Adaptive Congestion-Based Algorithms for Multi-Goal Task Assignment and Path Finding in Large-Scale Multi-Agent Systems

Ye Gao[1], Hang Ding[2], Yuxuan Wang[3], Junjie Zhang[4], Qian Sun[5], Qian Zhang[6], Yiwen Huang[7], Mao Luo[8], Zhouxing Su[9], Junwen Ding[10], Zhipeng Lü[11]

[1]School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China
[2,8]School of Computer Science, Hubei University of Technology, Wuhan, China
[3,4,5,6,7,9,10,11]School of Computer Science, Huazhong University of Science and Technology, Wuhan, China
gaoye@hust.edu.cn, 102411256@hbut.edu.cn, yxuanwkeith@outlook.com, junjie_zhang@hust.edu.cn, 2500658103@qq.com, nana_103@163.com, 453992179@qq.com, luomao@hbut.edu.cn, suzhouxing@hust.edu.cn, junwending@hust.edu.cn, zhipeng.lv@hust.edu.cn

Abstract

The Multi-Goal Task Assignment and Path Finding (MG-TAPF) problem involves coordinating a team of agents to execute a set of tasks, each characterized by a specific release time and an ordered sequence of goal locations. Completing a task requires an agent to visit all the goal locations in the prescribed order, starting no earlier than the release time of the task, while avoiding collisions with other agents. The core challenge lies in efficiently assigning tasks to agents and planning collision-free paths for their execution.

Traditional approaches to the Task Assignment and Path Finding (TAPF) problem often fail to achieve timely and efficient task assignment, especially in large-scale multi-agent systems. To address this limitation, we propose several adaptive congestion-based methods that enable rapid task assignment under tight time constraints in scenarios involving large numbers of agents and tasks. These methods estimate the delay each free agent would incur in completing each available task by assessing congestion around the task's first goal location, leveraging this delay to predict task completion times across all agents.

Experimentally, our approaches demonstrate the best performance on large-scale instances, highlighting their effectiveness in addressing the complexities of MG-TAPF. We earn second place on the Task Scheduling Track of 2024 League of Robot Runners (LoRR) competition.

Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding conflict-free paths for multiple agents to move from their start locations to their goals while minimizing the total travel times of all agents (Stern et al. 2019). MAPF has a wide variety of real-world applications, including unmanned aerial vehicle management (Ho et al. 2019), video game action planning (Ma et al. 2017b), and automated warehouse operation (Li et al. 2021; Varambally, Li, and Koenig 2022).

A specialized variant of MAPF is the Combined Target Assignment and Path Finding (TAPF) problem, which integrates the assignment of target locations to agents with the planning of collision-free paths to these targets (Ma and Koenig 2016). Another similar variant is Multi-Agent Pickup-and-Delivery (MAPD), in which each agent undertakes tasks characterized by pickup locations, delivery destinations, and release times.

The Multi-Goal Task Assignment and Path Finding (MG-TAPF) problem further refines TAPF by introducing tasks with ordered sequences of goal locations and release times. For successful task execution, agents must visit each goal location of the task sequentially, starting no earlier than its release time, while strictly avoiding collisions with other agents.

During the past decades, many efforts have been made to handle task assignment in TAPF. For example, Ma et al. (Ma et al. 2017a) proposed using the Hungarian method (Kuhn 1955) to assign tasks. However, previous methods may fail to solve the problem within a reasonable time, thus limiting the overall effectiveness of large-scale multi-agent systems.

Therefore, the League of Robot Runners competition (LoRR 2024), sponsored by Amazon Robotics, was organized online in 2024 to examine more challenging MG-TAPF settings, including a large number of agents of up to 10,000 and limited planning time (including task scheduling and path planning) of 1 second per timestep. Inspired by these challenges, we propose several adaptive congestion-based methods that improve MG-TAPF performance with respect to task assignment.

Experimental evaluations on the 10 benchmark instances of LoRR (LoRR Benchmark 2024) demonstrate the effectiveness of these methods, particularly in large-scale sortation and warehouse environments, where traditional algorithms often falter.

Related Work

The TAPF problem is a combination of task assignment and path finding. So TAPF methods can be categorized into coupled and decoupled approaches. Coupled approaches consider task assignment and path planning jointly. Ma and Koenig (2016) present the CBM (Conflict-Based Min-Cost-Flow) algorithm, which is a hierarchical algorithm that uses CBS to resolve conflicts among agent teams at the high level and the min-cost max-flow algorithm to assign targets and plan paths at the low level. Nguyen et al. (2019) solved MAPF with answer set programming. Hönig et al.

