large search space of all candidate nodes to identify the shortest path. In contrast, the LiteRBS algorithm selects the most promising node from each direction while reserving the rest to fall back if the current search does not produce a viable path.

As in traditional bidirectional search, identifying a merge point between the forward and backwards expansions in a non-trivial problem, particularly in dense or irregular environments, where the instances may advance unevenly or bypass each other. Therefore, to ensure completeness, the LiteRBS algorithm employs different path-merging strategies (Figure 2). Both search instances advance by always selecting the neighbour with the best fitness toward the opposite end, aiming for direct convergence by greedily moving along the most promising front (Figure 2A). One feature of the LiteRBS algorithm is that both search instances maintain a list of visited nodes from the opposite counterpart (algorithm 1). Thus, when direct convergence is not possible (e.g. due to the presence of obstacles) or if the next best candidate in either direction was already visited by the opposite search, the search frontiers will intersect at that vertex, allowing the algorithm to terminate early (Figure 2B). This can occur during regular expansion towards promising nodes or when falling back to reserve nodes (algorithm 1, lines 13, 26). The path is reconstructed from the start (S) through the intersection vertex (I) to the destination (D). While this

method does not guarantee the shortest path, it enhances the algorithmic adaptability and efficiency, especially to avoid stalling in dense areas characterised by closely spaced obstacles, where instances may approach each other closely but cannot meet due to an obstruction in between (Figure 2B).

Lastly, a defining feature of the LiteRBS algorithm is that the search target nodes in each direction update with every iteration, approaching gradually. The most recently reached node in each direction becomes the new target for the opposite search. This is achieved by calculating the fitness function based on the distance to this newly reached node rather than the original start and goal nodes. For instance, if the search from the start node (S) has progressed to node S[1] and the search from the goal (D) has reached node D[−1], then in the next iteration, S[1] will calculate the distance to reach D[−1], and D[−1] will likewise evaluate its fitness function towards S[1]. This strategy reduces the search grid after each iteration and guides nodes to the nearest point of convergence by "attracting" them towards each other. This design is particularly advantageous in environments where the ideal merging point is not centrally located, a common weakness in traditional bidirectional search algorithms [29]. In such cases, conventional approaches often expand unnecessary nodes near the centre, missing earlier convergence opportunities. LiteRBS, by contrast, prioritises convergence at the likely intersection point, regardless of its position within the grid. As a result, the algorithm significantly reduces the explored area and improves efficiency, particularly in asymmetric or obstacle-dense environments where central merging is impractical.

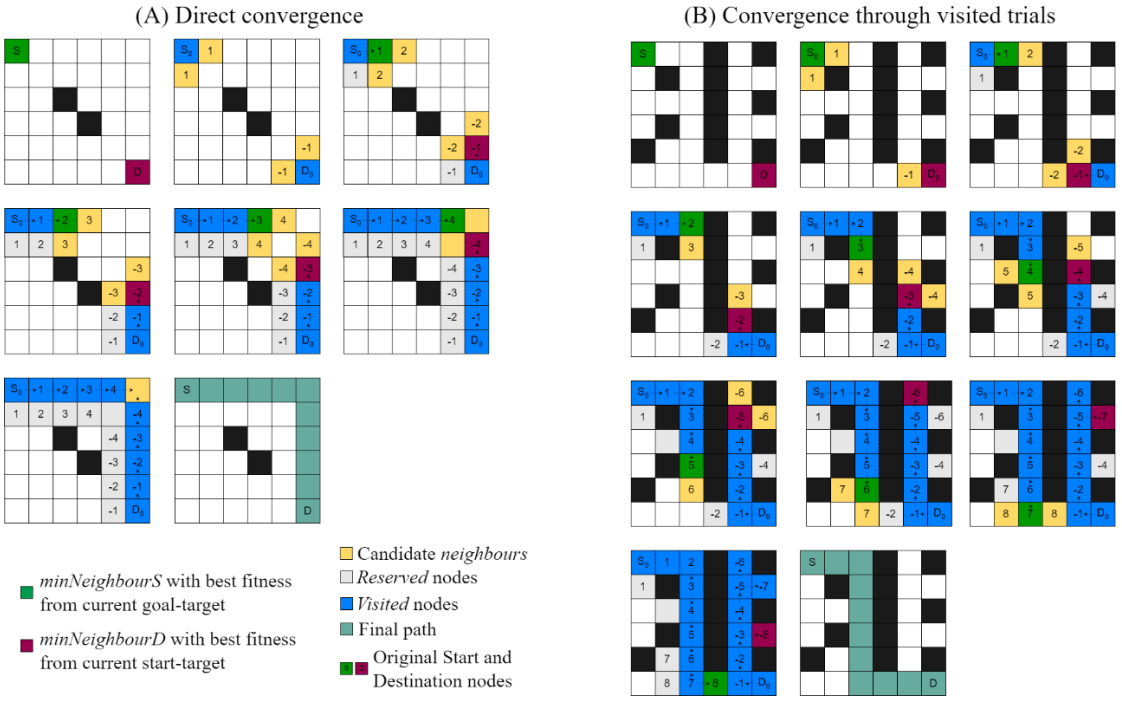


Figure 2. The convergence behaviour of the LiteRBS algorithm, where forward and backward search frontiers intersect either directly **(A)** or by inspecting the visited nodes of the counterpart **(B)**.

2.1.1. Time complexity

The LiteRBS algorithm performs a simultaneous bidirectional search guided by an admissible heuristic such as the Manhattan or Euclidean distance. We denote:
b: branching factor, where $b = 4$ (4-connected graph), or $b = 8$ (8-connected graph)
d_s, d_g: search depths from the start and goal nodes, respectively
d: total solution depth ($d \approx d_s + d_g$)

Each direction of the algorithm expands at most by a branching factor b, since each node has at most b edges to adjacent nodes. Each node is expanded at most once, as the algorithm maintains visited lists to prevent re-expansion; that is, each node is inserted and removed exactly once.

The maximum number of nodes expanded in each direction up to the depth in that direction is approximately $O(b^{d_s})$ and $O(b^{d_g})$, assuming asymmetrical search. Hence, the total worst-case complexity of the algorithm is:

$$O(b^{d_s}) + O(b^{d_g}) = O(b^{\max(d_s+d_g)}), \text{ where } \max(d_s + d_g) \geq \frac{d}{2} \quad (1)$$

If the admissible heuristic used by the algorithm guides the effective search depths symmetrically to $\frac{d}{2}$, the time complexity approaches:

$$O\left(2 \cdot b^{d/2}\right) = O(b^{d/2}) \quad (2.1)$$

However, the LiteRBS algorithm expands a single best-fit neighbour at each step, falling back to reserves when no immediate neighbours are available. Hence, in the best case, with $b \approx 1$, the algorithm behaves approximately linearly with respect to the search depth, exploring only a narrow path toward the goal up to depth $\frac{d}{2}$ per direction. The number of nodes expanded satisfies:

$$O\left(b \cdot \frac{d}{2}\right) \approx O\left(1 \cdot \frac{d}{2}\right) = O(d) \quad (2.2)$$

2.1.2. Space complexity

The space complexity involves the storage required to maintain $visitedList[i]$, $reserveList[i]$ and $queue[i]$, for $i \in \{0, 1\}$, *i.e.*, in each direction.

Given that a node marked as visited is never reconsidered for exploration to avoid redundant expansions and infinite loops, there are no duplicates between the $queue[i]$ and $visitedList[i]$. Similarly, a node is added to a $reserveList[i]$ only if it does not already exist in any of the $queue[i]$, $visitedList[i]$ or $reserveList[i]$ of either direction.

