Caching-Augmented Lifelong Multi-Agent Path Finding

Yimin Tang[1*], Zhenghong Yu[2*], Yi Zheng[1], T. K. Satish Kumar[1], Jiaoyang Li[3], Sven Koenig[1]

***Abstract*— Multi-Agent Path Finding (MAPF), which involves finding collision-free paths for multiple robots, is crucial in various applications. Lifelong MAPF, where targets are reassigned to agents as soon as they complete their initial targets, offers a more accurate approximation of real-world warehouse planning. In this paper, we present a novel mechanism named Caching-Augmented Lifelong MAPF (CAL-MAPF), designed to improve the performance of Lifelong MAPF. We have developed a new type of map grid called cache for temporary item storage and replacement, and created a locking mechanism to improve the planning solution's stability. A task assigner (TA) is designed for CAL-MAPF to allocate target locations to agents and control agent status in different situations. CAL-MAPF has been evaluated using various cache replacement policies and input task distributions. We have identified three main factors significantly impacting CAL-MAPF performance through experimentation: suitable input task distribution, high cache hit rate, and smooth traffic. In general, CAL-MAPF has demonstrated potential for performance improvements in certain task distributions, map and agent configurations.**

I. INTRODUCTION

Automated warehouses represent a multibillion-dollar industry led by corporations such as Amazon, Ocado, and inVia. These facilities deploy hundreds of robots to transport goods from one location to another [1]. Planning collision-free solutions for hundreds of robots is a key component of these automated warehouses, and can be simplified to a multi-agent path finding problem (MAPF) [2]. The MAPF problem requires planning collision-free paths for multiple agents from their start locations to pre-assigned target locations in a known environment while minimizing a given cost function. Many algorithms have been developed to solve this problem optimally and suboptimally, such as Conflict Based Search (CBS) [3], M^*, [4] and Enhanced CBS (ECBS) [5].

Although MAPF is classified as an NP-hard problem [6], it still represents a simplistic approximation of actual warehouse planning. It is the 'one-shot' version of the real application challenge, where an agent only needs to reach one target location and remains stationary until every agent has arrived at their respective targets. To address this limitation, a new problem called Multi-Agent Pickup and Delivery (MAPD) or Lifelong MAPF [7] has been introduced. This version assigns new targets to agents once they reach their current targets, reflecting continuous operational scenarios better. Various algorithms, such as RHCR [8], MAPF-LNS [9], PIBT [10], and LaCAM [11], have been created and applied in practical environments.

Although many studies focus on developing effective algorithms, there is also interest in improving the throughput of automated warehouses within the Lifelong MAPF framework by optimizing warehouse layouts. Prominent approaches include Fishbone design [12], DSAGE [13], and GGO [14]. However, these works mainly focus on static storage design and do not consider input distributions with specific patterns that may change over time. To address this problem, some studies have explored intelligent decision-making strategies [15], [16], [17]. Notable examples include Kiva system and various policies, such as the velocity-based policy [16], and mixed-integer optimization and heuristic methods [18]. However, these methods are designed for specific warehouse systems, making applying them to different warehouses difficult. They also require implementing complex rearrangement policies for the entire storage.

In this paper, we present a dynamic cache delivery mechanism named Caching-Augmented Lifelong MAPF (CAL-MAPF), inspired by cache design, a foundational idea extensively utilized in computer architecture, databases, and various other domains. This cache mechanism is adaptable to multiple Lifelong MAPF algorithms, warehouse storage strategies, and incoming task distributions. We developed a new map grid type that allows items to be temporarily stored and replaced. We also designed a lock mechanism to enhance the stability of the planning solution and a task assigner (TA), which allocates target locations to agents and controls agent status in different situations. We have evaluated this cache mechanism using different cache replacement policies and input task distributions. Through experimentation, we identified three main factors that significantly impact CAL-MAPF performance: input task distribution, cache hit rate, and map design. In general, CAL-MAPF has demonstrated potential for performance improvements in certain task distributions, map and agent configurations.

II. PROBLEM DEFINITION

Lifelong Multi-Agent Path Finding (Lifelong MAPF) problem presents unique challenges due to the dynamic and unending nature of the tasks. Let $I = \{1, 2, \cdots, N\}$ denote a set of N agents. $G = (V, E)$ represents an undirected graph, where each vertex $v \in V$ represents a possible location of an agent in the workspace, and each edge $e \in E$ is a unit-length edge between two vertices that moves an agent from one vertex to the other. Self-loop edges are allowed, which represent "wait-in-place" actions. Each agent $i \in I$

*Equal Contribution

[1]University of Southern California, 3650 McClintock Ave, Los Angeles, CA 90089 {yimintan,yzheng63}@usc.edu, tkskwork@gmail.com, skoenig@usc.edu

[2]ShanghaiTech University, 393 Middle Huaxia Road, Shanghai 201210 yuzhh1@shanghaitech.edu.cn

[3]Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213 jiaoyanl@andrew.cmu.edu

Fig. 1: Caching-Augmented Maps: (1) Blue grids represent Shelves S. (2) Purple grids represent Caches C. (3) Green grids represent unloading ports U. The upper map has multiple ports, while a single port is in the bottom one. In the multi-port map, each unloading port has an independent cache area, task queue, and agents. The cache areas are near the unloading ports within ± 2 rows. For the single-port map, the port can utilize all agents and all caches. Given that the number of cache grids can affect the cache hit rate, we also tested different numbers of cache grids, ranging from 80 to 16, by removing cache grids column by column from right to left.

has a unique start location $s_i \in V$. Assume there is an task queue $Q = [q_1, q_2, ...]$. For each task $q_i \in Q$, $q_i = (idx, loc)$ represents a certain kind of item with type number idx should be sent to position loc. This idx item should be located in a reachable vertex in V where at least one agent can find a path to the vertex. Each item must be delivered to a vertex in V as the user specifies. These tasks are generated from a specific probability distribution to enhance realism.