(2018) proposed CBS-TA, which solves the problem optimally by extending CBS to operate on a search forest. Zhong et al. (2022) proposed Conflict-Based Search with Task Assignment with Multi-Label A* algorithm (CBS-TA-MLA), which uses CBS-TA (Hönig et al. 2018) on the high level and MLA* (Grenouilleau, Van Hoeve, and Hooker 2019) on the low level. Tang et al. (2023) developed Incremental Target Assignment CBS (ITA-CBS) to avoid computing K-best assignments, thus running faster than CBS-TA.

Decoupled approaches consider task assignment and path planning independently. Ma et al. (2017a) proposed CENTRAL algorithm, which used the Hungarian method (Kuhn 1955) to assign tasks and Conflict-Based Search (Sharon et al. 2015) to plan paths. Liu et al. (2019) computed task sequence for each agent by solving a special traveling salesman problem and then planned paths according to these task sequences. Kou et al. (2020) presented algorithms based on a min-cost max-flow formulation that minimizes the total idle time of stations. Xu et al. (2022) proposed LNS-PBS algorithm, which used Large Neighborhood Search (LNS) to assign tasks and Priority Based Search (PBS) to plan paths.

Problem Definition

MAPF

Let $G = (V, E)$ be a connected graph, where the set of vertices V contains all possible locations of agents and the set of edges E contains all connections between adjacent locations. If G is a grid map, a cell is called an obstacle if it is blocked. Let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ be a set of agents. Let $loc(a_i, t) \in V$ denote the location of agent a_i at discrete timestep $t \in N$. Each agent a_i has a unique parking location $p_i \in V$ assigned to it and starts at p_i at time step 0. At each timestep, an agent can move to an adjacent vertex, i.e., $(loc(a_i, t), loc(a_i, t+1)) \in E$; or wait at its current location, i.e., $loc(a_i, t) = loc(a_i, t+1)$. If G is a 4-neighbor grid map, an agent can move forward, backward, left, right, or stay.

A feasible action of an agent should avoid the following two types of conflicts. (1) A vertex conflict occurs when two agents a_i, a_j occupy the same location at the same timestep, i.e., $loc(a_i, t) = loc(a_j, t)$. (2) An edge conflict occurs when two agents a_i, a_j traverse the same edge from opposite directions at the same time, i.e., $loc(a_i, t) = loc(a_j, t+1) \wedge loc(a_i, t+1) = loc(a_j, t)$.

Path π_i is a sequence of agent a_i's actions from one location to another. $l(\pi_i)$ is the length of the path. The distance $d(x, y)$ from vertex x to vertex y is the length of the shortest path from x to y. A path is feasible if and only if its actions are all feasible. A solution is a set of conflict-free paths that navigate all agents from their start locations to their goal locations. The objective of MAPF is to find a solution with the minimum sum of individual costs (SIC), i.e., the sum of the path lengths of all agents.

MAPF with Rotations

MAPF with rotation is a variant of MAPF with different agent states and actions. Each agent has an orientation $o \in \{East, South, West, North\}$. At each step, an agent can move forward to an adjacent vertex, rotate 90° clockwise, rotate 90° counterclockwise, or stay at its current vertex.

MG-TAPF

Let $T = \{\tau_1, \tau_2, \ldots, \tau_{|T|}\}$ be a set of tasks, where each task $\tau_j \in T$ has an ordered sequence of goal locations $\{g_1^j, g_2^j, ..., g_k^j\}$ and a release time $r_i \in \mathbb{N}$. A task is open if one or more goals in the sequence have been visited. Once open, a task cannot be reassigned to another agent. We define task distance $d(\tau_j)$ as the sum of the distances between consecutive goal locations of the task $\sum_{i=0}^{k-1} d(g_i^j, g_{i+1}^j)$. We define agent-task distance $d(a_i, \tau_j)$ as the distance between the current location of agent a_i and the first goal location of task τ_j.

An agent that is not executing a task is called a free agent; otherwise, it is called an occupied agent. Each free agent can be assigned any task τ_j that is not open. To complete a task, the agent needs to visit all the goal locations of this task in order. It must be at the first goal location at or after the release time r_j to start this task. When it reaches the last goal location, it completes its task and is called a free agent again.