Using the same notations as for the time complexity, in the worst case, the algorithm stores the total number of active nodes, and the maximum frontier space comes from an exponential growth at depth $\frac{d}{2}$:

$$O\left(b \cdot \frac{d}{2}\right) \approx O\left(1 \cdot \frac{d}{2}\right) = O(d) \quad (3)$$

The worst-case complexity assumes that the algorithm will have to fall back to all possible reserves and explore all possible paths up to half the solution depth. However, LiteRBS typically falls back to reserves only when necessary, thus reducing actual expansions. Hence;

$$reserve_{size}\ (per\ directon) \leq (b-1) \cdot \frac{d}{2} \tag{4.1}$$

Hence, the best-case complexity of LiteRBS is approximately linear in depth:

$$Total\ space \leq 2 \cdot (\frac{d}{2} + (b-1) \cdot \frac{d}{2}) = O(b \cdot d) \tag{4.2}$$

2.2. Validation protocol

The performance of the LiteRBS algorithm is assessed in comparison to the baseline algorithms: Lee's algorithm, A*, bidirectional A*, Jump-Point Search (JPS), and Shortest Path Faster Algorithm (SPFA). The A* algorithm and its bidirectional variant used in these evaluations employ a heap data structure to reduce their computation time. Each test evaluates different aspects of algorithmic behaviour based on the following objective functions: (i) path length/size; (ii) computation time; (iii) expanded nodes; and (iv) peak Random Access Memory (RAM) during computation. To ensure a thorough evaluation, extensive maps are generated for each test.

2.2.1. Optimality test

This test compares the algorithms based on their ability to optimise one or more objective functions listed above. The Recursive Division algorithm (RDA) is employed to generate a total of 99,998 random and unique maps with multiple potential paths. The optimality test reveals how each algorithm performs on average under varying obstacle densities.

2.2.2. Robustness test

While the optimality test measures overall performance, the robustness test evaluates how the algorithms scale up with increasing obstacle density. Obstacles are randomly introduced, starting at 1% of the nodes and incrementally rising to 30%. The 30% cap is chosen to prevent excessive obstacle density around the start and goal nodes. 1,000 unique maps are generated for each density level, totalling 30,000 test cases.

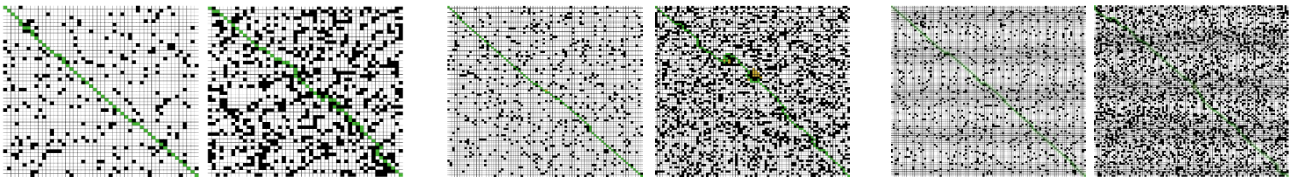
2.2.3. Convergence analysis

In this analysis, we empirically evaluate the LiteRBS algorithm following a format similar to the robustness test. It examines different scenarios in which the LiteRBS algorithm locates the merge point: (i) when the start and goal nodes converge directly towards each other, and (ii) when an intersection point is established using a frontier of visited traces if direct convergence is not feasible. The test is conducted across various obstacle density levels, each level using 1,000 unique maps to identify the conditions under which the merging method is most likely employed. This test provides an empirical evaluation of the algorithmic completeness.

3. Results

The results of all tests are reported in square mazes of dimensions 50, 80 and 100. In all maps, the start node is located in the farthest top-left corner, and the goal node is in the bottom-right corner that can be

found. The maps exhibit considerable complexity, especially those with high obstacle density, with multiple dead-end routes and U-turns (Figure 3). For a fair comparison, all algorithms were evaluated in 8-connected grid maps, with possible propagation directions up, down, left, right and along the four diagonals. Each test evaluates the algorithms on an identical set of pre-generated maps, randomised in terms of obstacle density, for each map size.



50 × 50
10% (left), 30% (right)

80 × 80
10% (left), 30% (right)

100 × 100
10% (left), 30% (right)

Figure 3. Generated mazes with different obstacle densities: 10% (left) and 30% (right).

The simulations are performed on a 12th Gen Intel® Core™ i5-1235U×12, 4.40 GHz laptop with 15.3 GiB RAM, Ubuntu 20.04.

3.1. Optimality test

Table 1 summarises the performances of the algorithms averaged across 99,998 unique maps per dimension (50, 80, 100), randomly generated with varying obstacle density. For 100 × 100 maps, A* and Lee were excluded due to their significantly slower performance, substantially increasing computation effort without providing additional meaningful insights.

Table 1. Summary of simulation results for the optimality test. The values represent the average performance across 99,998 maps per grid size. Highlighted values indicate statistically significant best performance in each metric.

Maps	Algorithm	Expanded Nodes	Computation Time (ms)	Path Size	Peak Memory Usage (Bytes)
50 × 50	LiteRBS (ours)	**178**	**0.48**	55	**7,895**
	Lee's algorithm	1750	80.4	**52**	143,701
	A*	497	4.15	**52**	61,357
	Bidirectional A*	298	2.90	**52**	64,450
	JPS	236	3.26	**52**	34,108
	SPFA	1764	7.86	**52**	253,398
80 × 80	LiteRBS (ours)	**291**	**0.80**	89	**13,440**
	Lee's algorithm	4545	505.34	**85**	342,120
	A*	1287	12.70	**85**	311,336
	Bidirectional A*	765	18.65	**85**	239,196
	JPS	589	9.52	**85**	64,632
	SPFA	4564	19.61	**85**	770,952
100 × 100	LiteRBS (ours)	**370**	**1.66**	112	**15,919**
	Bidirectional A*	1200	29.43	**107**	77,473
	JPS	994	26.73	**107**	118,096
	SPFA	7155	52.57	**107**	1,327,024

The results reveal that the LiteRBS algorithm outperforms the baseline across several key metrics. In particular, it operates within a smaller search space, *i.e.*, expanded nodes, which directly affects the algorithmic computational overhead in terms of time and memory utilisation. The LiteRBS algorithm discovers a viable path in an average time that is negligible, using approximately 4–5 times less memory than its closest competitors, bidirectional A* or JPS. Such fast reaction times are crucial when robots operate in dynamic environments with shifting obstacles, enabling quick and frequent recomputation of alternative paths. Swift recalibration may enable robots or autonomous driving systems to respond promptly to unexpected obstacles, ensuring operational stability and the safety of the robot and its surroundings [30]. In addition, low memory overhead can be vital when the available memory is limited or for robots to maintain essential functionalities without resource depletion.

These results were significant across all maze dimensions. The Wilcoxon paired test revealed p-values < 0.001 for all metrics when comparing the LiteRBS algorithm against each competitor. They confirmed that LiteRBS performed significantly faster, using less space and memory than all other algorithms.

However, the LiteRBS algorithm produced slightly longer average paths than competitors, which may be disadvantageous in scenarios where the shortest path is crucial. Despite this mean deviation from the optimal path being relatively small (3–5 nodes), it was statistically significant across all maze sizes ($p < 0.001$), with a large effect (Cliff's Delta ≥ 0.7), meaning that the LiteRBS tended to produce sub-optimal paths more often.

We measured the sub-optimality bounds for all grids, comparing the paths produced by LiteRBS and an optimal baseline (e.g., A*) over 100,000 iterations. If we denote as C^* the cost of the shortest (optimal) path between the start node (s) and the goal node (g), and $C_{LiteRBS}$ the path returned by the LiteRBS algorithm, the suboptimality is then calculated as:

$$\alpha = \frac{C_{LiteRBS}}{C^*}$$

Table 2 presents the median sub-optimality bound and the proportion of paths produced by LiteRBS that lie within 10% of the optimal path ($\alpha = 1.10$). This threshold aligns with previous work on bounded suboptimality in related pathfinding literature, which uses Euclidean distance heuristics on planar maps [31], here applied to discrete grid approximation.

Table 2. The sub-optimality bound of the LiteRBS algorithm across map sizes.

Maps	C^* (median)	$C_{LiteRBS}$ (median)	α (median)	$\alpha \leq 1.10$
50 × 50	52	55	1.04	93.23%
80 × 80	85	89	1.035	94.57%
100 × 100	107	112	1.038	95.18%

The results in Table 2 suggest that LiteRBS consistently generated paths close to optimal, with a median around 1.04, *i.e.*, paths 4% longer than the optimal and over 93% of paths within the 10% sub-optimality bound for all tested grid sizes. This highlights that, despite some loss in optimality, LiteRBS constitutes a competitor choice in many practical robotic applications that may tolerate near-optimal paths.

