

Solving the Multi-Agent Path Finding Problem Using a Hierarchical Multi-Resolution Planner

Gicu Rata ¹, Liviu Octavian Mafteiu-Scail ²

¹ Faculty of Informatics, West University of Timisoara, Romania

Abstract - Multi-Agent Path Finding (MAPF) studies how to compute collision-free routes for multiple robots moving simultaneously on a shared map, a setting in which naive shortest-path behavior often leads to bottlenecks, deadlocks, and costly replanning. This paper proposes a hierarchical, reservation-based, multi-resolution planner for discrete-time grid environments. This paper extends CREDAR, our previously published algorithm [3], along the hierarchical direction that was proposed there as a future work. Each robot plans in a time-expanded state space ((x, y, t)) using A* while coordinating through a shared reservation table that prevents both vertex conflicts and edge-swap collisions. To improve guidance beyond Manhattan distance in maps with large obstacles and dead ends, the method builds a block-level abstraction of the grid. It computes a goal-rooted “true distance” estimate by running Reverse Dijkstra on the abstract graph. The resulting abstract distance map is cached per goal location and reused across agents and timesteps, enabling fast heuristic lookups during repeated replans. Several robustness mechanisms are included to stabilize execution in dense traffic, including deadline-aware priority guidelines to resolve reservation contention, boundary-based validation of abstract connectivity to avoid phantom links, and a wait-and-retry fallback when planning temporarily fails.

DOI: 10.18421/SAR93-05

<https://doi.org/10.18421/SAR93-05>

Corresponding author: Liviu Octavian Mafteiu-Scail,
Faculty of Informatics, West University of Timisoara,
Romania

Email: liviu.mafteiu@e-uvt.ro

Received: 06 July 2026.

Revised: 03 September 2026.

Accepted: 10 September 2026.

Published: 27 September 2026.



©2026 Gicu Rata & Liviu Octavian Mafteiu-Scail; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License.

The article is published with Open Access at <https://www.sarjournal.com/>

To reduce agent convergence in narrow corridors without explicit communication, the planner adds a small congestion penalty derived from historical reservations, thereby biasing the search toward underutilized routes whilst preserving feasibility.

Keywords - multi-agent pathfinding, robot, path conflict, collision, agent.

1. Introduction

Multi-Agent Path Finding (MAPF) is the problem of optimizing the simultaneous movement of multiple agents/robots, each with a starting and destination point within a shared environment, while avoiding collisions. From a general perspective, MAPF is an optimization problem for each agent in turn, a generalization of pathfinding and shortest-path problems in graph theory. As it is well known, the optimal solution for MAPF is NP-hard.

Formally, an instance consists of a set of agents, a graph whose vertices are the admissible positions and whose edges are the transitions between them, a start and a goal vertex for each agent [1]. Depending on the real problems to be solved, additional elements necessary to customize the objective function can be incorporated, such as agent velocity, orientation, and shape. The prioritized planning family of algorithms orders the agents and plans them one at a time, each treating the trajectories that are already committed by its predecessors as moving obstacles. Cooperative A*[2] is its canonical member. Those algorithms are fast and simple to implement, but it is incomplete. Whether a solution is found depends on the ordering chosen, and it offers no guarantee on solution quality. CREDAR [3], the direct predecessor of the present work, extends CA* with deadline-aware priorities that are recomputed at every time step and with reactive, local replanning: a robot whose reserved trajectory has been invalidated re-plans against the shared table instead of the whole schedule being restarted. Its low-level search, however, is still guided by Manhattan distance, and it is precisely this that the current paper replaces.

Other MAPF algorithms, with different implementations and purposes, are described and reviewed in papers [4], [5], [6], [7].

There are many applications of MAPF in solving real-world problems, such as autonomous mobile service robots [8], airport operations [9], pipe routing [10], video games [11], job distribution [12], 3D warehouse management [13], etc.

2. The Proposed Method

HMRP is built directly on top of CREDAR [3], which is itself a prioritised planner in the CA* family [2].

The environment model, the reservation table, the priority function, the simulation loop, and the per-robot step procedure are taken over from [3] without semantic change and are only summarised here. What HMRP changes is the guidance and the cost model of the low-level search, and this is what the remainder of Section 2 describes. Table 1 summarises the differences. We identify four problems in those approaches and describe how to address them using the HMRP algorithm in the following subsections.