Competition Problem

The problem given in the LoRR 2024 competition is a combination of MAPF with Rotations and MG-TAPF. In addition, tasks do not have prescribed release times. When an old task is completed, a new task is revealed. Each free agent can be assigned at most one task. The objective of the competition problem is to complete as many tasks as possible by a given timestep.

There are three evaluation tracks. The Task Scheduling Track uses a default path planner (Chen et al. 2024) and requires participants only to design the task scheduler. The Path Planning Track uses a default task scheduler (see Algorithm 1) and requires participants only to design the path planner. The Combined Track requires participants to design both the task scheduler and the path planner.

Timestep is set to 1 second. At the end of each timestep, the planner and scheduler must return valid path plans and valid task assignments.

Congestion-Based Approach

The organizers of LoRR 2024 provide a priority-based greedy task assignment method, illustrated in Algorithm 1. The initial steps involve updating the sets of free agents A_0 and free tasks T_0 by including new free agents and tasks (Lines 1-2). The task scheduler then assigns tasks to agents sequentially based on their IDs (Line 3), ensuring that the process terminates if the allocated time runs out (Lines 4-6). For each free agent, a free task with the minimum estimated completion distance is selected, calculated as the sum of the agent-task distance and the task distance (Lines 8-14). Once a task is assigned to an agent, both the agent and the task are removed from their respective sets (Lines 15-17).

However, the greedy method has limitations in accurately estimating task completion time. The estimation approximates the minimum task completion distance as task dura-

Algorithm 1: Priority-Based Greedy Task Assignment

Input: free agent set A_0, free task set T_0
Output: task assignment result

1: $A_0 := A_0 \cup \{newFreeAgents\}$
2: $T_0 := T_0 \cup \{newFreeTasks\}$
3: **for** $a_i \in A_0$ **do**
4: **if** time runs out **then**
5: **break**
6: **end if**
7: minTaskDuration := IntMax
8: **for** $\tau_j \in T_0$ **do**
9: taskDuration := $d(a_i, \tau_j) + d(\tau_j)$
10: **if** $taskDuration < minTaskDuration$ **then**
11: $\tau_{min} := \tau_j$
12: minTaskDuration := taskDuration
13: **end if**
14: **end for**
15: assign τ_{min} to a_i
16: $A_0 := A_0 - \{a_i\}$
17: $T_0 := T_0 - \{\tau_{min}\}$
18: **end for**

tion (Line 9 in Algorithm 1), which neglects delays caused by the need to avoid collisions with other agents during task execution. Tasks with shorter estimated distances might take longer due to traffic jams in congested areas.

To address this, we developed several adaptive congestion-based methods to estimate the agent-task delay, which is defined as the difference between the minimum time an agent required to complete a task and the actual time an agent consumed to complete a task, while the minimum time an agent required to complete a task is the sum of agent-task distance and task distance. The framework of the congestion-based approach is shown in Algorithm 2. They all estimate the agent-task delay based on the congestion level around the first goal location of the task (Line 13), and their differences lie in the regions and counting objects chosen for congestion evaluation. The estimated delay value, as a penalty, is then added to the minimum task completion distance to provide a more accurate estimate of task duration (Line 14).

The adaptive congestion coefficient γ is calculated based on all the finished tasks in the past. γ is calculated using the difference between the sum of actual duration of the completed tasks and the sum of previous estimated minimum durations of the completed tasks, divided by the total number of the completed tasks (Line 3-5). This ensures dynamic and context-aware adjustment of congestion impact, improving task assignment efficiency.

Count Agent in Task-Agent Circle

Count Agent in Task-Agent Circle (CATAC) evaluates congestion around each free task by defining an task-agent circle centered at the task's first goal location with a radius equal to the agent-task distance. The number of other agents within this circle is counted (as shown in Figure 1 and Algorithm 3). The key insight here is that a higher number of other agent