Focusing on LiteRBS' computational efficiency, we conducted an additional controlled experiment, in which we constrained path variability and forced the algorithms to discover the same solution path. We employed Prim's algorithm to generate maps with a single viable path, allowing us to evaluate

algorithmic overhead under equal conditions. We measured the computation time, peak memory usage and number of expanded nodes for each algorithm. The evaluation was performed exhaustively across 99,998 procedurally generated maps per grid size (50, 80, 100) with varying obstacle densities. The mean results are illustrated in Table 3. The analysis emphasises the search overhead introduced by each algorithm and the impact of map size.

The LiteRBS algorithm demonstrated the lowest computational overhead, *i.e.*, time, peak memory, required to identify the same solution compared to the baseline. The results were statistically (Wilcoxon $p < 0.001$) and practically significant (Cliff's Delta ≤ -0.994), indicating a consistently large effect in favour of LiteRBS. Bidirectional A* expanded the fewest nodes overall, since in single-solution configurations it benefits from minimal side exploration, generating a smaller search space than in open, multi-path maps. Despite the narrower expansion, bidirectional A* did not outperform LiteRBS in runtime and used significantly more memory across all grid sizes. This suggests that LiteRBS is well-suited for practical scenarios where computational efficiency is critical, such as real-time or low-resource navigation systems.

Table 3. Summary of algorithmic performance results on maps with only one viable path. The highlighted values indicate statistically significant best performance for each metric.

Maps	Algorithm	Expanded Nodes	Computation Time (ms)	Peak Memory Usage (Bytes)
50 × 50 ($mean_{path}$, 79)	LiteRBS (ours)	372	**2.7**	**21,956**
	Lee's algorithm	914	20.1	79,736
	A*	404	3.3	65,563
	Bidirectional A*	**215**	**2.7**	49,319
	JPS	344	4.2	47,290
	SPFA	1032	5.4	103,626
80 × 80 ($mean_{path}$, 129)	LiteRBS (ours)	785	**5.6**	**45,867**
	Lee's algorithm	2413	106.15	252,260
	A*	1019	7.3	173,288
	Bidirectional A*	**477**	6.99	138,853
	JPS	871	8.97	114,778
	SPFA	2752	12.6	486,308
100 × 100 ($mean_{path}$, 162)	LiteRBS (ours)	1116	**8.4**	**67,129**
	Bidirectional A*	**696**	9.9	198,318
	JPS	1351	23.95	106,379
	SPFA	4358	36.9	768,710

3.2. Robustness test

The Robustness Test is designed to assess the stability of the algorithm under various environmental conditions and spatial complexity. Specifically, it assesses how effectively the algorithm optimises the objective functions in sparse or heavily obstructed environments. The test is performed as follows:

(1) A map is generated with an initial obstacle density of 1%.

(2) The proposed pathfinding algorithm and baseline are tested on the obstacle-laden map.

(3) Step 1 was repeated with 1,000 unique and randomly generated maps with the same obstacle density.

(4) The obstacle density was incremented by 1%, and steps 1–3 were repeated until a 30% density level, inclusive.

In the following graphs, each objective function is plotted against obstacle density to assess the algorithmic robustness in maps of increasing size (50, 80, 100). Lee's algorithm and A* were excluded from this comparison in maps of 100 × 100, since their results distorted the visual scaling of the plots, rendering the differences between the other algorithms more difficult to interpret.

3.2.1. Path size or length

Path size, or path length, quantifies the number of nodes from start to destination, inclusive.

Consistent with the optimality test, Figure 4 shows that LiteRBS tends to produce suboptimal paths, in particular as the density increases. This behaviour can be explained by the algorithm's heuristic propensity to minimise the time to discover a path. The LiteRBS algorithm terminates immediately upon finding a viable path, without evaluating if it is the best possible route. While this approach may not guarantee the shortest path as the baseline, it significantly reduces runtime even with increasing environmental noise and spatial complexity, while still maintaining path lengths comparable to other methods (suboptimality bound, Table 2).

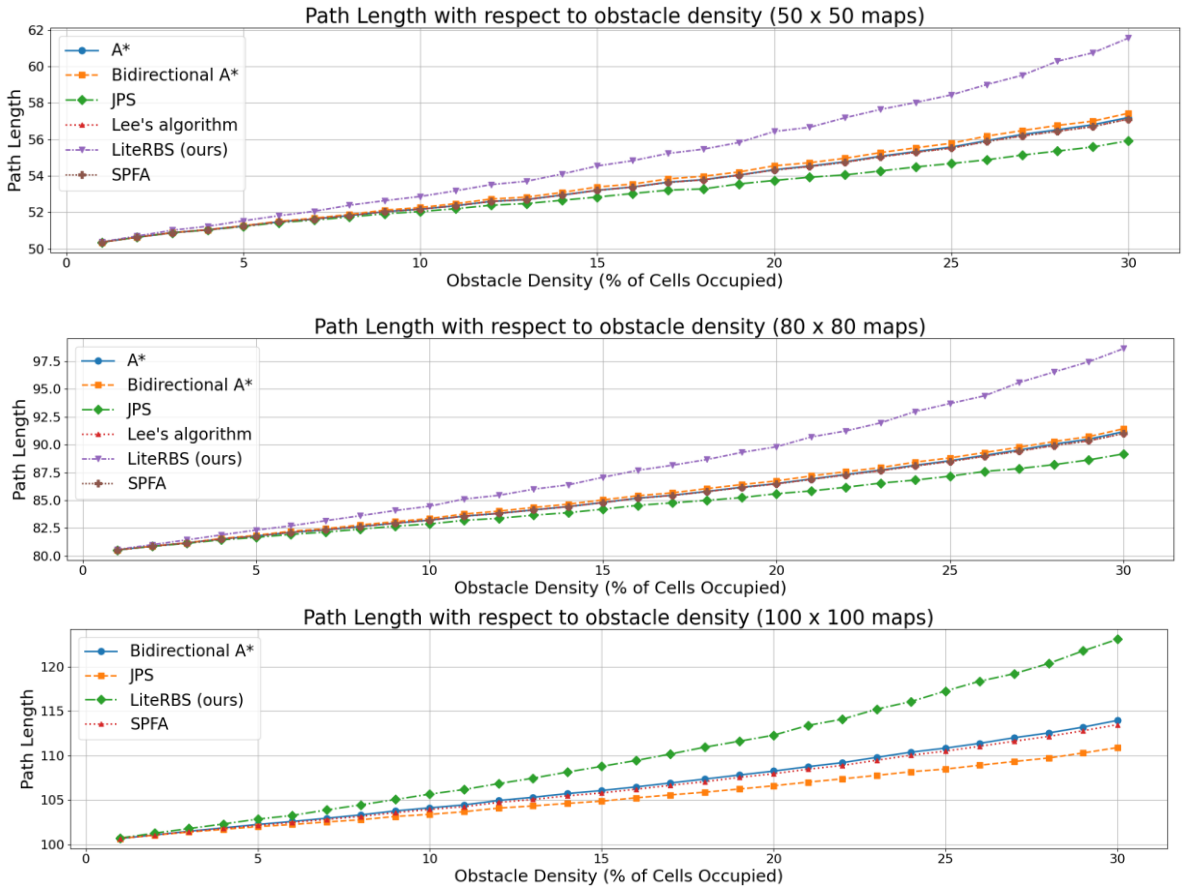


Figure 4. Path size relative to obstacle density in grids of size 50, 80, 100. LiteRBS produces longer paths as the density increases due to its early termination strategy.

It is important to consider that while Lee's algorithm, A*, bidirectional A* and SPFA overlap in the plots due to producing identical paths, JPS exhibits a slight difference. This is because, in these simulations, JPS uses the octile distance, which approximates the Euclidean distance used by the other heuristic-guided algorithms in 8-connected grids. This difference is typically less than 0.1 units (1%), but can lead to small

variations over longer paths, which are more evident with increasing obstacle density. However, the results of the optimality test showed that this difference fades over 100,000 iterations, and the average path of JPS converges to that of the other algorithms (Table 1).

3.2.2. Computation time

Computation time is the time elapsed from the start of the algorithm until it successfully returns a path. It excludes the time spent generating the maps and focuses solely on the algorithm's runtime searching for a path.

Figure 5 shows that the LiteRBS algorithm consistently achieves the fastest computation times across increasing obstacle density compared to the baseline. Although the plots show subtle differences between the algorithms, except Lee's algorithm, the Wilcoxon signed-ranked test produced a statistic of 0 with a p-value < 0.0001, indicating that nearly all paired samples for the LiteRBS algorithm yielded lower computational times. The median runtime (~0.00079) was substantially lower compared to the baseline. The Kolmogorov-Smirnov test (statistic 1, p-value < 0.0001) also confirmed very strong distributional differences. The statistically significant advantage gap of LiteRBS over other algorithms, like JPS and bidirectional A*, becomes more pronounced in larger maps (100 × 100) and increasing environmental noise, demonstrating robust scalability.