We study an online setting in which incoming tasks are not known in advance. It is assumed that there is a task assigner (external to MAPF algorithm). The TA determines the specific target locations for agents based on tasks in Q. In our paper, TA is considered naive and will return the target location based on the first unassigned task in Q for each spare agent. Each agent $i \in I$ starts from a location $s_i \in V$ and its target location depends on the TA. Each action of agents, which could be waiting in place or moving to an adjacent vertex, takes a unit of time. An agent's path, $p^i = [v_0^i, v_1^i, \ldots, v_{T_i}^i]$, tracks the sequence of vertices traversed by agent i from its start to its target. As agents complete their paths at their assigned targets, they are immediately assigned new target locations, reflecting warehouse operations' perpetual and iterative nature.

As shown in Figure 1, we primarily focus on 2D warehouse layout maps and categorize all non-obstacle map grids into three types: blue grids $B = \{b_i | b_i \in V\}$ for shelves that store items, white grids $W = \{w_i | w_i \in V\}$ for normal aisle grids, and green grids $U = \{u_i | u_i \in V\}$ for unloading ports where agents deliver items to complete a task. Agents can enter B positions only from W positions. Direct transitions between two positions in B are not allowed.

III. Related work

A. MAPF

(One-Shot) MAPF, which has been proved an NP-hard problem with optimality [6], has a long history [19]. This problem is finding collision-free paths for multiple agents while minimizing a given cost function. It has inspired a wide range of solutions for its related challenges. Decoupled strategies, as outlined in [19], [20], [21], approach the problem by independently planning paths for each agent before integrating these paths. In contrast, coupled approaches [22], [23] devise a unified plan for all agents simultaneously. There also exist dynamically coupled methods [3], [24] that consider agents planned independently at first and then together only when needed in order to resolve agent-agent conflicts. Among these, Conflict-Based Search (CBS) algorithm [3] stands out as a centralized and optimal method for MAPF, with several bounded-suboptimal variants such as ECBS [5] and EECBS [25]. Some suboptimal MAPF solvers, such as Prioritized Planning (PP) [26], [19], PBS [27], PIBT [10] and their variant methods [28], [29], [11], [30] exhibit better scalability and efficiency.

B. Lifelong MAPF

Compared to the MAPF problem, Lifelong MAPF continuously assigns new target locations to agents once they have reached their current targets. Agents don't need to arrive at their targets simultaneously in Lifelong MAPF. There are mainly three kinds of methods: solving the problem as a whole [31], using MAPF methods but replanning all paths at each specified length timestep [8], [10], and replanning only when agents reach their current target locations and are assigned new targets [32], [33], [11], [30]. Among them, MAPF-LNS [9], [29] and LaCAM [11], [30] are the leading methods.

C. Cache

The cache [34] serves as a crucial component in computer science designed to temporarily store data, enhancing the efficiency of future request processing. Its main goal is to speed up access to frequently used data, thus minimizing dependence on slower storage disks. Popular caching policies include Least Recently Used (LRU) [35] and First-In-First-Out (FIFO) [36], with studies indicating LRU's superiority over FIFO [37], [38], [39]. The caching concept is widely applied in various fields, such as Databases [40] and CDN [41].

D. Warehouse Storage Strategy

Automated Storage and Retrieval Systems (AS/RS) have attracted attention for their potential to enhance warehouse efficiency and reduce operational costs [42], [43]. Various strategies for assigning items to storage locations have been widely adopted and evaluated [43], [44]. The random storage policy allocates each item type to a randomly chosen storage location and offers high space utilization [45]. The closest

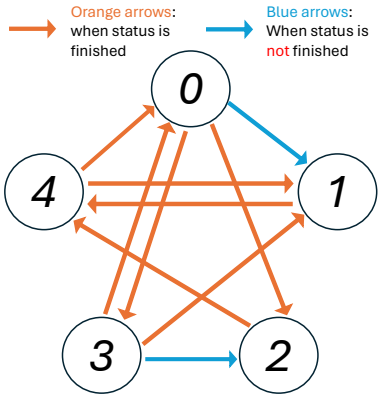

Fig. 2: **Status 0**: If the agent's task item is not in the cache when assigned, it must retrieve the item from a shelf. **Status 1**: If the agent's task item is in cache when assigned, it should retrieve the item from the cache. **Status 2**: If at least one writable cache exists when the agent retrieves the item from the shelf, the agent needs to insert the item into cache. **Status 3**: If no writable cache is available when an agent gets the item from the shelf, the agent goes to the unloading port. **Status 4**: After retrieving or inserting the item from/to the cache, the agent heads to the unloading port.

storage policy places new items at the nearest available storage location to minimize the immediate travel distance [46], [47]. The turnover-based storage policy assigns items to storage locations based on their demand frequency [16], [17].