Table 1. What HMRP adopts from CREDAR [3] and what is new in this paper

Adopted from [3] without semantic change	
Environment and robot model	2D four-connected grid $G = (V, E)$, discrete time; robot $R_i = (s_i, g_i, r_i, d_i, id_i)$ with release time and optional deadline. [3, Sec. 2.2.1]
Reservation table and conflict rules	Shared table $R: V \times N_0 \rightarrow \text{Robot} \cup \{\text{None}\}$; vertex and edge (swap) conflicts; goal held until the horizon. [3, Sec. 2.2.2]
Dynamic priority	Tuple (feasibility code, deadline value, distance to goal). [3, Sec. 2.2.3]
Simulation loop and step procedure	Per-timestep sequential execution with reactive replanning. [3, Sub-Algorithms 1 and 3]
Evaluation protocol	Metric set (success, runtime, makespan, optimality) and hardware/software configuration. [3, Sec. 3]
Adopted and modified	
Low-level search	The time-expanded A* over (x, y, t) is kept; its f-score is redefined (heuristic and g-cost) and a search horizon is added.
Benchmark set	The two maze families of [3] are re-used; Berlin_1_256 is added; team sizes $k \in \{10, \dots, 80\}$ instead of $\{10, \dots, 110\}$; the baselines are extended from $\{\text{CA}^*\}$ to $\{\text{CA}^*, \text{CREDAR}\}$.
New in this paper	
Block abstraction Gabs	Grid down-sampled into $B \times B$ blocks ($B = 4$), fully blocked blocks removed. Announced as future work in [3, Sec. 5b], not implemented there.
Abstract-edge validation	An abstract edge exists only if a boundary scan finds at least one pair of free adjacent cells on the shared border of the two blocks, which suppresses phantom connections.
True-distance heuristic	Reverse Dijkstra from the goal block on Gabs, combined as $h(x, y) = \max(D_{\text{abs}} \cdot B, \text{Manhattan})$, replacing the pure Manhattan heuristic of [3].
Two-level heuristic cache	Block costs cached per (map, B); abstract distance maps cached per (map, B, goal block) and shared across robots and across replans.
Congestion penalty	The g-cost is augmented by $\alpha \cdot C(x, y)$ with $\alpha = 0.01$, where C is the per-cell reservation histogram.
Failure handling	Dynamic fail-fast horizon plus a bounded wait-and-retry policy, replacing the immediate failure marking of [3].

2.1. True Distance Heuristic and Reverse Dijkstra

To calculate the cost of a move, in the Cooperative path algorithm [2], the Manhattan Distance heuristic [14] was used to compute the cost:

$$h(n) = |x_{goal} - x_n| + |y_{goal} - y_n| \quad (1)$$

and signifies the distance between two coordinate points, (x_{goal}, y_{goal}) and (x_n, y_n) .

In our opinion, in maze environments, this heuristic is often misleading because it can lead the search into dead ends. Although in such cases, from a geometric point of view, we are in positions close to the goal, from a topological point of view, we are in a deadlock. The solver has to expand the entire dead end before backtracking, which results in massive runtime spikes.

To solve such situations, we implemented a True Distance Heuristic (TDH) [15] based on a coarse abstraction of the grid:

- the grid is down-sampled into 4×4 blocks;
- The reverse Dijkstra algorithm was run on this abstract graph, starting from the goal block. Basically, the Reverse Dijkstra algorithm adapts the classic algorithm by changing the search direction or graph structure.

Let $G_{abs} = (V_{abs}, E_{abs})$ be the abstract graph where each node $v \in V_{abs}$ represents a 4×4 region.

The heuristic $h_{abs}(n)$ for a node n at position (x, y) is:

$$h_{abs}(n) = D_{Dijkstra}(block(n), block(goal)) \times BLOCK\ SIZE \quad (2)$$

where $D_{Dijkstra}$ is the pre-computed path cost on the abstract graph.

- thus, the true shortest path distance from every block to the goal is calculated, accounting for all static obstacles.

As a result, the low-level A* search becomes “globally aware.” It will never enter a dead end unless it is the only viable path, because the heuristic value inside the dead end correctly reflects the high cost of exiting it.

This causes a major computational bottleneck within high-density instances or in dynamic settings, where agents repeatedly encounter collisions or timing disruptions, demanding frequent replanning. Recomputing global paths and heuristics for each robot at every replanning step is both computationally expensive and computationally heavy.

To address this issue, the algorithm adopts a static abstraction combined with in-memory caching. Rather than reconstructing the abstract grid costs and the Abstract Distance Map for each new path request, these structures are computed once per distinct goal location and stored.

Subsequent planning requests, whether from the same robot or another robot targeting the same goal block, use the cached precomputed values, thereby avoiding unnecessary computations.

This approach transforms replanning from a repeated graph-search and heuristic-construction workload into a lightweight lookup operation, where the robot accesses cached heuristic values during the A* search.

2.2. Robustness Improvements

A persistent problem in hierarchical pathfinding is the mismatch between how the planner sees the map at the abstract level and what actually happens in the more detailed map, creating a “phantom connection”: edges that look valid in the abstract graph but are physically impossible to traverse in the underlying grid because of walls and obstacles.

