Puzzle Heuristics: Efficient Lifelong Multi-Agent Pathfinding Algorithm for Large-scale Challenging Environments

Wonjong Lee[1]*, Joonyeol Sim[2]*, Changjoo Nam[2]

[1]Department of Artificial Intelligence, Sogang University, Seoul, Korea
[2]Department of Electronic Engineering, Sogang University, Seoul, Korea

Abstract

This paper describes the solution method of Team AIRLAB used to participate in the League of Robot Runners Competition which tackles the problem of Lifelong Multi-agent Pathfinding (MAPF). In lifelong MAPF, multiple agents are tasked to navigate to their respective goal locations where new goals are consecutively revealed once they reach initial goals. Our method consists of (i) Puzzle Heuristics, (ii) MAPF-LNS2, and (iii) RHCR. The Puzzle Heuristics is our own algorithm that generates a compact heuristic table contributing to reduce memory consumption and computation time. MAPF-LNS2 and RHCR are state-of-the-art algorithms for MAPF. By combining these three algorithms, our method can improve the efficiency of paths for all agents significantly.

Introduction

Multi-agent Pathfinding (MAPF) is the problem that finds collision-free paths for multiple agents from their start locations to their goal locations while optimizing the sum of total path lengths or the time taken for all tasks to complete (Stern et al. 2019). It has numerous applications in various domains, such as logistics, aircraft, and urban traffic systems. In many such applications, the agents are tasked to visit multiple destinations consecutively.

Lifelong MAPF (Ma et al. 2017) is an extended version of the MAPF problem. In contrast to the canonical form of MAPF, where each agent has only one task and remains at its goal location, lifelong MAPF requires agents to generate paths to their respective new goal locations after reaching their current goals. The challenges lie in the fact that the next goals are not known to the agents beforehand and the time that the agents finish their current tasks are not synchronized. Thus, it is paramount to have an efficient method that can find collision-free paths promptly for a dynamically varying task set given a limited time budget.

The League Of Robot Runners (Harabor 2023) is a competition to tackle the lifelong MAPF in challenging environments in terms of the complexity of the environments as well as the scale of the agent team. Our team AIRLAB (ranked 8th) developed a method employing one of the state-of-the-art algorithms MAPF-LNS2 (Li et al. 2022) and Rolling-Horizon Collision Resolution (RHCR) framework (Li et al. 2021) as the MAPF solver. Our key contribution is to develop our own approach, *Puzzle Heuristics*, which generates a compact heuristic tables to reduce memory consumption and computation.

*These authors contributed equally.

Problem Definition

Our goal is to solve the lifelong MAPF instances presented in the League of Robot Runners competition. Therefore, the problem definition is dictated by the goal and rules of the competition.

A team of k agents $\mathcal{A} = \{a_1, a_2, ..., a_k\}$ perform errands (i.e., tasks) that appear in an online manner in a deterministic and a fully observable environment. The environment is represented by a grid map $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices (representing grid cells) and $\mathcal{E}$ is the set of edges representing the adjacency of the cells (i.e., G is a 4-connected graph). The agents operate on discrete time steps, denoted as $t = 0, 1, 2, ..., \tau$ where a constant τ is the final time step of an instance. At each time step, each agent can perform one of the four actions: move forward, rotate left or right 90 degrees, or wait.

The position $v_i^t \in V$ is the position of agent a_i on the grid map at time step t. Let $\mathcal{P}_i^t = \{v_i^t, v_i^{t+1}, ..., v_i^{t+T}\}$ be a path of agent a_i from t to $t + T$ that the agent plans to move. Along the path, an agent must avoid conflicts with other agents. Two types of conflicts are defined: (i) the vertex conflict where more than one agents move to the same position and (ii) the edge conflict where two agents swap their positions simultaneously. Specifically, a vertex conflict occurs at t if $\exists i, j$ such that $i \neq j$ and $v_i^t = v_j^t$. An edge conflict occurs at t if $\exists i, j$ such that $i \neq j$, $v_i^t = v_j^{t-1}$ and $v_j^t = v_i^{t-1}$. An errand is located on a position in the map. It can be completed only if the agent assigned to it arrives at the position of the errand. An agent becomes to know the next errand only after it finishes the current one.

Given the assumptions and definitions, the objective of the lifelong MAPF is to maximize the number of errands to be completed by $\mathcal{A}$ within $t = 0, \cdots \tau$.