IV. Method

In this section, we introduce our Caching-Augmented Lifelong MAPF (CAL-MAPF) framework along with a new map grid design, a TA and a cache lock mechanism.

A. Cache Grid

As shown in Figure 1, we assume each shelf $b_i \in B$ stores an infinite number of only one unique type of item. There will be a total of M different types of items corresponding to the number of shelves. Each unloading port u_k is associated with an independent task queue Q_k. Each task $q_i \in Q_k$ must be delivered to the unloading port U_k. We will also involve a new type of map grid: caches. These additional vertices, designated as $C = \{c_i | c_i \in V\}$ in purple, serve as interim storage areas to streamline the completion of tasks by reducing the travel time of the agents retrieving items.

To the best of our knowledge, no previous MAPF works have explored this new cache mechanism, so we adopt the following simplifying assumptions: (1) The warehouse has an unlimited supply of items, eliminating worries about stock running out. (2) Agents have infinite capacity to carry items. However, they are limited to transporting one type of item at a time, enabling them to move any quantity of that specific item without restriction. (3) Items removed or evicted from the cache are treated as if they vanish immediately.

B. Task Assigner

The task assigner (TA) is external to MAPF algorithm but can define all agent's target locations at any timestep. When the TA encounters a new task in Q requiring completion, the first step typically determines where to get the item and which location should be allocated to an available agent. So, the TA first checks whether the task item exists in the cache grids. If so, TA will assign an agent with the cache location of the item. After the agent gets the item, TA then assigns the unloading port location to it. If not in cache grids, TA will give the item's shelf location to the agent and then one insertable cache grid location. Subsequently, the item is stored in the cache grid. Once stored, the agent can transport the item from the cache to an unloading port.

However, similar to caching in computer architecture, there is a risk that other agents could replace the item in the cache with other items in multi-agent scenarios. This situation could result in MAPF algorithm, such as LaCAM, generating unusual collision-free plans at each timestep, causing agents to move back and forth between two adjacent grids. Thus, a lock mechanism ensures agents can secure an item after confirming its availability in cache grids.

C. Cache Lock Mechanism

In the CAL-MAPF framework, TA has a cache lock mechanism supported by a state machine to ensure efficient and synchronized access to cache locations. This approach effectively prevents race conditions during cache interactions. The mechanism uses read locks for agents fetching items and writes locks for agents updating the cache. Every cache grid will have its independent read lock and write lock. This streamlined system guarantees that agents can access and modify cache contents safely, facilitating seamless task execution within the dynamic warehouse environment.

1) Lock Types: The cache lock mechanism is designed around two principal lock types, facilitating controlled access to the cache locations: (1) **Read Lock (Shared Lock):** This type of lock permits an agent to access a cache location to retrieve items. Multiple agents can simultaneously hold a read lock on a single cache location as long as no write lock is active on that location. This arrangement ensures that when agents need to retrieve items from the cache, the item remains unchanged until all agents have successfully retrieved it. (2) **Write Lock (Exclusive Lock):** An agent is required to secure a write lock prior to inserting an item into a cache location. Write locks are exclusive, implying that once an agent possesses a write lock on a cache location, no other agent can acquire either a read lock or another write lock on that location. This ensures that an agent can insert an item without impacting other agents.

2) Lock Acquisition and Release: The mechanism defines a protocol for lock acquisition and release: (1) **Acquisition:** To obtain a read lock, an agent must verify no write lock is active on the desired cache location. Conversely, to secure a write lock, an agent must confirm that the target cache location is free from any active read or write locks. (2) **Release:** Agents release read locks after successfully retrieving items. Agents release write locks once they have updated the cache, making the inserted new items available to others.

The cache lock mechanism ensures that no agent needs to return to shelf when its target location is in cache or

Algorithm 1 CAL-MAPF with One Group Overview

Input: Map G, Initial Locations $S = s_i$, Task Queue Q, Agents A

```
portLoc = getPortLoc(G)
for agent in agents do
    agent.startLoc = s_i
    agent.task = Q.pop()
    agent.targetLoc = getMapLoc(agent.task)
    agent.status = 0
while not Q.empty() do
    plan = MAPF(agents)
    agents.excute(plan)
    AgentReleaseLocks(agents, cache)
    AgentGetLocks(agents, cache)
```

Algorithm 2 Task Assigner Functions

```
function AGENTRELEASELOCKS(agents, cache)
    for agent in agents do
        agent.startLoc = plan.endLoc[agent]
        if agent.targetLoc == agent.startLoc then
            if agent.status == 1 or 2 then
                agent.status = 4
                cache.releaseAllLock(agent)
                agent.targetLoc = portLoc
function AGENTGETLOCKS(agents, cache)
    for agent in agents do
        if agent.status == 0 then
            if agent.targetLoc == agent.startLoc then
                targetLoc = cache.insert(agent)
                CheckTarget(agent, targetLoc, 3, 2)
            else
                targetLoc = cache.check(agent)
                CheckTarget(agent, targetLoc, 0, 1)
            continue
        if agent.status == 3 then
            if agent.targetLoc == agent.startLoc then
                AgentReachPort(agent)
            else
                targetLoc = cache.insert(agent)
                CheckTarget(agent, targetLoc, 3, 2)
            continue
        if agent.status == 4 then
            if agent.targetLoc == agent.startLoc then
                AgentReachPort(agent)
function AGENTREACHPORT(agent)
    agent.task = Q.pop()
    targetLoc = cache.check(agent)
    CheckTarget(agent, targetLoc, 0, 1)
function CHECKTARGET(agent, targetLoc, statusa, statusb)
    agent.status = statusa
    if targetLoc != portLoc then
        agent.status = statusb
    agent.targetLoc=targetLoc
```