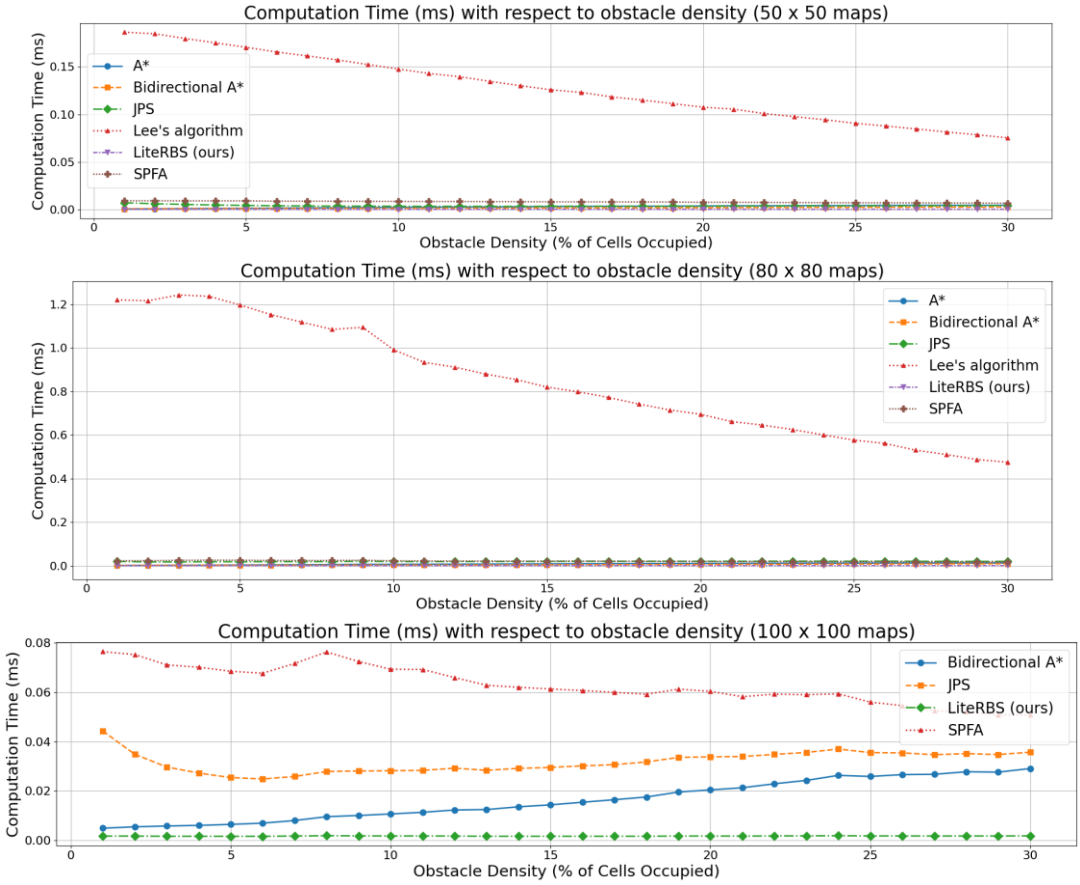


Figure 5. Computation time relative to obstacle density in grids of size 50, 80 and 100. The LiteRBS algorithm outperforms the baseline, demonstrating robust scalability under increasing spatial complexity and environmental noise.

3.2.3. Expanded nodes

The number of expanded nodes denotes the degree of exploration or redundancy in the algorithm's pathfinding process. It represents the total number of nodes the algorithm has considered during execution, which did not contribute to the final path, as shown in equation (5).

$$\text{expandedNodes} = \text{totalNodes} - \text{pathSize} \tag{5}$$

Figure 6 illustrates the relationship between the search space, *i.e.*, expanded nodes and environmental noise for all algorithms in 50, 80 and 100-size maps. The LiteRBS algorithm consistently visits a minimal number of nodes across increasing densities. While it is slightly outperformed by JPS and Bidirectional A* in sparse and small maps (50 × 50), our algorithm demonstrates statistically significant stability in node expansion as the environment scales, outperforming the baseline. In particular, it maintains a flat and density-invariant node expansion profile, where other algorithms' overhead increases in larger and cluttered maps. Maintaining a low search space under varying environmental conditions becomes more critical when scaling to larger or more complex environments, like robotics, autonomous vehicles, or large-scale simulations in real-world applications. As problem size increases, e.g., with more vehicles or requests, high search complexity, typically NP, leads to excessively long computation times, highlighting the need for efficient algorithms to find viable solutions quickly across different environments [14]. Our LiteRBS algorithm suggests a promising, scalable solution, with consistent performance as the problem size increases.

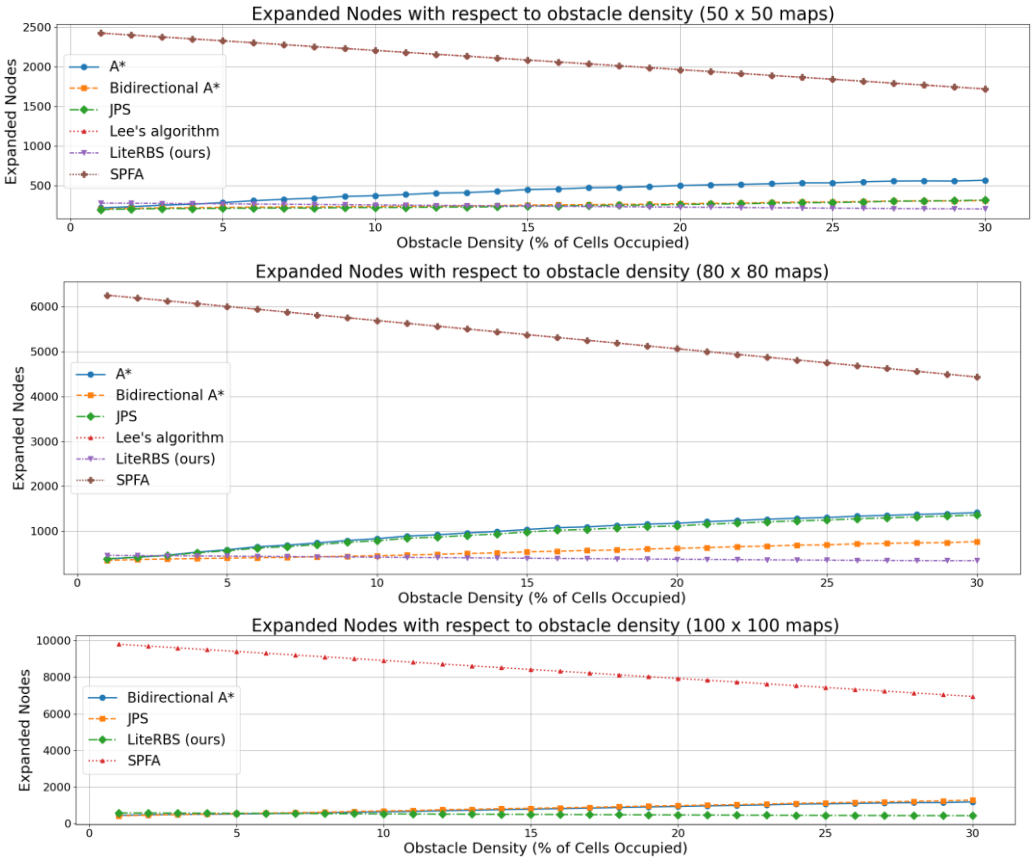


Figure 6. Number of expanded nodes relative to obstacle density in grids of size 50, 80 and 100. The LiteRBS algorithm maintains a flat and density-invariant node expansion, outperforming the baseline, particularly as the environmental noise and spatial complexity scale.

3.2.4 Peak memory usage

Peak memory usage computes the maximum amount of computer memory consumed during the execution of the algorithm. We used the ‘tracemalloc’ function from the Python library to calculate memory usage.