Methods

By integrating the Puzzle Heuristics with MAPF-LNS2 and RHCR, and Puzzle Heuristics, our method can handle large-

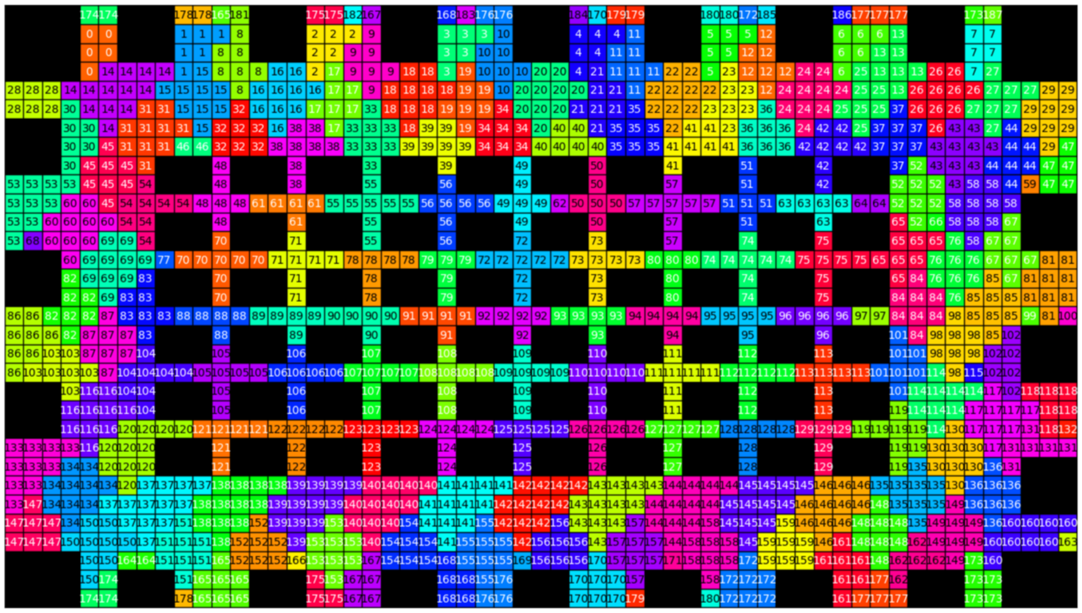

Figure 1: An example result is the Puzzle Heuristics method for generating a compact heuristic table. The same-colored neighboring cells belong to the same area. Each area is formed around a pivot point such that the belonging cells are within the depth limit $d = 3$ using BFS, the root of which is the pivot point. The cells within the same area are assigned the same puzzle index. The heuristic table stores only the distances between the areas (i.e., pivot points).

scale lifelong MAPF instances in challenging environments. We provide a detailed description of each component and how they contribute to the overall effectiveness and efficiency of our solution method.

Puzzle Heuristics

The shortest distance between a pair of cells is frequently queried in an MAPF instance. In a fast-paced situation where the quick online computation of paths is necessary, it is a common approach to precalculate the distances of all pairs. However, as the size of the map grows, the memory required to store the (Manhattan) distances increases significantly. A naïve method has the space complexity $O(N^2)$ for a grid map with N cells. This is particularly problematic in the competition since the memory size allocated to our method is limited. If we reduce the memory usage by storing only part of the distances, some of the computation time to resolve inter-agent conflicts at runtime needs to be allocated to calculate the distances. Therefore, the solution quality should be compromised to balance the trade-off between storage space and computational overhead.

The core idea of Puzzle Heuristics is to divide the grid map into *areas* such that each area contains contiguous cells within a depth d in a tree search from its central cell called *pivot point*. The algorithm begins by selecting the lowest indexed cell (the index increments from top to bottom and left to right like the pixel coordinates) in the grid map as the first pivot point. Using Breadth-First Search (BFS), the area expands over non-obstacle cells from the pivot point, ensuring that the frontier of the BFS does not exceed a depth of d. Cells within the area are labeled with the same unique puzzle index. The process selects the next unlabeled cell with the lowest index as the new pivot point to repeat the area expansion. This procedure iterates until all cells in the grid map are labeled. The same-colored neighboring cells shown in the map of Figure 1 belong to the same area.

Once all cells are formed into areas, the shortest distances between all pairs of areas (i.e., pivot points) are calculated and stored in the heuristic table. By storing only the distances between pivot points instead of all cells, we could be able to meet the memory constraint. When the distance between two cells is queried, the heuristic value in the table effectively approximates the actual distance.

The number of cells in an area can reach up to $2d^2 + 2d + 1$ so at most $2d^2 + 2d + 1$ cells can be represented by a single value. As a result, the size of the heuristic table using Puzzle Heuristics could be reduced to $(N/(2d^2 + 2d + 1))^2$, leading to significant memory savings compared to the naïve approach.