Algorithm 2: Adaptive Congestion-Based Task Assignment

Input: free agent set A_0, free task set T_0
Output: task assignment result

1: $A_0 := A_0 \cup \{newFreeAgents\}$
2: $T_0 := T_0 \cup \{newFreeTasks\}$
3: totalActualDuration += newCompletedActualDuration
4: totalMinTaskDuration += newCompletedMinTaskDuration
5: γ := (totalActualDuration - totalMinTaskDuration) / numTaskFinished
6: **for** $a_i \in A_0$ **do**
7: **if** time runs out **then**
8: **break**
9: **end if**
10: minTaskDuration := IntMax
11: **for** $\tau_j \in T_0$ **do**
12: taskDist := $d(a_i, \tau_j) + d(\tau_j)$
13: $\Delta = \gamma \cdot$ AdaptiveEvaluateCongestion(a_i, τ_j)
14: taskDuration := taskDist + Δ
15: **if** $taskDuration < minTaskDuration$ **then**
16: $\tau_{min} := \tau_j$
17: minTaskDuration := taskDuration
18: **end if**
19: **end for**
20: assign τ_{min} to a_i
21: $A_0 := A_0 - \{a_i\}$
22: $T_0 := T_0 - \{\tau_{min}\}$
23: **end for**

within the task-agent circle can lead to increased delays for the free agent as it navigates toward the first goal location of the task.

Algorithm 3: Count Agent in Task-Agent Circle

Input: agent a_i, task τ_j
Output: $congestion$

1: $congestion$:= number of agents in the task-agent circle

Count Intersections between Agent-Task and Agent-Goal

The agent-task vector represents the vector from the current location of the free agent to the first goal location of the task, while the agent-goal vector represents the vector from the current location of the occupied agent to the next goal location. The key insight here is that a higher number of intersection points between an agent-task vector and multiple agent-goal vectors can lead to increased delays for the free agent as it navigates toward the first goal location of the task. Thus, we introduce the Count Intersections between Agent-Task and Agent-Goal (CIATAG) as a metric to quantify congestion. CIATAG calculates the number of intersection points between a specified free agent-task vector and the agent-goal vectors of occupied agents. This approach is illustrated in Figure 2 and is detailed in Algorithm 4.

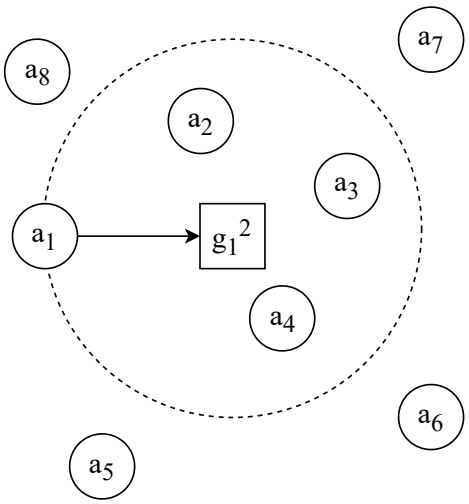

Figure 1: This figure shows congestion computation when agent a_1 selects task τ_2. Circles with solid lines represent agents' current locations. The square containing g_1^2 represent the first goal location of τ_2. Draw a circle with the first goal location g_1^2 of τ_2 as the center and the agent-task distance $d(a_2, \tau_2)$ as the radius. Count the number of other agents in this circle. There are 3 other agents in the circle range, namely a_2, a_3 and a_4.

Algorithm 4: Adaptive Delay Estimation Current First Goal Intersect Current Goal

Input: agent a_i, task τ_j
Output: $congestion$

1: $congestion$:= number of intersection points between selected free agent-task vector and occupied agent-goal vectors

Count Agent in the Task Square

In Algorithm 3 we count the number of agents in the task-agent circle to estimate the delay of each free agent in completing each free task. However, because the distance (representing the radius of the circle) between each free agent and each free task can vary, this requires counting agents across $|A_0| \times |T_0|$ circles, where $|A_0|$ is the number of free agents and $|T_0|$ is the number of free tasks. This results in inefficiency when the number of free agents exceeds 1,000. To address this challenge, we introduce the Count Agent in the Task Square (CATS), as detailed in Algorithm 5.

Rather than dynamically calculating circles, CATS divides the map into fixed $k \times k$ square regions, as shown in Figure 3. We define the square containing the task's first goal location as the task square. At each timestep, CATS determines the task square of each free task and counts the number of agents within that square (Lines 6-7) before iterating over each free agent. An intuition is that a higher number of other agent within the task square can lead to increased delays for the free agent as it navigates toward the first goal location of the task. Because these square boundaries are static, the computed agent counts for a square apply to all free agents estimating congestion in that range. This significantly reduces computational overhead by avoiding repetitive calculations, thus improving efficiency, especially in scenarios with a large number of agents and tasks.