Algorithm 3 Cache Operation Functions

```
function RELEASEALLLOCK(agent)
    cacheGrid = findCache(agent.startLoc)
    cacheGrid.item = agent.task
    cacheGrid.readLock.remove(agent)
    cacheGrid.writeLock.remove(agent)
    incoming.remove(taskItemId)
function CHECK(agent)
    for cacheGrid in cache.allGrids do
        noWrite = cacheGrid.writeLock.empty()
        hasItem = cacheGrid.item == agent.task
        if noWrite and hasItem then
            cacheGrid.readLock.add(agent.id)
            return cacheGrid.loc
        return portLoc
function INSERT(agent)
    isInCache = cache.findItem(agent)
    isIncoming = incoming.find(agent.task)
    if isInCache or isIncoming then
        return portLoc
    cacheGrid = cache.findEmptyCache()
    if cacheGrid != -1 then
        cacheGrid.writeLock.add(agent.id)
        return cacheGrid.loc
    candidates = sortByCacheEvitedPolicy(cache.allGrids)
    for cacheGrid in candidates do
        if cacheGrid.readLock.empty() then
            if cacheGrid.writeLock.empty() then
                incoming.append(agent.task)
                cacheGrid.writeLock.add(agent.id)
                return cacheGrid.loc
    return portLoc
```

unloading port, no two agents access the same cache block concurrently for writing, and multiple agents may read from the same cache block concurrently if no write lock is active. This concurrency control is critical to prevent race conditions and maintain the cache's consistency.

D. Algorithm

As shown in Algorithms 1 to 3, we will introduce the whole procedure of CAL-MAPF. It is mainly based on the TA, who encounters new tasks and decides what locations the agents need to visit. As illustrated in Figure 1, CAL-MAPF could have multiple unloading ports and we call each unloading port and cache around the port a group. Each group is independent in task queue, cache grids, and agents. Agents in one group can only accept tasks from this group and use cache grids of this group. Algorithm 1 illustrates the operation of one group.

TA uses a state machine to control agent status and how they transfer to another based on different situations. When any agent reaches its target location, TA must check all agents, update their status, and assign new target locations. The state machine is illustrated in Figure 2, and the pseudocode of Algorithm 1 outlines how an agent transitions through five states while performing the MAPF algorithm for a group. The TA monitors and updates the status of agents based on the item's location, the state of the cache, and the agents' current status. This algorithm ensures the efficient completion of all tasks by orchestrating the agents' paths and their interactions with the cache and unloading port.

Overall, CAL-MAPF needs to use TA to assign target locations to all spare agents based on Q. Then invoke MAPF solver to find a collision-free solution and execute it until at least one agent reaches its target location. Then TA will check all agent's statuses, release locks and change agent status following the state machine. When changing status, TA needs to use tasks in Q and cache replacement policy to

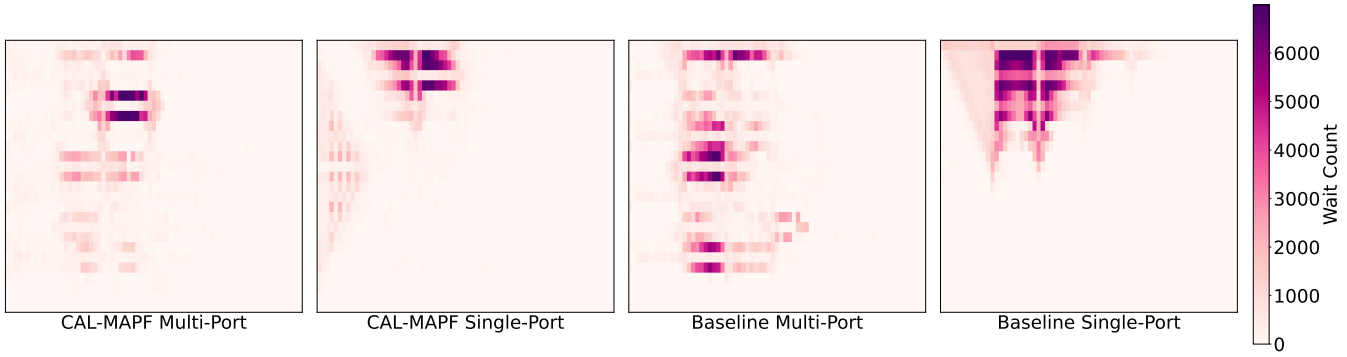

Fig. 3: The average frequency of agent wait actions on each map grid with 256 agents under Zhang distribution. As the agent number is large enough, both CAL-MAPF and baseline experience severe traffic congestion. The congestion position bias in the map could be caused by the randomization of item indexes and tasks in Q.

determine target locations for agents and acquire locks for agents. CAL-MAPF repeats this process during the lifelong scenario.