MAPF-LNS2

MAPF-LNS2 is one of the state-of-the-art algorithms for solving large MAPF instances by leveraging the powerful meta-heuristic technique Large Neighborhood Search (Shaw 1998). A key advantage of MAPF-LNS2 is its ability to start from any of feasible or infeasible initial solutions. The algorithm iteratively selects a subset of agents using a neighborhood selection method. The method destroys the current paths of agents and repairs them using an efficient single-agent path planner that minimizes the number of collisions. If the repaired solution reduces the number of collisions, the new paths replace the old ones. This destroy-and-repair pro-

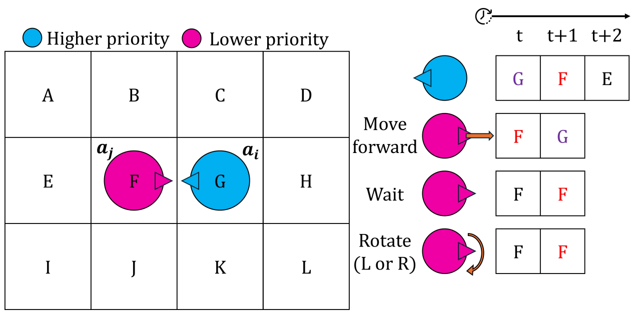

Figure 2: An illustration of deadlock situations where a_i and a_j face each other and attempt to move forward as shown in the grid map in the left. Blue (a_i) has a higher priority so plans P_i^t which passes through the current position of a_j at $t+1$. If a_j moves forward, an edge conflict occurs. If a_j chooses to rotate or wait, a vertex conflict occurs.

cedure continues until a stopping criterion is satisfied.

To achieve high efficiency, MAPF-LNS2 employs Safe Interval Path Planning with Soft Constraints (SIPPS), a variant of the Safe Interval Path Planning algorithm (Phillips and Likhachev 2011) that can handle hard and soft constraints. Integrating SIPPS has shown significant improvements of MAPF-LNS2. While MAPF-LNS2 can be used with many existing MAPF algorithms, we choose to use Prioritized Planning (PP) (Silver 2005) which is simple to implement and fast.

RHCR

RHCR is a framework for solving the lifelong MAPF by decomposing an instance into a sequence of Windowed MAPF instances. RHCR utilizes two user-specified parameters: the time horizon w and the replanning period h. The time horizon w specifies the time window such that the MAPF solver must resolve conflicts within the next w steps (i.e., windowed MAPF solver). The replanning period h determines the frequency of the solver to replan paths. To avoid collisions, w should be larger than or equal to h. The values of these parameters are critical for the performance of RHCR. Too small values of w would lead to deadlocks whereas too large values could result in inefficient solutions and increased computation time.

Greedy Randomized Search

Although MAPF-LNS2 and RHCR can solve large instances quickly, they would lead to deadlocks (i.e., no solution found) as PP is not proven to be complete.

Consider a scenario where a_i and a_j face each other at t and attempt to move forward. Here a_i has a higher priority. During the planning phase, a_i plans its path P_i^t while ignoring the presence of a_j as a_j has a lower priority. As a result, the planned path of a_i goes through the current location of a_j at $t+1$ (i.e., $v_i^{t+1} = v_j^t$). Given this path P_i^t, a_j begins to plan P_j^t.

In this situation, all possible actions for a_j at $t+1$ result in conflicts because P_i^t passes through v_j^{t+1} in any case. Specifically, if a_j chooses to move forward, then an edge conflict occurs (i.e., $v_j^t = v_i^{t+1}$ and $v_j^{t+1} = v_i^t$). If a_j chooses to wait or rotate, a vertex conflict occurs (i.e., $v_i^{t+1} = v_j^t = v_j^{t+1}$ or $v_j^{t+1} = v_j^t = v_i^{t+1}$), respectively). Thus, a_j cannot find a conflict-free action at $t+1$. Figure 2 illustrates the three conflict situations incurring deadlocks.

We develop a greedy randomized search that combines greediness and randomness to deal with deadlocks . We probabilistically multiply the number of edge collisions and vertex collisions by a factor of two. By doing so, our method could have an opportunity to explore alternative paths rather than relying on routes determined by the heuristic values. This simple yet effective approach enables the algorithm to explore a broader range of possibilities and escape from deadlocks.