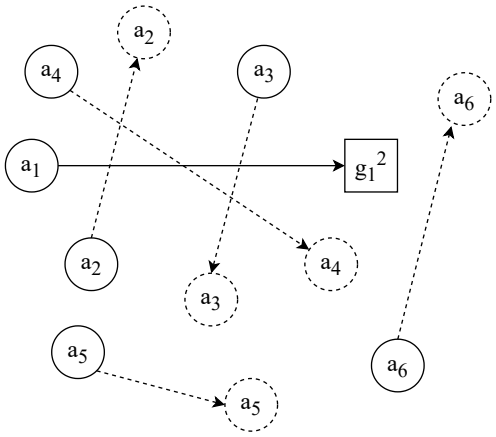

Figure 2: This figure shows congestion computation when agent a_1 selects task τ_2. Circles with solid lines represent agents' current locations. Circles with dashed lines represent agents' next goals. The square containing g_1^2 represent the first goal location of τ_2. Draw agent-goal vectors from the current location of each agent to its next goal. Draw a vector from the current location of a_2 to the first goal location g_1^2 of τ_2. Count the number of intersection points between vector $< loc(a_2), loc(p_2) >$ and the occupied agent-goal vectors. There are 3 intersection points between vector $< a2, p2 >$ with occupied agent-goal vectors, namely $< loc(a_2), goal(a_2) >$, $< loc(a_3), goal(a_3) >$ and $< loc(a_4), goal(a_4) >$.

Task Square Density Times Agent-Task Distance

The Task Square Density Times Agent-Task Distance (TSDTATD) method modifies the Count Agent in the Task Square (CATS) approach by explicitly considering the impact of static obstacles on congestion. While CATS estimates the delay of free agents completing tasks by counting the number of agents in the square region where the task's first goal point is located, it overlooks static obstacles that can significantly influence congestion and delay. In TSDTATD, both the number of agents and static obstacles are taken into account, as illustrated in Figure 4 and Algorithm 6. An intuition is that a higher number of other agent and a lower number of static obstacle within the square can lead to increased delays for the free agent as it navigates toward the first goal location of the task.

To compute congestion, agent density (denoted as ρ) is calculated as the ratio of the number of agents in the square

Algorithm 5: Count Agent in the Task Square

Input: agent a_i, task τ_j
Output: $congestion$

1: $A_0 := A_0 \cup \{newFreeAgents\}$
2: $T_0 := T_0 \cup \{newFreeTasks\}$
3: totalActualDuration += newCompletedActualDuration
4: totalMinTaskDuration += newCompletedMinTaskDuration
5: γ := (totalActualDuration - totalMinTaskDuration) / numTaskFinished
6: count the number of agent in each square
7: compute each free task first goal in which square
8: **for** $a_i \in A_0$ **do**
9: **if** time runs out **then**
10: **break**
11: **end if**
12: minTaskDuration := IntMax
13: **for** $\tau_j \in T_0$ **do**
14: taskDist := $d(a_i, \tau_j) + d(\tau_j)$
15: $\Delta = \gamma$ numAgentInTaskSquare
16: taskDuration := taskDist + Δ
17: **if** $taskDuration < minTaskDuration$ **then**
18: $\tau_{min} := \tau_j$
19: minTaskDuration := taskDuration
20: **end if**
21: **end for**
22: assign τ_{min} to a_i
23: $A_0 := A_0 - \{a_i\}$
24: $T_0 := T_0 - \{\tau_{min}\}$
25: **end for**

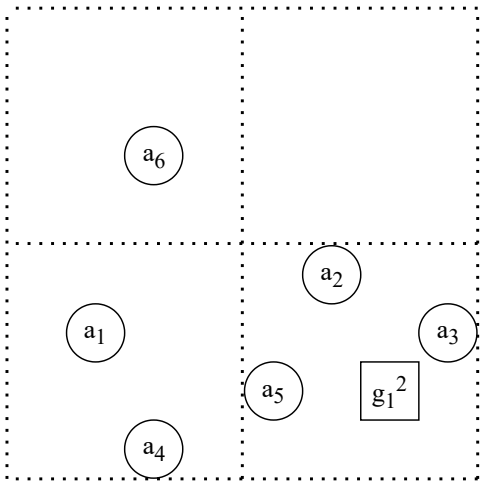

Figure 3: This figure shows congestion computation when agent a_1 selects task τ_2. Circles with solid lines represent agents' current locations. The square containing g_1^2 represent the first goal location of τ_2. Squares with dashed lines represent ranges divided on the map. Count the number of other agents in the square where the first goal g_1^2 of τ_2 is located. There are 3 other agents in the first goal square, namely a_2, a_3 and a_5.