In Algorithm 1, we first set the status of each agent to 0 and assign them initial target locations (Lines 1-6). Next, we obtain collision-free paths for each agent by invoking a MAPF algorithm. The plan will be executed until at least one agent reaches its target location (Lines 7-9). Upon arrival, TA updates the lock information, attempts to assign new target locations to available agents, and updates the status of all agents based on the current warehouse situation (Lines 10-11).

TA should first attempt to release locks to allow other agents the opportunity to access cache grids. Status 1 is trying to read a cache grid, and Status 2 is going to write. Only agents in status 1 and status 2 can hold locks. After reaching their target location, they must go to the unloading port to deliver items. So, we verify whether these agents have reached their targets before releasing their held locks (Algorithm 2 Lines 1-8).

Then we evaluate whether other agents can alter their status. For agents in status 0, heading to shelves, we examine if they have reached the shelves and gotten items. We then check if they can do a write operation in the cache and apply a cache replacement policy to select a cache grid. Thus, status 0 may transition to status 2 (write to cache) or Status 3 (proceed directly to unloading port) (Algorithm 2 lines 9-14). If they have not yet arrived, we check whether they can retrieve an item from the cache, which would change their status to 1 (read cache) (Algorithm 2 Lines 15-18). We do the same status changes for agents in Status 3 and Status 4 as we describe in Figure 2 (Algorithm 2 Lines 19-28).

In Algorithm 3, we detail the process of reading or insertion (writing) a new item into cache grids using a cache replacement policy. For the read operation, it is necessary to verify if the item exists in the caches and if there is an existing write lock on the cache grid (Algorithm 3 lines 7-14). For the write operation, we first check whether the item is already in the cache area or if another agent plans to insert the same item into the caches (Algorithm 3 lines 15-19). Then, we determine if an available cache exists for inserting the item (Algorithm 3 Lines 20-31).

V. Experimental Results

We use Lifelong MAPF without cache as a baseline, which aligns with our problem definition. To evaluate performance, we compare CAL-MAPF with Lifelong MAPF, both using LaCAM as the MAPF solver for generating collision-free paths. CAL-MAPF and Lifelong MAPF were implemented in C++, building on parts of the existing LaCAM codebase[1] All experiments were conducted on a system running Ubuntu 20.04.1, equipped with an AMD Ryzen 3990X 64-core CPU and 64GB RAM at 2133 MHz.

A. Test Settings

As shown in Figure 1, we designed a warehouse map (27x71) with caches based on our problem definition. The map is adapted from the warehouse map of MAPF benchmark [2]. It includes 1600 shelf grids, a maximum of 80 cache grids and 4 unloading ports. The maximum cache-to-shelf ratio is 5%. We tested CAL-MAPF in both multi-port and single-port scenarios, as depicted in Figure 1. The multi-port scenario has 4 working groups of unloading ports and cache grids, each with a maximum of 20 cache grids. The single-port scenario maintains the same number of total cache grids and agents as the multi-port but only has one unloading port. Because each shelf grid represents a unique kind of item, and we randomly assign an index to each shelf grid. We test all scenarios with different cache numbers {16, 32, 48, 64, 80} by deleting some cache grids (described in Figure 1). We have also chosen various total numbers of agents, $\{4, 8, 16, 32, 64, 128, 192, 256\}$. In the multi-port scenario, each group has an equal number of agents $\{1, 2, 4, 8, 16, 32, 48, 64\}$.

Since we can expect the distribution of the task queue could significantly affect the performance of cache design, we designed three input task distributions to test CAL-MAPF, including: (1) M-K distribution (MK): For any consecutive subarray of length M in the task queue Q, there

[1]The LaCAM source code is publicly accessible at `https://github.com/Kei18/lacam`. Our implementation is available at `https://github.com/HarukiMoriarty/CAL-MAPF`.

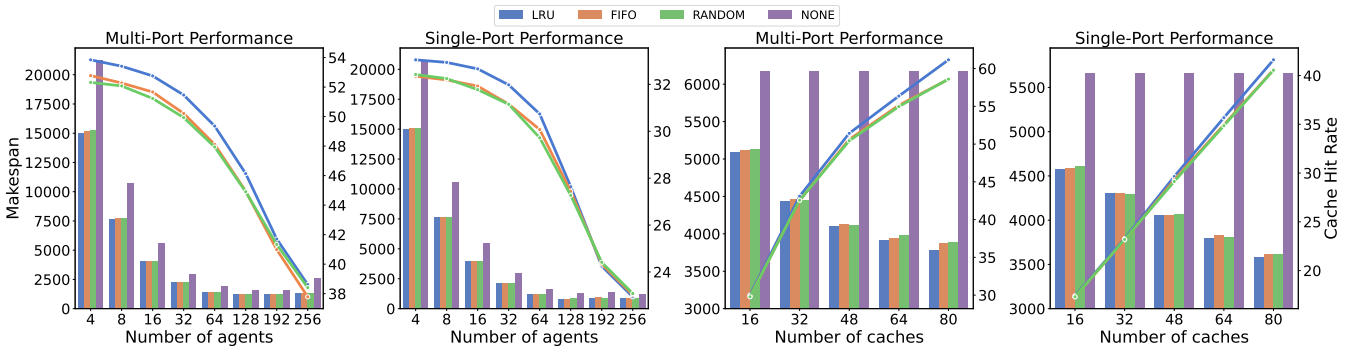

Fig. 4: Makespan (Bar chart, lower is better) and Cache Hit Rate (Line chart, higher is better). LRU, FIFO, and RANDOM represent CAL-MAPF with different cache replacement policies. NONE represents Lifelong MAPF without cache.