These false links are more than a minor modeling error. Once they exist, the Abstract Distance Heuristic can become misleading in a very confident way: it creates a “downhill” direction toward the goal that does not correspond to any real passage. The low-level A* search then wastes time following that gradient until it hits a wall, at which point it has to unwind, explore alternatives, and recover through backtracking. In cluttered environments or within multi-agent settings where replanning is already frequent, such misguidance quickly becomes a significant performance penalty.

To prevent this kind of mismatch, the planner does not assume that neighboring blocks are connected. Instead, it checks the cells between two adjacent blocks. It scans along the shared border and identifies at least one valid crossing point. Only if such a “gate” exists does the algorithm allow an abstract edge between those blocks. This sustains the abstract layer grounded in the real grid topology: every high-level connection corresponds to a concrete move a robot can execute. As a result, the abstract distance map stops suggesting impossible shortcuts, and the low-level A* is much less likely to waste time following misleading guidance into blocked seams.

The second robustness mechanism bounds the search itself. Each call to the low-level planner is given a horizon of $Manhattan(R) + 5(rows + cols)$ timesteps; if the search exhausts that horizon without reaching the goal, the robot does not fail immediately but holds its current cell for a few timesteps and retries, and it is declared failed only after ten consecutive unsuccessful attempts. This dynamic fail-fast policy prevents unbounded searches on large maps while keeping temporarily blocked robots in the simulation.

2.3. Congestion Penalty

A common challenge in MAPF is agent convergence, where independently planning robots tend to utilize the same short corridors. This “everyone takes the highway” behavior rapidly creates bottlenecks, which, as the map becomes crowded, can escalate into repeated conflicts and occasional deadlocks. The objective is to reduce crowding without requiring constant negotiation among robots.

The algorithm handles this issue by introducing a lightweight congestion penalty into the low-level A* cost. Before planning, each robot constructs a congestion map from the global reservation table by counting the frequency of reservations for each cell by previous robots, resulting in a heatmap that highlights areas of concentrated traffic. This process identifies the primary crowded routes that are more likely to cause future conflicts.

During the time-expanded A* search, an additional cost proportional to each cell’s congestion count, scaled by 0.01, is added when evaluating moves into neighboring cells. This value is sufficiently small to avoid overriding the distance-to-goal metric or causing unnecessary detours. Instead, it functions primarily as a tie-breaker, encouraging traffic distribution. When two options are of similar length, the search favours the less congested path, while still selecting the crowded corridor if it is the single feasible route.

2.4. Algorithm Descriptions

As noted in the previous subsections, the method proposed in this paper differs from the algorithms in [2] and [3], as shown in the pseudocode below.

The key elements of these algorithms are:

- the environment is represented as a 2D grid $G = (V, E)$, where in graph G , V is the set of grid cells (locations), and E contains edges connecting adjacent, non-obstacle cells that typically have four-connectivity: North, South, East, West.
- an agent/robot R_i is defined by a tuple $(s_i, g_i, r_i, d_i, id_i)$, where s_i and $g_i \in V$ are start and goal locations, $r_i \in \mathbb{N}_0$ is the release time i.e. the earliest time step robot I can leave s_i , $d_i \in \mathbb{N}_0 \cup \{\infty\}$ is the deadline i.e. the latest time step robot I must arrive at g_i (if $d_i = \infty$, there is no deadline) and a unique identifier id_i . Also, a robot R_i is characterized at a given time t by its current location $(x_i(t), y_i(t))$ and two internal flags indicating whether it has reached its goal (at_goal_flag) or failed its goal (deadline_missed).

2.4.1. Abstract Heuristic Computation Algorithm

One of the main enhancements over the versions proposed in [2] and [3] is the replacement of Manhattan distance with a heuristic, namely the True Distance Heuristic, computed via the Reverse Dijkstra method on a 4×4 coarse block abstraction. This provides globally aware guidance, preventing the search from entering dead ends in maze environments.

We down-sample the grid into $B \times B$ blocks and run Reverse Dijkstra from the goal block to compute true distances. These are stored in memory for each goal location.

Algorithm 1: compute abstract heuristic(R, G)

Input: Robot R with goal (g_x, g_y) , grid G , block size $B = 4$

Output: Abstract distance map D_{abs}

$goal_{abs} \leftarrow ([g_x/B], [g_y/B])$

if D_{abs} cached for $goal_{abs}$ **then return** cached map costs $\leftarrow$ compute block costs (density-weighted)

$dist \leftarrow \{goal_{abs} : 0\}$; $PQ \leftarrow [(0, goal_{abs})]$

while $PQ \neq \emptyset$ **do**

$(d, curr) \leftarrow$ pop min from PQ

foreach neighbor block n adjacent to $curr$ **do**

if n fully blocked $\vee$ no free boundary cells **then continue**

$w \leftarrow costs[n]$

if $d + w < dist[n]$ **then**

$dist[n] \leftarrow d + w$; push $(d + w, n)$ to PQ

cache and **return** $dist$

The heuristic function becomes:

$$h(x, y) = \max(D_{abs}[\lfloor x/B \rfloor, \lfloor y/B \rfloor] \times B, |x - g_x| + |y - g_y|) \quad (3)$$

2.4.2. Simulation Loop aAlgorithm

The simulation loop is that of CREDAR [3, Sub-Algorithm 1] and is unchanged: at each timestep the robots that have neither reached their goal nor missed their deadline are sorted by the dynamic priority tuple, and each executes one step in that order, and the loop terminates early when no robot is active. HMRP does not modify this schedule, which is what makes the comparison in Section 4 a controlled one: CREDAR and HMRP differ only in how a single robot plans, not in how the robots are sequenced.