to the number of blank cells within the square, where blank cells refer to cells not occupied by obstacles (Line 1). A greater number of static obstacles reduces the number of blank cells, thereby increasing agent density and contributing to delays in task completion. To maintain dimensional consistency with delays, the agent density is multiplied by the distance between the current location of the agent and the first goal location of the task (Line 2).

Algorithm 6: Task Square Density Times Agent-Task Distance

Input: agent a_i, task τ_j
Output: $congestion$

1: ρ = numAgentInTaskSquare / (numCellsInPickupSquare - numObstaclesInPickupSquare)
2: $congestion = \cdot \rho \cdot d(loc(a_i), loc(p_j))$

Experiments

To evaluate the effectiveness of our congestion-based approaches (CATAC, CIATAG, CATS, and TSDTATD), we compared them with three task assignment algorithms: the default priority-based Greedy algorithm, the CENTRAL algorithm (Ma et al. 2017a), and No Man's Sky (NMS), a method developed by the team that secured the first place on the task scheduler track. The partition square size used in CATS and TSDTATD is set to 12x12. Due to computational constraints, algorithms mentioned in Related Work

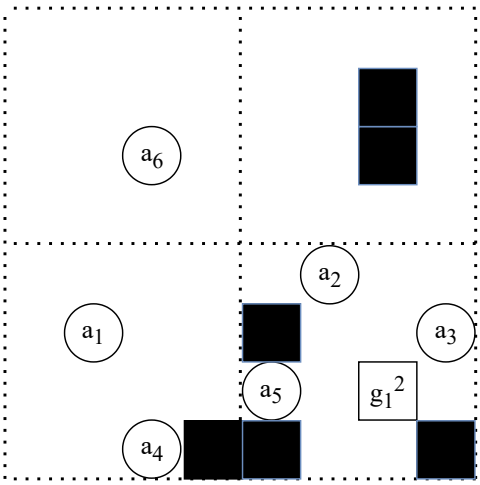

Figure 4: This figure shows congestion computation when agent a_1 selects task t_2. Circles with solid lines represent agents' current locations. The square containing g_1^2 represent the first goal location of τ_2. Black squares represent static obstacles in the map. Squares with dashed lines represent ranges divided on the map. Count the number of other agents in the square where the first goal g_1^2 of τ_2 is located. There are 3 other agents and 3 static obstacles in the first goal square.

are excluded as they cannot complete task assignment within the one-second limit. All the above task assignment algorithms share the default path planner (Chen et al. 2024). They are coded in in C++ and tested on an Ubuntu 22.04 server with an Intel Xeon Gold 6133 CPU@2.50GHz processor and 125GiB RAM. All algorithms are executed 3 times on each instance. The implementations are publicly available at https://github.com/ssfc/LoRR2024-verstand.

To ensure consistency with the LoRR evaluation platform, the tests are conducted across 10 main round instances provided by the organizers: RANDOM-01 to RANDOM-05 (small-scale), CITY-01 and CITY-02 (medium-scale), and GAME, SORTATION, and WAREHOUSE (large-scale). The agent number for each instance and the simulation timesteps are shown in Table 1. The maps used ranged from 32x32 obstacle-rich grids to complex simulated environments such as city sections, video games, and synthetic fulfillment centers.

The results in Table 1 highlight significant differences in algorithm performance across instance scales. CENTRAL performed optimally on small-instance maps and CITY-01 but faltered when the number of agents exceeded 3,000 in CITY-02, exposing limitations in handling high-density environments. NMS exhibited consistent strength on small- and medium-scale instances, achieving the best results in CITY-02, which showcased its adaptability.