are at most K different kinds of items. This distribution is inspired by [37], where LRU has been proven to have an upper bound on the cache miss rate and to be better than FIFO. This distribution can also represent several items people purchase daily, and some items may become very popular at one time, replacing previously popular items. (2) 7:2:1 distribution (Zhang): There are 70% kinds of items with only a 10% appearance probability in the task queue, 20% kinds of items with a 20% probability, and 10% with a 70% appearance probability [48]. (3) Real Data Distribution (RDD): We obtain data from Kaggle's public warehouse data[2], build a probability distribution based on the frequency of data and generate tasks from this probability distribution.

We randomly generate agent start locations and Q for all input task distributions with a total length of 1,000. For the MK distribution, we select $M = 200$ and K values of $\{1, 20, 40, 80, 120, 160\}$. We allocate a total of 10 seconds for the MAPF solver to find a collision-free solution for all 1,000 tasks in Q. We use makespan, the completion time when the last task has been accomplished, to show performance. For CAL-MAPF, we have tested three different cache replacement policies. Least Recently Used (LRU), First-In-First-Out (FIFO), and RANDOM. We use NONE to represent the baseline method.

B. Performance

Many variables can influence the performance of CAL-MAPF, including the number of agents, the number of caches, and the distributions of input tasks. As illustrated in Figure 4, increasing the number of caches improves cache hit rate and makespan performance for both single- and multi-port scenarios. And the cache hit rate decreases as the number of agents grows. It is also observed that as the number of agents continues to increase (from 64 to 256), there is not much difference in the makespan for both CAL-MAPF and the baseline, and the cache hit rate is dropping.

Two reasons could contribute to this scenario: (1) CAL-MAPF and baseline's makespan remain at the same level or even worse with too many agents: traffic congestions occur in the map as the number of agents increases. (2) The CAL-MAPF cache hit rate continues to decrease as the makespan performance shows minimal differences: We employ a lock mechanism to maintain items stored in the cache area. Traffic congestion prevents many agents from reaching their target in the cache. For other agents not significantly affected by traffic congestion, they are constantly assigned new tasks. They cannot obtain locks in the cache because locks are held by agents in congestion. So the item distribution cannot update with input tasks, which ultimately leads to a continuous drop in the cache hit rate. This second reason can also clarify why CAL-MAPF performs better as the number of caches increases: enlarging the cache area mitigates traffic congestion and enhances the cache hit rate.

In Figure 5 and Figure 6, we present the performance of CAL-MAPF and the baseline for various task distributions. Figure 5 demonstrates that as the number of caches increases, CAL-MAPF's performance improves. In the MK, Real, and Zhang distributions, CAL-MAPF surpasses the baseline in most test settings. However, it is observable that as the number of agents increases, the improvement offered by CAL-MAPF diminishes. One possible reason is that the low hit rate leads agents to require longer paths to complete a task. As depicted in Figure 6, the hit rate significantly depends on the input task distribution, and the hit rate also greatly affects the final makespan performance.

Furthermore, in Figure 5, an abnormal increase in makespan is observed at baseline under the Zhang distribution with 256 agents. Traffic congestion could contribute to this abnormally poor performance. Figure 3 illustrates the average frequency of agent wait actions on each grid when the number of agents reaches 256 with the Zhang distribution. Both CAL-MAPF and baseline experience severe traffic congestion.

High cache hit rates and smooth traffic are crucial for the performance of CAL-MAPF. We can increase the number of caches to enhance the cache hit rate. And we can also directly add more agents to improve the makespan performance. However, both methods come with their drawbacks. The number of caches is constrained by the space available close to the unloading port. Introducing more agents can lead to severe traffic congestion, ultimately causing the hit rate to drop. Nevertheless, there are potential solutions to mitigate these adverse effects, such as implementing