Figure 7 illustrates the memory usage of the pathfinding algorithms. The proposed LiteRBS algorithm significantly outperforms the baseline, demonstrating the least memory consumption among all algorithms. In particular, memory usage of LiteRBS remains flat across densities and with increasing map size, confirming its lightweight design. This efficiency is attributed to its lower node visitation and reduced runtime from the early termination strategy. These findings underscore the practical advantages of LiteRBS for mobile robot design and memory-limited operation, where memory efficiency is essential due to potential hardware constraints, especially in rapidly changing environments where frequent rerouting may be necessary.

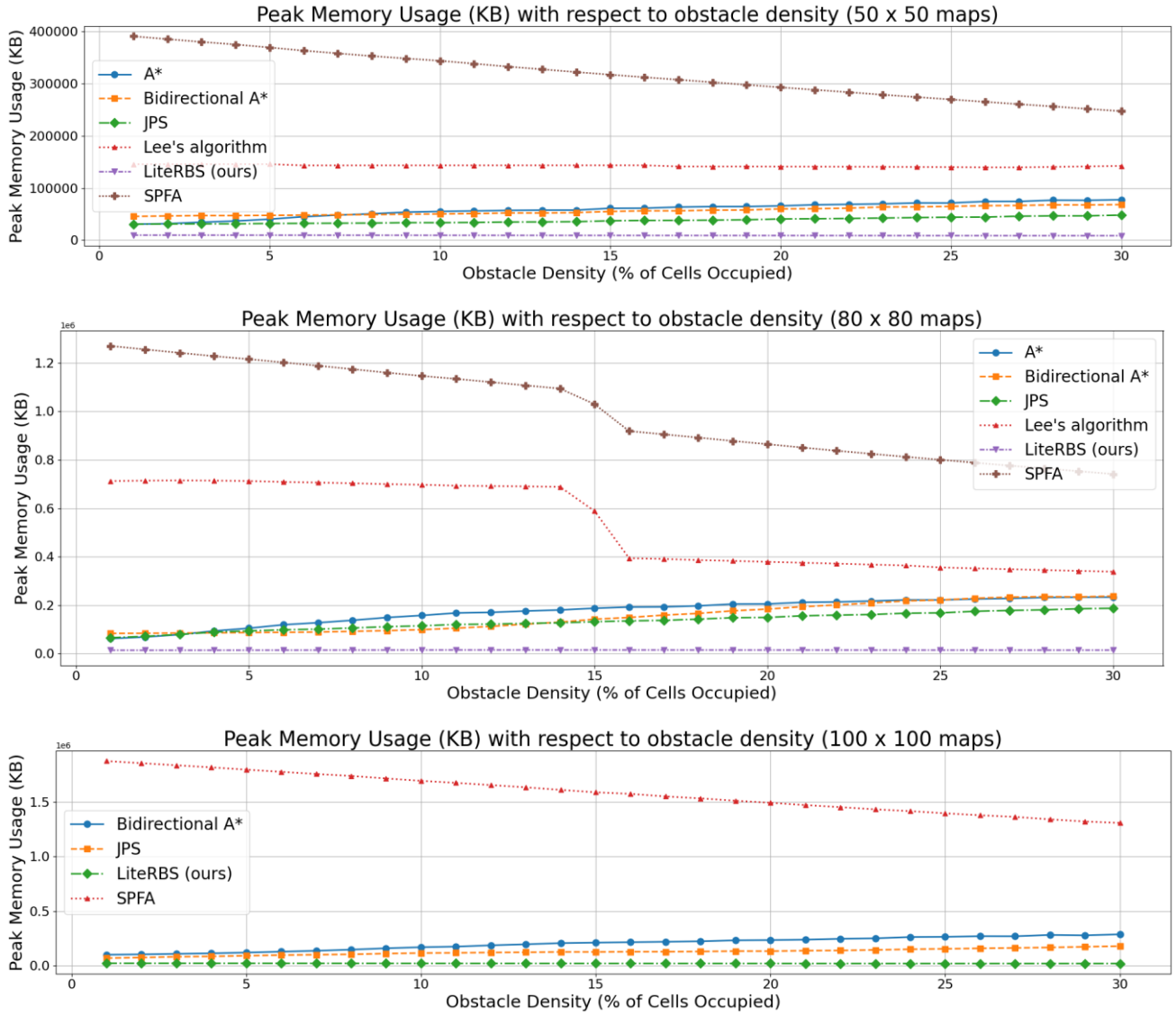


Figure 7. Peak memory usage relative to obstacle density in grids of size 50, 80 and 100. The LiteRBS algorithm maintains flat and low memory consumption compared to the baseline, demonstrating robustness to environmental and spatial complexity.

A live visualisation of the simulation outcomes comparing LiteRBS to the baseline in maps of varying sizes (50, 80, 100) with 10% obstacle density is shown in Figure 8. The straight near-diagonal lines (green) represent the paths discovered by each algorithm on the same map, and the coloured nodes (amber) represent the frontier that is expanded during the search. Additional simulations across variable obstacle densities and map sizes can be found in the Supplemental Materials.

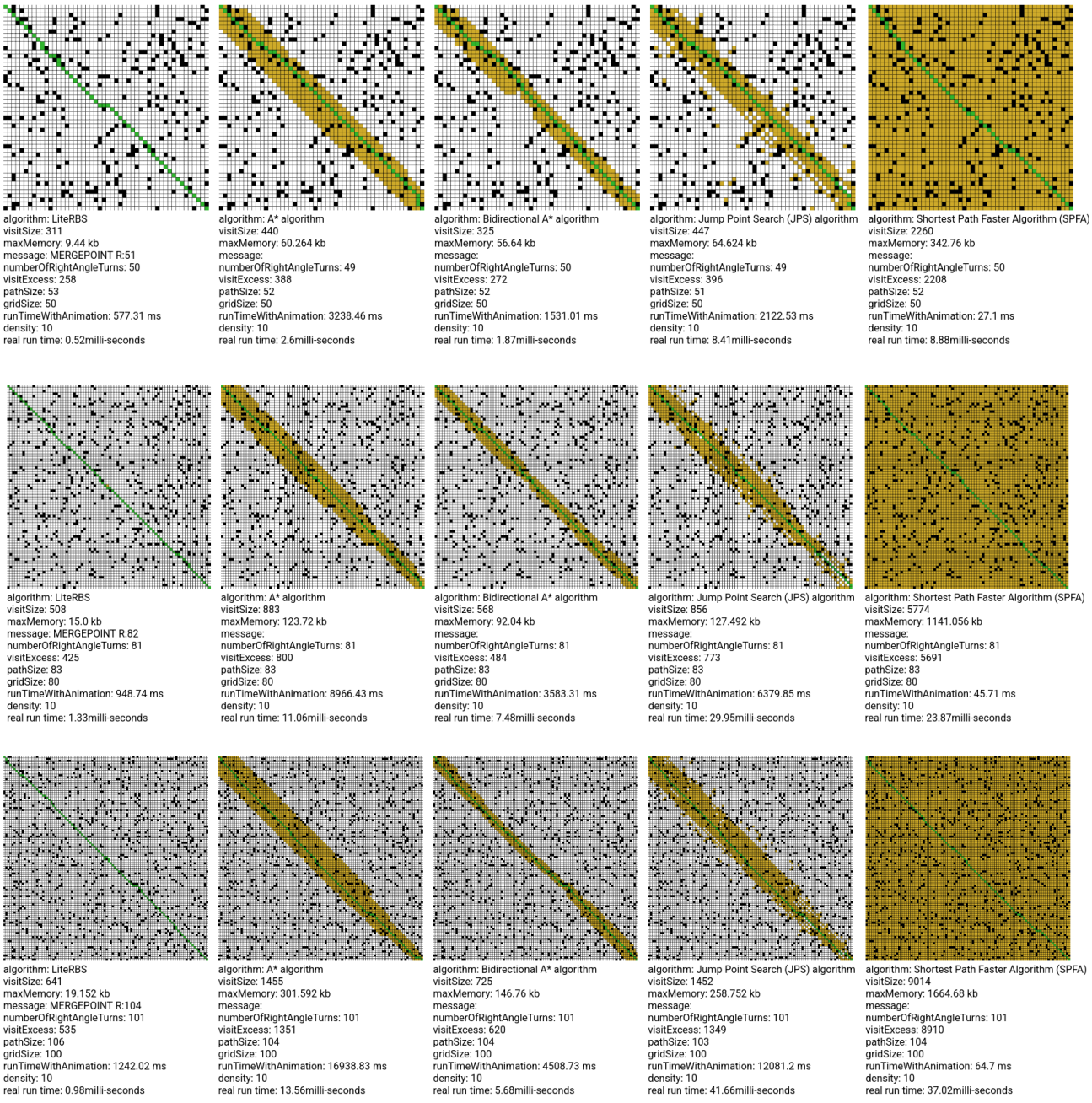


Figure 8. A live visualisation of the performance of our proposed LiteRBS pathfinding algorithm against the baseline (A*, Bidirectional A*, JPS, SPFA) in 50, 80 and 100-grids (top-down) with 10% obstacle density. The path is indicated by the near-diagonal line (green), while the examined nodes are the coloured frontier (amber).