For large-scale instances, congestion-based algorithms demonstrated outstanding performance. CATAC consistently outperformed Greedy in small- and medium-scale settings, but lagged behind CENTRAL and NMS. CIATAG achieved top results on the GAME map, benefiting from its ability to model vector intersection points, a distinctive feature of GAME's complex traffic dynamics. CATS ranked highest on SORTATION and second on WAREHOUSE, while TSDTATD excelled on WAREHOUSE and performed competitively in SORTATION. Both underscore their effectiveness in high-traffic synthetic environments. This is because in scenarios where large-scale agents or insufficient computational time pose challenges, rule-based methods have an advantage over search-based methods (CENTRAL and NMS). It also indicates that in cases of heavy traffic, congestion-based methods can better reflect the agent-task delay, thereby assigning more appropriate tasks to agents.

Conclusion

In this work, we introduce an adaptive congestion-based algorithm to enhance task assignment performance in large-scale multi-agent systems. We present four metrics to evaluate congestion, namely CATAC, CIATAG, CATS, and TSDTATD. By incorporating these metrics, the congestion-based methods can more accurately predict task completion times, especially in high-density environments, by reflecting real-time traffic conditions. Our experimental results demonstrate that these methods outperform traditional algorithms in scenarios with heavy traffic, offering significant potential for practical applications in automated warehousing and sorting centers.

References

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic flow optimisation for lifelong multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20674–20682.

Grenouilleau, F.; Van Hoeve, W.-J.; and Hooker, J. N. 2019. A multi-label A* algorithm for multi-agent pathfinding. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, 181–185.

Ho, F.; Goncalves, A.; Salta, A.; Cavazza, M.; Geraldes, R.; and Prendinger, H. 2019. Multi-agent path finding for UAV traffic management: Robotics track.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J.; and Ayanian, N. 2018. Conflict-based search with optimal task assignment. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*.

Kou, N. M.; Peng, C.; Ma, H.; Kumar, T. S.; and Koenig, S. 2020. Idle time optimization for target assignment and path finding in sortation centers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9925–9932.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2): 83–97.

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.

Table 1: The number of tasks completed by each approach during the simulation time on main round instances. If an algorithm fails to complete a task within the given time limit, the corresponding column is marked as 'timeout'.

	Agents	SimTime	Greedy	CENTRAL	NMS	CATAC	CIATAG	CATS	TSDTATD
RANDOM-01	100	600	490	**573**	571	555	549	490	539
RANDOM-02	200	600	780	925	**935**	887	871	771	857
RANDOM-03	400	800	972	1202	**1220**	1118	1075	943	1098
RANDOM-04	700	1000	745	**888**	875	767	765	754	777
RANDOM-05	800	2000	1122	1320	**1386**	1183	1154	1052	1185
CITY-01	1500	3000	6119	7197	**7359**	7008	7029	6130	6383
CITY-02	3000	3000	11372	2249	**14030**	12870	10431	11563	12259
GAME	6500	5000	6195	2562	7045	8061	**9003**	6024	6706
SORTATION	10000	5000	84686	timeout	17276	4744	9509	**93597**	89104
WAREHOUSE	10000	5000	62539	timeout	44350	4720	9462	71760	**75475**

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

LoRR. 2024. The League of Robot Runners. https://www.leagueofrobotrunners.org/. Accessed: 2025-05-10.

LoRR Benchmark. 2024. LoRR 2024 Benchmark. https://github.com/MAPF-Competition/Benchmark-Archive/tree/main/2024%20Competition. Accessed: 2025-05-21.

Ma, H.; and Koenig, S. 2016. Optimal target assignment and path finding for teams of agents. *arXiv preprint arXiv:1612.05693*.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017a. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868*.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T.; and Koenig, S. 2017b. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13, 270–272.

Nguyen, V.; Obermeier, P.; Son, T.; Schaub, T.; and Yeoh, W. 2019. Generalized target assignment and path finding using answer set programming. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 194–195.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Tang, Y.; Ren, Z.; Li, J.; and Sycara, K. 2023. Solving multi-agent target assignment and path finding with a single constraint tree. In *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 8–14. IEEE.

Varambally, S.; Li, J.; and Koenig, S. 2022. Which MAPF model works best for automated warehousing? In *Proceedings of the international symposium on combinatorial search*, volume 15, 190–198.

Xu, Q.; Li, J.; Koenig, S.; and Ma, H. 2022. Multi-goal multi-agent pickup and delivery. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9964–9971. IEEE.

Zhong, X.; Li, J.; Koenig, S.; and Ma, H. 2022. Optimal and bounded-suboptimal multi-goal task assignment and path finding. In *2022 International Conference on Robotics and Automation (ICRA)*, 10731–10737. IEEE.