[2]kaggle.com/datasets/felixzhao/productdemandforecasting

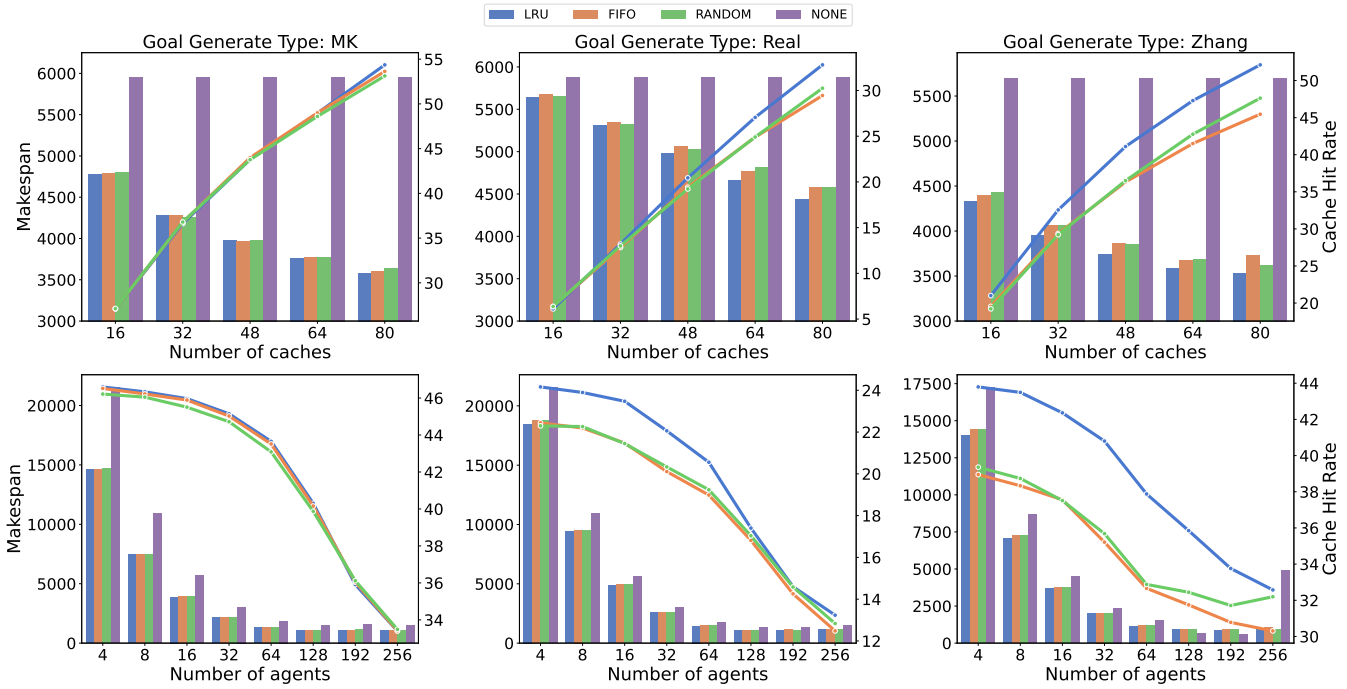

Fig. 5: Makespan (Bar chart, lower is better) and Cache Hit Rate (Line chart, higher is better). As the number of caches increases, CAL-MAPF's cache hit rate and makespan performance improve. In the MK, Real, and Zhang distributions, CAL-MAPF surpasses the baseline in most test settings. However, it is observable that as the number of agents increases, the improvement offered by CAL-MAPF diminishes. Additionally, an abnormal increase in makespan is observed at baseline under the Zhang distribution with 256 agents.

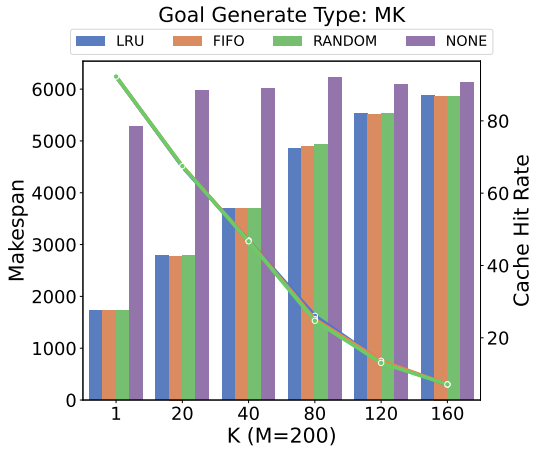

Fig. 6: Makespan (Bar chart, lower is better) and Cache Hit Rate (Line chart, higher is better). The hit rate significantly depends on the input task distribution, and the hit rate also greatly affects the final makespan performance.

more efficient map designs [13] and using one-way systems near the cache and unloading ports [14]. Additionally, as we currently employ a simple task assignment policy, we could implement a more advanced policy with predictive capabilities, leveraging real warehouse task data to enhance the cache hit rate. The cache replacement policies, such as LRU and FIFO, may be overly simplistic for CAL-MAPF. Incorporating more complex policies, including learning-based approaches, could further improve the cache hit rate, especially since CAL-MAPF allows more planning time than traditional caches in computer architecture.

VI. Conclusion

This work presents a novel mechanism, Caching-Augmented Lifelong MAPF (CAL-MAPF), designed to improve the performance of Lifelong MAPF. We have developed a new map grid type called cache for temporary item storage and replacement. Additionally, we have devised a locking mechanism for caches to improve the stability of the planning solution. This cache mechanism was evaluated using various cache replacement policies and a spectrum of input task distributions. We identified three main factors that significantly impact CAL-MAPF performance through experimentation: suitable input task distribution, high cache hit rate, and smooth traffic. In general, CAL-MAPF has demonstrated potential for performance improvements in certain task distributions and map and agent configurations.

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *Artificial Intelligence*, vol. 29, no. 1, pp. 9–9, 2008.

[2] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, no. 1, 2019, pp. 151–158.

[3] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[4] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 3260–3267.

[5] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2014.

[6] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 27, no. 1, 2013, pp. 1443–1449.

[7] H. Ma, J. Li, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017, p. 837–845.

[8] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[9] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021, pp. 4127–4135.

[10] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[11] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 10, 2023, pp. 11 655–11 662.

[12] K. R. Gue and R. D. Meller, "Aisle configurations for unit-load warehouses," *IIE Transactions*, vol. 41, no. 3, pp. 171–182, 2009.