3.2.5. Algorithmic scalability

Figure 9 provides a scalability analysis of our LiteRBS algorithm under increasing spatial and environmental complexity. For the scalability metric, we focused on the computational cost, evaluated in terms of computation time and peak memory usage. We present the results in maps of size 50, 100, 150, 500 and 1000. The 500 × 500 grid size reflects a practical upper bound for many real-world terrestrial navigation tasks, particularly in large indoor spaces or constrained outdoor environments. Instead, while the 1000-grid exceeds the resolution typical to robotic deployments, we included it in the experiment to stress-test the algorithm's scalability.

The two plots indicate that for small to medium maps up to 150, our LiteRBS algorithm maintains low and nearly constant computation times (< 1.6 ms) with minimal sensitivity to obstacle density. For larger grids, the growth in computation time is gradual, without notable peaks and scales linearly to sub-linearly. Similarly, memory usage remains stable and broadly proportional to map size with limited sensitivity to density. The algorithm demonstrates controlled growth in computation time and memory footprint, which is desirable for low-resource robotic path planning.

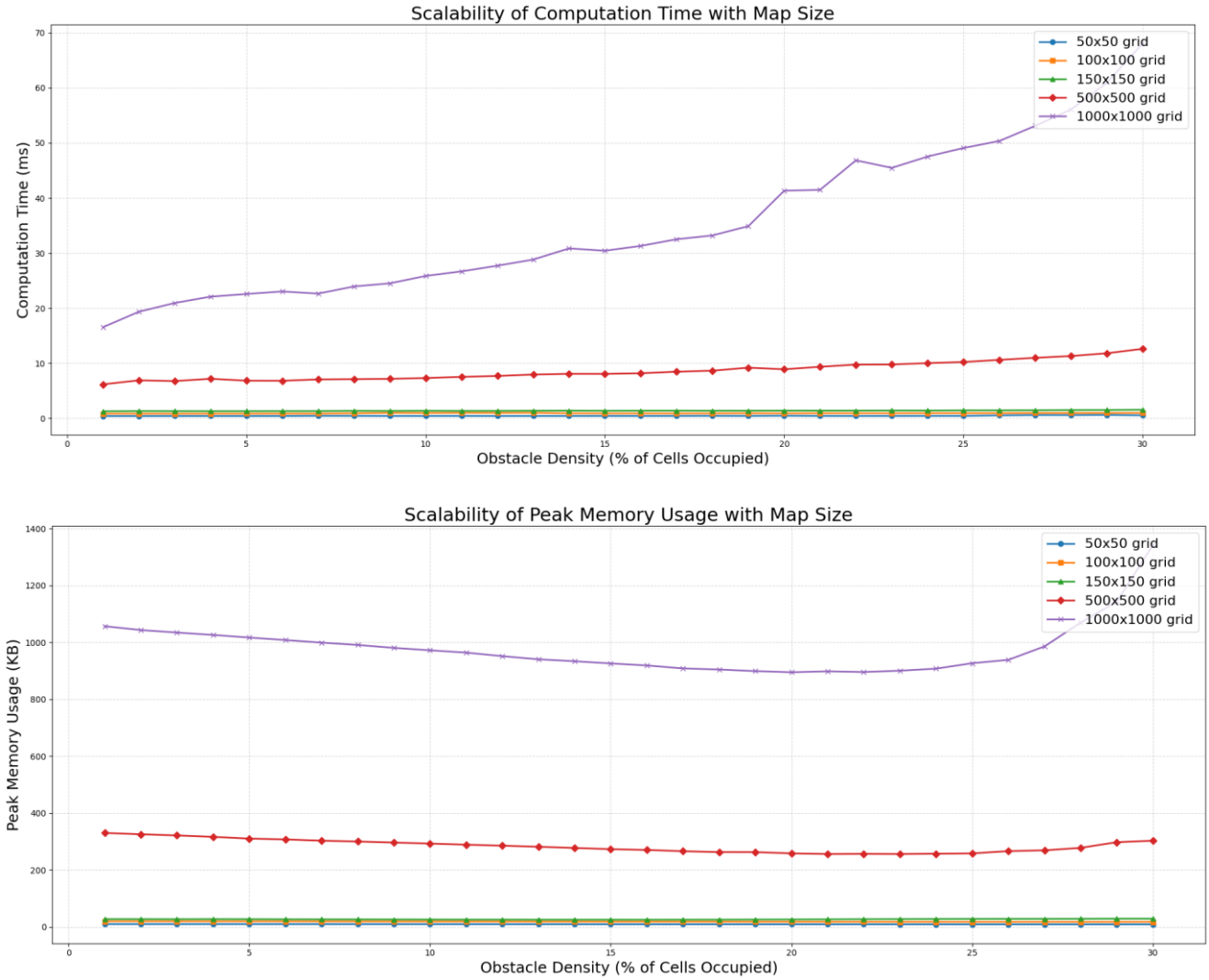


Figure 9. Scalability analysis of the LiteRBS algorithm across varying map sizes (50 × 50 to 1000 × 1000). Plots show computation time (up) and peak memory usage (down), demonstrating stable growth in computational effort as spatial resolution and environmental complexity increase.

3.3. Convergence analysis

3.3.1. Mathematical proof

Let $s \in V$ and $g \in V$ be the start and goal nodes in our grid, denoted as $G = (V, E)$. A path exists when there is a sequence of adjacent nodes between s and g, such that $P = s \to n_1 \to n_2 \to \cdots \to g$.

Given the notations in Algorithm 1, regarding $VisitedList[i]$, $queue[i]$, $reserveList[i]$, $currentNode[i]$, for $i \in \{0, 1\}$, the frontiers meet when:

$$\exists (x, y) \in \text{VisitedList}[0] \cap \text{VisitedList}[1]$$

Proof of completeness (by contradiction)

Assumption (Completeness Hypothesis)
There exists a valid path P from start to goal. That is, s and g lie in the same connected component of G.

$$\exists\, P = \{n_0 = s, n_1, \ldots, n_k = g\} \subseteq V \mid (n_j, n_{j+1}) \in E, \forall j$$

Suppose, for contradiction
The algorithm fails to generate a path (path not found), i.e., frontiers do not meet.
THEN, both of the following must be *TRUE*
The frontiers never intersect:

$$\text{VisitedList}[0] \cap \text{VisitedList}[1] = \emptyset$$

AND at least one direction exhausts its entire search space, including reserves:

$$\exists i \in \{0, 1\} \mid \text{queue}[i] = \emptyset \wedge \text{reserveList}[i] = \emptyset$$

However, by its construction (Algorithm 1), LiteRBS's fallback to reserves is guaranteed and exhaustive. Nodes are never pruned permanently from the frontier unless they are already visited or unreachable. As a result, all reachable nodes from each end are eventually explored. Hence, if a path exists, the algorithm must reach a state where:

$$\text{VisitedList}[0] \cap \text{VisitedList}[1] \neq \emptyset \vee \text{distance}(\text{currentNode}[0], \text{currentNode}[1]) \leq 1$$

By reductio ad absurdum, the assumption that the algorithm fails when a valid path exists leads to a contradiction. Therefore, if a path exists in the grid, the LiteRBS algorithm is guaranteed to find it, making it a complete algorithm.

3.3.2. Empirical proof

We empirically assessed the merging behaviour of the LiteRBS algorithm under varied environmental conditions. This test, therefore, examines how and where merging occurs to achieve completeness and how these merging patterns relate to the observed reductions in computation time.