2.4.3. Hierarchical Path Planning Algorithm

In this algorithm, the search explores (x, y, t) states, prunes those that collide with static obstacles, and reserves the resulting path.

Unlike the methods proposed in [2] and [3] that use the Manhattan distance, this algorithm uses the habs heuristic (explained in subsection 2.1), guiding it around obstacles globally.

It is worth mentioning that the abstract heuristic is calculated before the A* search.

In summary, the abstract heuristic map is computed lazily (once per robot, cached per goal block), and the f-score uses h_{abs} instead of Manhattan distance.

Additionally, a small congestion penalty is applied to the g-score based on the number of reservations at each cell. This acts as a tie-breaker, guiding robots away from heavily congested areas when multiple paths have similar costs.

Algorithm 3 retains the structure of the time-expanded A* search of [3, Sub-Algorithm 2]: the same state space (x, y, t) , the same successor set, the same vertex- and edge-conflict tests, and the same goal-holding construction. Three lines are new, and they carry the contribution of this paper: the abstract heuristic is computed once and cached before the search starts, the g-cost is augmented by the congestion term, and the search is bounded by a map-dependent horizon whose exhaustion triggers the wait-and-retry policy rather than an immediate failure.

Algorithm 3: `plan_path(R, G, t)` the time-expanded A* of [3, Sub-Algorithm 2]; the lines marked ► are new in HMRP.

Input: Robot R, grid G, current time t, penalty factor $\alpha = 0.01$

Output: Reserves and stores a new path or marks failure

```

R.remove_reservations()
► compute_abstract_heuristic(R, G)
goals_blocked ← positions of robots at their goal
► C ← congestion map: count reservations per cell
► horizon ← t + Manhattan(R) + 5(rows + cols)
// dynamic fail-fast
OPEN ← { (R.x, R.y, t) } with g = 0, f = h(R.x, R.y)
while OPEN ≠ ∅ and expansions < limit do
    pop (x, y, t) with smallest f
    if (x, y) = R.goal and no reservation conflict ahead then
        goto construct_path
    if τ ≥ horizon then continue
    foreach (dx, dy) ∈ {↑, ↓, ←, →, wait} do
        (x', y', τ') ← (x + dx, y + dy, τ + 1)
        if off-grid ∨ blocked ∨ τ' > di ∨ in goals_blocked then continue
        if vertex/edge conflict at (x', y', τ') then continue

```

```

    ► g' ← g + 1 + α · C(x', y')
    //congestion penalty
    if g' < g[(x', y', τ')] then update and push to OPEN
return R.handle_planning_failure(t)
construct_path: backtrace, append goal-holds, reserve;
if R.reserve_path(path) then
    store and return success
else return failure

```

2.4.4. Step Method Algorithm

The per-robot step procedure is likewise taken unchanged from [3, Sub-Algorithm 3]: a robot validates its plan, replans through `plan_path` when the plan is empty, desynchronised from the current timestep, or invalidated by a reservation conflict, and otherwise advances one cell and, on arrival, records its arrival time. The only HMRP-specific behaviour reachable from this procedure is the failure path described in Section 2.2.

3. Implementation

All three algorithms were implemented in Python 3.12.10 and run on the same machine and software configuration as [3] (Intel Core Ultra 7 155H, 16 GB RAM, Windows 11;

4. Experimental results

The evaluation comprises 1020 benchmark instances per algorithm, that is 3060 solver runs: 350 on maze-32-32-2, 350 on maze-32-32-4 and 320 on Berlin_1_256. Each instance is obtained by crossing a MovingAI even scenario of the map with one of the eight team sizes and taking the first k agents of that scenario, which makes every instance reproducible. The maps and scenarios are taken from the public 2D Pathfinding Benchmarks set available at <https://www.movingai.com/benchmarks/mapf/index.html>.

Figure 1, Figure 2, Figure 3, and Figure 4 present the comparative experimental results for the maze-32-32-2, maze-32-32-4, and Berlin_1_256.map map families, obtained for the three algorithms: CA*, CREDAR, and HMDP.

For each map, we sample robot team sizes $k \in \{10, 20, \dots, 80\}$ in steps of ten. The robots are randomly selected from a list of robots provided by the benchmark.