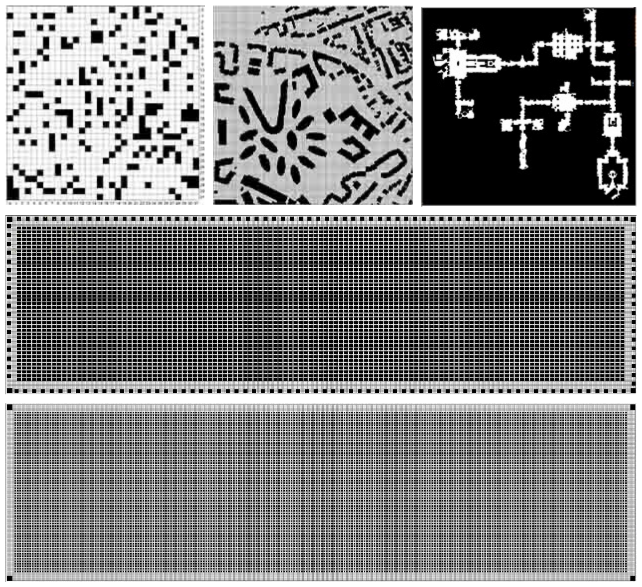

Figure 3: Test environments. From the top and left, (1) Random, (2) City, and (3) Game, (4) Warehouse, and (5) Sortation maps.

Experiments

We evaluate our method in five environments provided by the competition: Sortation, Warehouse, Game, City, and Random map as shown in Figure 3. Each instance has predetermined agent-task pairs. We vary the number of agents from 50 to 3000 agents. The evaluation metric is the total number of errands completed in 5,000 seconds. The test system is with AMD Ryzen 5800X 3.8GHz CPU and 32G RAM. The source code is written in C++17.

As shown in Figure 4, our method effectively solves instances in the random map environment up to 200 agents. In other environments, the solution quality remains reasonable for instances with up to 1,200 agents but declines as the number of agents increases.

The ability of MAPF-LNS2 and RHCR to solve large instances allows our algorithm to handle scenarios with a significantly large number of agents. The Puzzle Heuristics

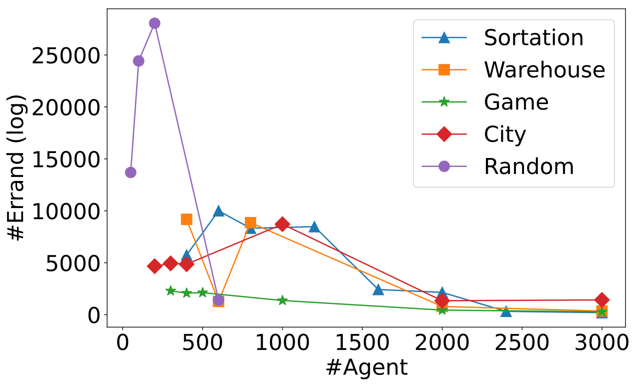

Figure 4: The number of completed errands by the proposed method in five challenging environments: Sortation, Warehouse, Game, City, and Random, with the number of agents ranging from 50 to 3000.

plays a crucial role in reducing the memory consumption and computation time for large-sized maps like Sortation, Warehouse, and Game. The greedy randomized search allows us to escape deadlocks effectively even in dense environments like Sortation, Warehouse, and Random.

The algorithm struggles to find individual paths of all agents within the time budget, particularly with a large number of agents owing to the characteristics of PP, which plans paths sequentially. Since higher-rank agents are considered fixed obstacles to lower-rank agents, the feasible positions becomes scarce.

We observe a trade-off in choosing the value of w, which is the planning horizon w in RHCR. A low value of w leads to efficient planning but could cause deadlocks because agents may not be able to plan far enough ahead to avoid conflicts. If we choose a high value, the ability to avoid deadlocks is improved. Balancing between these two cases is critical for the performance of algorithm.

Conclusion

In this paper, we presented a novel approach for solving the challenging lifelong MAPF problem in the context of the League of Robot Runners competition. Our method combines three key techniques: Puzzle Heuristics, MAPF-LNS2, and RHCR. Additionally, we introduced a greedy randomized search to escape from deadlock. Experimental results demonstrated the effectiveness of our approach in various environments, particularly in random maps with a moderate number of agents. Through participation in the competition, we found our future directions that include improving the scalability of the method and implementing methods to avoid deadlock situations.

References

Harabor, D. 2023. League Of Robot Runner. https://www.leagueofrobotrunners.org. [Online].

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10256–10265.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11272–11281.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE international conference on robotics and automation*, 5628–5635. IEEE.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, 417–431. Springer.

Silver, D. 2005. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, 117–122.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.