The results in Figure 10 correspond to the LiteRBS algorithm with the Euclidean distance in 8-connected grids of varying sizes. The plots present empirical validation of the algorithm's completeness. Specifically, in sparse maps, the algorithm converges primarily by merging directly, where the two frontiers meet head-on. Instead, in cluttered maps, convergence is predominantly achieved by trail-based

merging, where either frontier expands into the nodes previously visited by the other. This is consistent with the algorithm's design to terminate early and reduce computation time.

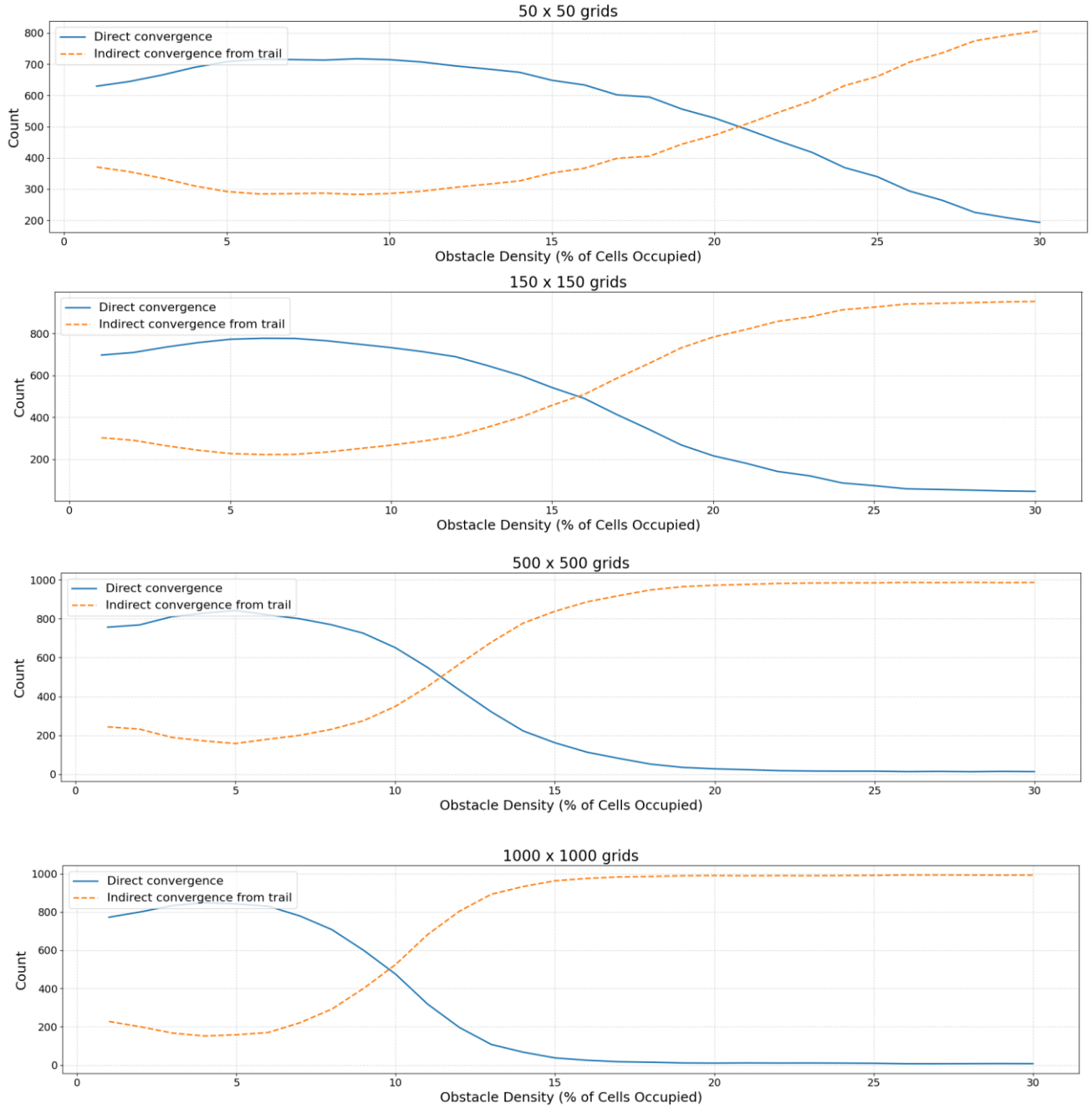


Figure 10. Empirical validation of completeness. Across mazes of increasing size (50, 150, 500, 1000), the curves remain symmetrical, with the total number of convergences (direct, indirect) constant (1,000) at each obstacle density (1%–30%), indicating successful completion of the algorithm. Direct merging (blue, continuous) typically occurs in low-density maps, while merging from trails of visited nodes (orange, dashed) takes precedence in densely packed maps. There is a clear transition point, which shifts left with larger grids, meaning the algorithm dynamically adjusts its strategy consistently with the environment and earlier in larger grids due to increased complexity.

Importantly, the results show a symmetrical distribution between direct and indirect convergence types across obstacle density, with every iteration resulting in successful convergence, totalling 1,000

attempts at each density level. The algorithm switches strategy to fall back correctly, intersecting at a "critical" density, as the completeness proof predicts. This tipping point shifts left with increasing grid dimensions, but the symmetry is maintained, suggesting that in larger environments, it leverages its fallback strategy earlier than in smaller ones to find a viable path in the least amount of time. This demonstrates that the completeness guarantee holds across a wide range of spatial and environmental complexity, *i.e.*, the algorithm is empirically complete.

4. LiteRBS algorithm on a mobile robot

In real-world scenarios, mobile robots function in environments where complete global information is frequently inaccessible due to the robot's restricted sensor range, posing challenges to the algorithm's path-planning process. This section briefly illustrates the implementation and behaviour of the LiteRBS algorithm on a real-world mobile robot under such conditions.

We deployed the algorithm on a Turtlebot3 Waffle robot with dimensions of 281 mm (L), 306 mm (W), and 141 mm (H), which is a common benchmark for indoor navigation tasks. We discretised the environment using a block size of 0.1 and an area size of 10, resulting in a lattice-like map of 100x100 in size. This map resolution is among the largest evaluated in our simulations (section 3) and relatively large for Turtlebot3 deployments, which are typically carried out in smaller lab testbeds. The implementation choices were as follows. To maintain symmetry and ensure consistency in the calculations with grid coordinates, we tracked the sign of each coordinate, floored the absolute value and restored the original sign afterwards. Precise calibration of the robot's positioning on the physical map was achieved using SLAM to generate odometry and rotational information necessary for the path calculations. The robot's turning angles were computed around a designated pivot point, and a damping value of 0.2 was applied to reduce abrupt changes in direction.

A live demonstration of the Turtlebot3 robot's algorithmic behaviour is available in the Supplemental Materials. The video shows the robot navigating a constrained indoor space under partial observability. The robot calculates paths quickly and dynamically reacts to sudden changes in its space, such as recalculating a new route when an unexpected obstacle blocks the original path. This demonstrates the algorithm's ability to operate reactively, which has important considerations in real-world conditions where assumptions of complete or static knowledge of the environment do not hold.

While the 100 × 100 map provides a reasonably large navigational space, in which the performance of baseline algorithms degraded, with LiteRBS scaling well even in larger map resolutions, further work is needed to investigate its deployment in larger or more complex real-world (including 3D) environments.

5. Conclusion

This paper proposes the LiteRBS, a novel and complete grid-based bidirectional pathfinding algorithm for terrestrial mobile robots. LiteRBS achieves high computational efficiency by maintaining minimal node expansion, low memory overhead and fast runtimes. It significantly outperforms established algorithms like A*, bidirectional A*, Jump-Point Search (JPS) and Shortest Path Faster Algorithm (SPFA) across a wide range of map sizes and obstacle densities, demonstrated through extensive simulations. These advantages were maintained consistently as the problem scaled, including in maps up to 1000 × 1000 in size. It produced over 93% of paths within a 10% sub-optimality bound. The algorithmic dynamic node

convergence and fallback strategy support robust scaling, maintaining flat and density-invariant computational overhead. While a small trade-off in path optimality exists, LiteRBS offers a lightweight and scalable alternative, particularly suited for time-sensitive or resource-constrained robotics navigation. In this work, we also demonstrated the adaptability of the algorithm by deploying it on a physical robot, which navigates in a partially observable and obstacle-dynamic space.