For comparison, we used four metrics:

- success: ratio of resolved instances, higher is better;
- runtime: wall-clock time (s), lower is better;
- make-span: steps until the last robot arrives, lower is better;
- optimality: Σ path-length / Σ shortest-path (only for successful robots), lower is better

No experimental result reported here is taken from [3]: all runs, for all three algorithms, were executed afresh for this paper on the machine described in Section 3, and every runtime reported below is a new measurement.

All Algorithms Comparison

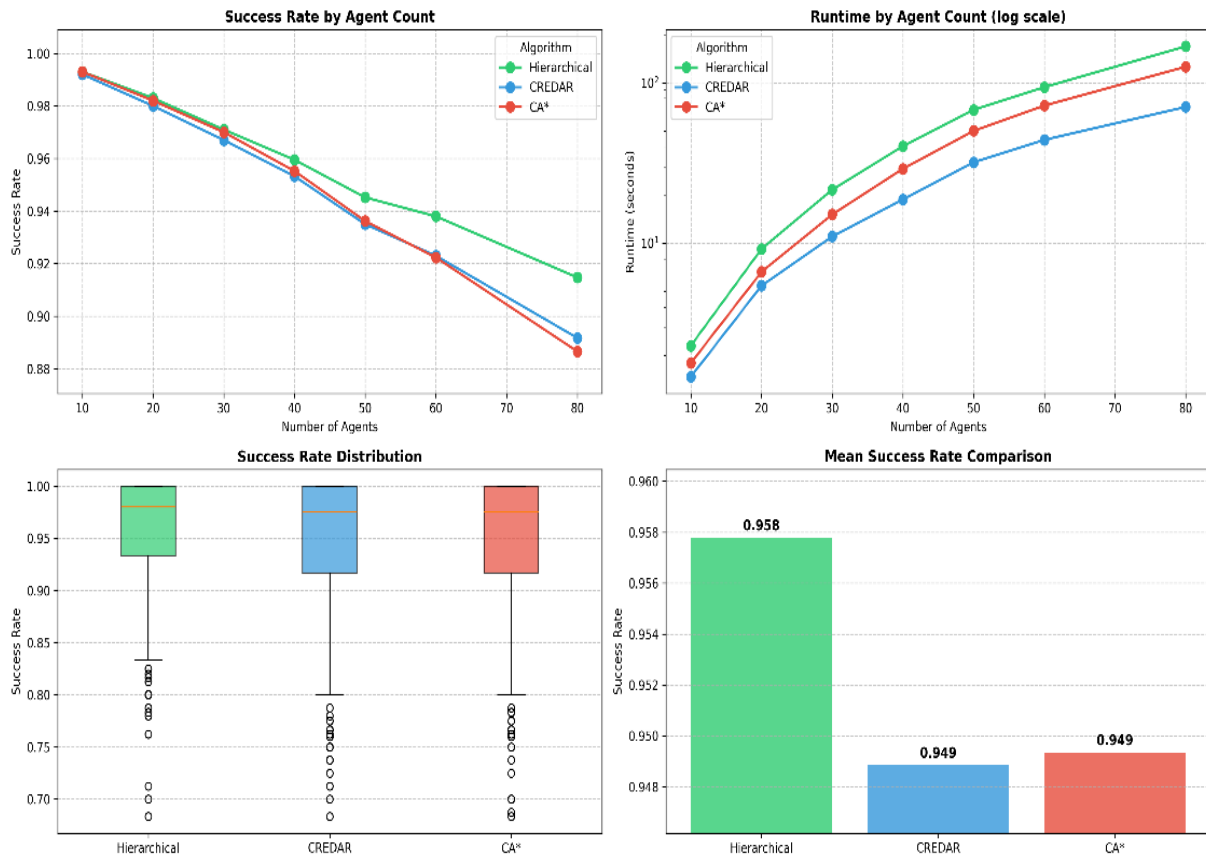


Figure 1. Comparison CA*, CREDAR and HMRP algorithms

All Algorithms Comparison

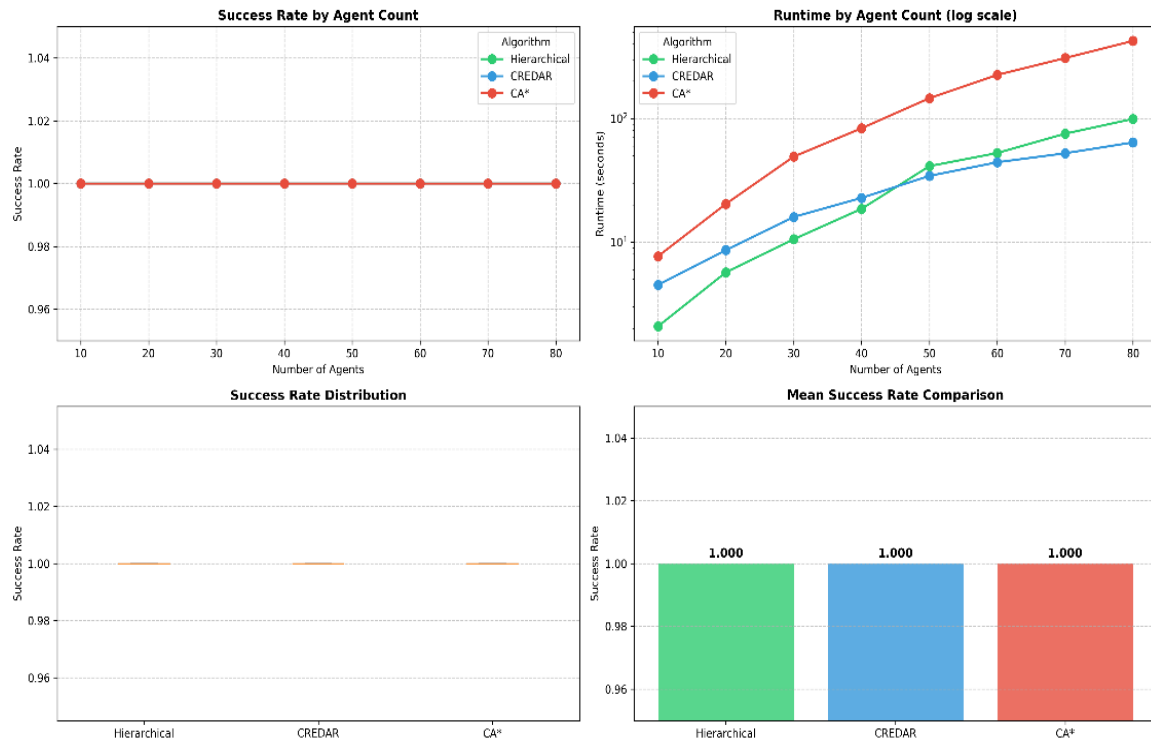