[13] Y. Zhang, M. C. Fontaine, V. Bhatt, S. Nikolaidis, and J. Li, "Multi-robot coordination and layout design for automated warehousing," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023, pp. 5503–5511.

[14] Y. Zhang, H. Jiang, V. Bhatt, S. Nikolaidis, and J. Li, "Guidance graph optimization for lifelong multi-agent path finding," *arXiv preprint arXiv:2402.01446*, 2024.

[15] C. S. Tang and L. P. Veelenturf, "The strategic role of logistics in the industry 4.0 era," *Transportation Research Part E: Logistics and Transportation Review*, vol. 129, pp. 1–11, 2019.

[16] R. Yuan, S. C. Graves, and T. Cezik, "Velocity-based storage assignment in semi-automated storage systems," *Production and Operations Management (POM)*, vol. 28, no. 2, pp. 354–373, 2019.

[17] X. Li, G. Hua, A. Huang, J.-B. Sheu, T. Cheng, and F. Huang, "Storage assignment policy with awareness of energy consumption in the kiva mobile fulfillment system," *Transportation Research Part E: Logistics and Transportation Review*, vol. 144, p. 102158, 2020.

[18] F. Weidinger, N. Boysen, and D. Briskorn, "Storage assignment with rack-moving mobile robots in KIVA warehouses," *Transportation Science (TS)*, vol. 52, no. 6, pp. 1479–1495, 2018.

[19] D. Silver, "Cooperative pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 1, no. 1, 2005, pp. 117–122.

[20] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 294–300.

[21] K.-H. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2008, pp. 380–387.

[22] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 24, no. 1, 2010, pp. 173–178.

[23] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 668–673.

[24] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial intelligence*, vol. 219, pp. 1–24, 2015.

[25] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 14, 2021, pp. 12 353–12 362.

[26] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[27] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, no. 01, 2019, pp. 7643–7650.

[28] S.-H. Chan, R. Stern, A. Felner, and S. Koenig, "Greedy priority-based search for suboptimal multi-agent path finding," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 16, no. 1, 2023, pp. 11–19.

[29] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 36, no. 9, 2022, pp. 10 256–10 265.

[30] K. Okumura, "Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding," *arXiv preprint*, 2023.

[31] V. Nguyen, P. Obermeier, T. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, no. 1, 2019, pp. 194–195.

[32] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proceedings of the International Conference on Auto- mated Planning and Scheduling (ICAPS)*, vol. 25, 2015, pp. 324–332.

[33] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the International Conference on Auto- mated Planning and Scheduling (ICAPS)*, vol. 29, 2019, pp. 181–185.

[34] A. W. Burks, H. H. Goldstine, and J. Von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," in *The origins of digital computers: Selected papers*. Springer, 1946, pp. 399–413.

[35] J. McCabe, "On serial files with relocatable records," *Operations Research*, vol. 13, no. 4, pp. 609–618, 1965.

[36] W. F. K. III, "Analysis of paging algorithms," in *in Proceedings of International Conference on Information Processing (IFIP) Congress*, 1971, pp. 485–490.

[37] S. Albers, L. M. Favrholdt, and O. Giel, "On paging with locality of reference," in *ACM Symposium on Theory of Computing (STOC)*, 2002, pp. 258–267.

[38] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 1990, pp. 143–152.

[39] M. Chrobak and J. Noga, "Lru is better than fifo," *Algorithmica*, vol. 23, pp. 180–185, 1999.

[40] M. Altınel, C. Bornhövd, S. Krishnamurthy, C. Mohan, H. Pirahesh, and B. Reinwald, "Cache tables: Paving the way for an adaptive database cache," in *Proceedings 2003 VLDB Conference*. Elsevier, 2003, pp. 718–729.

[41] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource discovery in distributed networks," in *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, 1999, pp. 229–237.

[42] J. Gu, M. Goetschalckx, and L. F. McGinnis, "Research on warehouse operation: A comprehensive review," *European Journal of Operational Research (EJOR)*, vol. 177, no. 1, pp. 1–21, 2007.

[43] K. J. Roodbergen and I. F. A. Vis, "A survey of literature on automated storage and retrieval systems," *European Journal of Operational Research (EJOR)*, vol. 194, no. 2, pp. 343–362, 2009.

[44] K. Azadeh, R. de Koster, and D. Roy, "Robotized and automated warehouse systems: Review and recent developments," *Transportation Science (TS)*, vol. 53, no. 4, pp. 917–945, 2019.

[45] B. C. Park, "An optimal dwell point policy for automated storage/retrieval systems with uniformly distributed, rectangular racks," *International journal of production research*, vol. 39, no. 7, pp. 1469–1480, 2001.

[46] M. Fukunari and C. Malmborg, "A heuristic travel time model for random storage systems using closest open location load dispatching," *International Journal of Production Research*, vol. 46, no. 8, pp. 2215–2228, 2008.

[47] J.-P. Gagliardi, J. Renaud, and A. Ruiz, "On storage assignment policies for unit-load automated storage and retrieval systems," *International Journal of Production Research*, vol. 50, no. 3, pp. 879–892, 2012.

[48] Y. Zhang, "Correlated storage assignment strategy to reduce travel distance in order picking," *IFAC-PapersOnLine*, vol. 49, no. 2, pp. 30–35, 2016.