6. Recommended future work

A recommended direction for future work is to extend the comparative evaluation to larger grid sizes and incorporate an expanded baseline representative of modern approaches. In the present study, we limited full comparisons to 100 × 100 grids against well-established methods, in line with current literature, to avoid the prohibitive computational overhead of running exhaustive or near-exhaustive algorithms such as Lee, A* and SPFA on very large dense grids. Preliminary results showed that our algorithm continues to scale effectively at 500 and 1000 grids, and a more comprehensive direct comparison across all baselines at these larger scales would be valuable in future investigations. Another interesting direction is to explore how the algorithm might perform in more complex environments, such as 3D maps. Further work is also needed to evaluate performance in large-scale, real-world robotic deployments, to ensure broader applicability beyond the controlled small-scale and low-complex setup used in this study.

Supplementary data

The authors confirm that the supplementary data are available within this article and can be accessed at https://github.com/literbs/literbs/.

Data and code availability

The source code accompanying this paper is available at https://github.com/literbs/literbs/.

Authors' contribution

MB—Investigation, methodology, software, formal analysis. IG—Investigation, validation, formal analysis, visualisation, writing original draft, review & editing. GLM—conceptualisation, formal analysis, validation, supervision, writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Conflicts of interests

The authors declare no conflict of interest.

List of Acronyms

Acronym	Full Term
LiteRBS	Lightweight and Rapid Bidirectional Search
A*	A-Star heuristic search algorithm
JPS	Jump Point Search
PO-JPS	Polygon pruning optimised JPS
SPFA	Shortest Path Faster Algorithm
RRT	Rapidly-exploring Random Tree
SLAM	Simultaneous Localisation and Mapping
RDA	Recursive Division Algorithm
RAM	Random Access Memory
LIFO	Last-In, First-Out

Symbols and Variables

Symbol	Description
b	Branching factor of the grid (e.g., 4 or 8 connections)
d_s	Depth of the search from the start node
d_g	Depth of the search from the goal node
d	Total solution depth ($d = d_s + d_g$)
G	Grid map representation of the environment, $G = (V, E)$.
V	A node (or vertex) in the grid, $V = \{ v = (x, y) \mid x, y \in Z, 0 \leq x < W, 0 \leq y < W \}$ where W is the size of the square maze (50, 80, 100)
E	An edge between two adjacent nodes v_i, v_j, indicating traversability
s, g	Start node, Goal node respectively
P	The complete path from start to goal $\exists\, P = \{n_0 = s, n_1, \ldots, n_k = g\} \subseteq V \mid (n_j, n_{j+1}) \in E, \forall j$
α	Sub-optimality ratio: $\alpha = \frac{C_{LiteRBS}}{C^*}$
C^*	Cost (length) of the optimal path (e.g., A* result)
$C_{LiteRBS}$	Cost of the path returned by LiteRBS

References

[1] Jeddisaravi K, Alitappeh RJ, Pimenta LCA, Guimarães FG. Multi-objective approach for robot motion planning in search tasks. *Appl. Intell*. 2016, 45:305–321.

[2] Poole DL, Mackworth AK. Artificial Intelligence: foundations of computational agents. Cambridge: Cambridge University Press. 2010.

[3] Sanchez-Ibanez JR, Pérez-del-Pulgar CJ, García-Cerezo A. Path planning for autonomous mobile robots: a review. *Sensors* 2021, 21(23):7898.

[4] Alenezi MR, Almeshal AM. Optimal path planning for a remote sensing unmanned ground vehicle in a hazardous indoor environment. *Intell. Control Autom.* 2018, 9(4):147.

[5] Ishigami G, Nagatani K, Yoshida K. Path planning and evaluation for planetary rovers based on dynamic mobility index. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, California, USA, September 25–30, 2011, pp. 601–606.

[6] Ul Islam N, Gul K, Faizullah F, Ullah SS, Syed I. Trajectory optimization and obstacle avoidance of autonomous robot using robust and efficient rapidly exploring random tree. *PLoS One* 2024, 19(10):e0311179.

[7] Liu YT, Sun RZ, Zhang TY, Zhang XN, Li L, *et al.* Warehouse-oriented optimal path planning for autonomous mobile fire-fighting robots. *Secur. Commun. Netw.* 2020, 2020(1):6371814.

[8] Jameel Al-Kamil S, Szabolcsi R. Optimizing path planning in mobile robot systems using motion capture technology. *Results Eng.* 2024, 22:102043.

[9] Saranya C, Rao KK, Unnikrishnan M, Brinda V, Lalithambika VR, *et al.* Real time evaluation of grid based path planning algorithms: a comparative study. *IFAC Proceed. Volumes* 2014, 47(1):766–772.

[10] Howden WE. The sofa problem. *Comput. J.* 1968, 11(3):299–301.

[11] Zafar MN, Mohanta JC. Methodology for path planning and optimization of mobile robots: a review. *Procedia computer sci.* 2018, 133:141–152.

[12] Liu L, Wang X, Yang X, Liu H, Li J, *et al.* Path planning techniques for mobile robots: review and prospect. *Expert Syst. Appl.* 2023, 227:120254.

[13] Qin H, Shao S, Wang T, Yu X, Jiang Y, *et al.* Review of autonomous path planning algorithms for mobile robots. *Drones* 2023, 7(3):211.

[14] Liu H, Zhang Y. ASL-DWA: an improved A-star algorithm for indoor cleaning robots. *IEEE Access* 2022, 10:99498–99515.

[15] Zhong X, Tian J, Hu H, Peng X. Hybrid path planning based on safe A* algorithm and adaptive window approach for mobile robot in large-scale dynamic environment. *J. Intell. Rob. Syst.* 2020, 99(1):65–77.

[16] Huang J, Chen C, Shen J, Liu G, Xu F. A self-adaptive neighborhood search A-star algorithm for mobile robots global path planning. *Comput. Electr. Eng.* 2025, 123:110018.

[17] Xu D, Yang J, Zhou X, Xu H. Hybrid path planning method for USV using bidirectional A* and improved DWA considering the manoeuvrability and COLREGs. *Ocean Eng.* 2024, 298:117210.

[18] Harabor D, Grastien A. Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI conference on artificial intelligence*, San Francisco, USA, August 7–11, 2011, pp. 1114–1119.

[19] Guo R, Quan X, Bao C. Research on mobile robot path planning based on an improved bidirectional jump point search algorithm. *Electronics* 2025, 14(8):1669.

[20] Wang Z, Tu H, Chan S, Huang C, Zhao Y. Vision-based initial localization of AGV and path planning with PO-JPS algorithm. *Egyptian Inf. J.* 2024, 27:100527.

[21] An Z, Li C, Han Y, Niu M. Improved bidirectional JPS algorithm for mobile robot path planning in complex environments. *Comput. Mater. Continua* 2025, 83(1).

[22] Zhou X. An improved SPFA algorithm for single-source shortest path problem using forward star data structure. *Int. J. Manag. Inf. Technol.* 2014, 6(1).

[23] Ul Islam N, Gul K, Faizullah F, Ullah SS, Syed I. Trajectory optimization and obstacle avoidance of autonomous robot using robust and efficient rapidly exploring random tree. *PLoS One* 2024, 19(10):e0311179.

[24] Sands T. Autonomous real-time mass center location and inertia identification for grappling space robotics. *Technologies* 2025, 13(4):148.
[25] Kuck E, Sands T. Space robot sensor noise amelioration using trajectory shaping. *Sensors* 2024, 24(2):666.
[26] Sands T. Flattening the curve of flexible space robotics. *Applied Sci.* 2022, 12(6):2992.
[27] Yang Y, Pan J, Wan W. Survey of optimal motion planning. *IET Cyber-syst. Robot.* 2019, 1(1):13–19.
[28] Lee CY. An algorithm for path connections and its applications. *IRE Trans. Electron. Comput.* 2009, 3:346–365.
[29] Holte R, Felner A, Sharon G, Sturtevant N. Bidirectional search that is guaranteed to meet in the middle. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Phoenix, USA, February 12–17, 2016, pp. 3411–3417.
[30] Jun S, Lee S, Yih Y. Pickup and delivery problem with recharging for material handling systems utilising autonomous mobile robots. *Eur. J. Oper. Res.* 2021, 289(3):1153–1168.
[31] Shen B, Cheema MA, Harabor DD, Stuckey PJ. Fast optimal and bounded suboptimal euclidean pathfinding. *Artif. Intell.* 2022, 302:103624.