Figure 2. Comparison CA*, CREDAR and HMRP algorithms

Hierarchical vs CREDAR Comparison

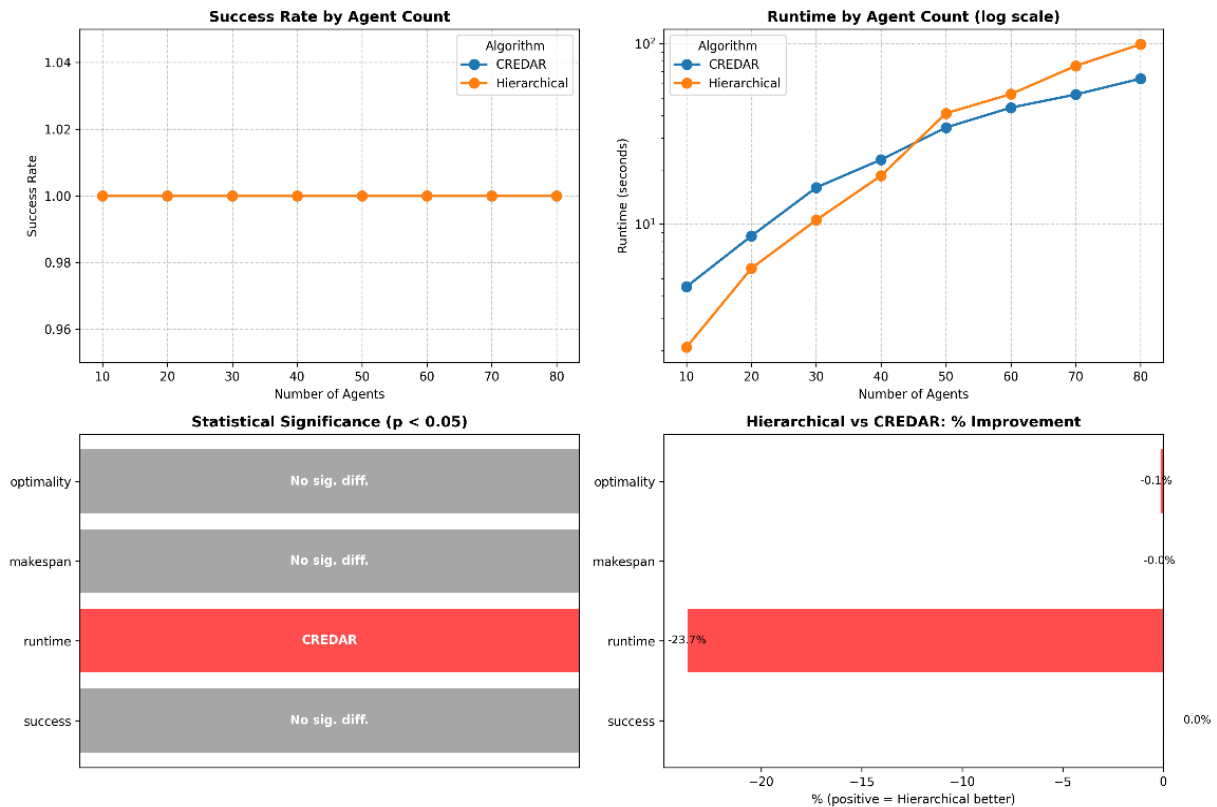


Figure 3. Comparison CREDAR vs HMRP algorithms

Hierarchical vs CREDAR Comparison

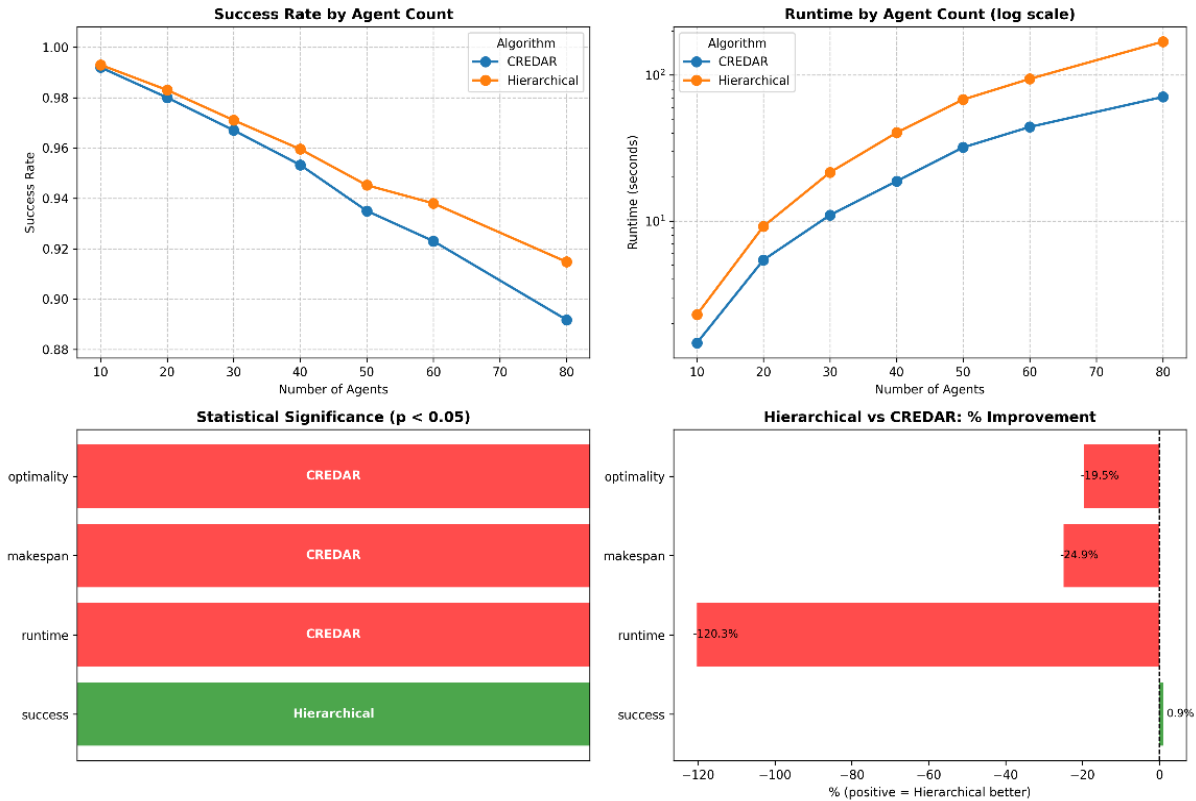


Figure 4. Comparison CREDAR vs HMRP algorithms

5. Conclusions and Future Work

In this paper we proposed HMRP, an extension of the CREDAR algorithm [3] in which the Manhattan heuristic of the low-level search is replaced by a cached true-distance estimate computed on a validated block abstraction. We evaluated experimentally the four mechanisms that this requires:

- Abstract Distance Heuristic: provides globally-aware guidance that avoids dead ends and local
- Minima;
- Hierarchical Caching: reduces repeated computation through intelligent memorization of abstract distance maps;
- Dynamic Fail-Fast: prevents infinite search loops while scaling appropriately to larger maps;
- Congestion penalty: to further improve multi-agent coordination.

From Figures 1-4, HMRP attains a small but statistically significant gain in success rate over both CA* and CREDAR, at the cost of longer paths and, on the small maze maps, of higher planning time.

Its runtime advantage appears where the abstraction was designed to pay off: on the large Berlin_1_256 map, HMRP needs 38.1 s on average against 157.9 s for CA*, a speed-up of 4.1 $\times$, whereas on the maze families it remains slower than CREDAR.

As a future research direction, to improve runtimes, we propose a parallelization of all three algorithms (CA*, CREDAR, and HMRP), i.e., a division of parts of these algorithms into smaller subtasks to be executed concurrently on multiple processors.

Another future research direction, as one way to speed up the algorithm, would be to plan paths in parallel for robots that are far apart. If two robots cannot cross paths given their positions and time constraints, there is no reason to plan them sequentially. They could run on separate threads with their own reservation tables and merge the results at the end. The abstract grid we already computed could help identify which robots are truly independent. This would help most in large maps where robots tend to stay in their own areas.

References:

- [1]. R. Stern, *et al.*, “Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks,” *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 151–158, 2021. doi: 10.1609/socs.v10i1.18510.
- [2]. D. Silver, “Cooperative Pathfinding,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, no. 1, pp. 117–122, 2005. doi: 10.1609/aiide.v1i1.18726.
- [3]. L. O. Maftciu-Scai and G. Rata, “A Cooperative Reservation and Deadline-Aware Routing Algorithm for Multi-Agent Pathfinding Problem,” *SAR Journal*, vol. 8, no. 3, pp. 240–247, 2025. doi: 10.18421/SAR83-04.
- [4]. J. Gao, Y. Li, X. Li, K. Yan, K. Lin, and X. Wu, “A review of graph-based multi-agent pathfinding solvers: From classical to beyond classical,” *Knowledge-Based Systems*, vol. 283, Art. no. 111121, 2024. doi: 10.1016/j.knosys.2023.111121.
- [5]. J.-M. Alkazzi and K. Okumura, “A Comprehensive Review on Leveraging Machine Learning for Multi-Agent Path Finding,” *IEEE Access*, vol. 12, pp. 57390–57409, 2024. doi: 10.1109/ACCESS.2024.3392305.
- [6]. J. Chung, J. Fayyad, Y. Al Younes, and H. Najjaran, “Learning team-based navigation: A review of deep reinforcement learning techniques for multi-agent pathfinding,” *Artificial Intelligence Review*, vol. 57, no. 2, Art. no. 41, 2024. doi: 10.1007/s10462-023-10670-6.
- [7]. Y. Li, Z. Zhang, W. Zhou, X. Tu, and H. He, “A Path Planning Approach to Address Urban Parking Challenges Through Optimized Plane Intensive Parking Lots,” in *Proc. 2024 9th Int. Conf. Computer and Communication Systems (ICCCS)*, 2024. doi: 10.1109/ICCCS61882.2024.10603241.
- [8]. M. M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal, “CoBots: Robust Symbiotic Autonomous Mobile Service Robots,” in *Proc. 24th Int. Joint Conf. Artificial Intelligence (IJCAI)*, Buenos Aires, Argentina, 2015, pp. 4423–4429.
- [9]. R. Morris, C. S. Pasareanu, K. Luckow, W. Malik, H. Ma, T. K. Satish Kumar, and S. Koenig, “Planning, Scheduling and Monitoring for Airport Surface Operations,” in *Proc. AAAI-16 Workshop on Planning for Hybrid Systems (PlanHS)*, 2016, pp. 608–614.
- [10]. G. Belov, W. Du, M. Garcia de La Banda, D. Harabor, S. Koenig, and X. Wei, “From Multi-Agent Pathfinding to 3D Pipe Routing,” *Proceedings of the International Symposium on Combinatorial Search*, vol. 11, no. 1, pp. 11–19, 2021. doi: 10.1609/socs.v11i1.18530.
- [11]. H. Ma, J. Yang, L. Cohen, T. K. Kumar, and S. Koenig, “Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, no. 1, pp. 270–272, 2017. doi: 10.1609/aiide.v13i1.12919.
- [12]. F. Semiz, M. A. Yorgancı, and F. Polat, “Solving an industry-inspired generalization of lifelong MAPF problem including multiple delivery locations,” *Advanced Engineering Informatics*, vol. 57, Art. no. 102026, 2023. doi: 10.1016/j.aei.2023.102026.
- [13]. Q. Wang, R. Veerapaneni, Y. Wu, J. Li, and M. Likhachev, “MAPF in 3D Warehouses: Dataset and Analysis,” *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, no. 1, pp. 623–632, 2024. doi: 10.1609/icaps.v34i1.31525.
- [14]. B. Li, C. Dong, Q. Chen, Y. Mu, Z. Fan, Q. Wang, and X. Chen, “Path planning of mobile robots based on an improved A* algorithm,” in *Proc. 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence (HPCCT & BDAI)*, 2020, pp. 49–53. doi: 10.1145/3409501.3409524.
- [15]. A. Felner, N. R. Sturtevant, and J. Schaeffer, “Abstraction-based heuristics with true distance computations,” in *Proc. 8th Symp. Abstraction, Reformulation Approximation (SARA)*, 2009, pp. 74–81.