Task Group Allocation for Solving Multi-Load Agent Pickup and Delivery problem

Anonymous Author(s)
Submission Id: «EasyChair submission id»

ABSTRACT

This paper addresses the multi-load agent pickup and delivery problem (MLAPD), in which the multi-load agent set is required to efficiently complete a set of dynamically arriving tasks. Note that each multi-load agent can carry multiple tasks simultaneously, which causes that multiple tasks may affect each other's completion time. Therefore, it is difficult to assign multiple tasks to the most suitable agents and schedule the completion order of them. To address this issue, we propose a Task Group Allocation algorithm (TGA) to minimum the completion time of all tasks. We first quantify the carpooling scores between multiple tasks, and then divide the set of tasks into groups, in which tasks in the same group have little influence on each other, via proposed K-Capacity Hierarchical Clustering Algorithm (KCHC). Finally, we allocate task groups to agents. Experimental results show that the TGA algorithm outperforms the state-of-the-art algorithms in terms of service time, makespan, and processing time.

CCS CONCEPTS

• **Computing methodologies → Multi-agent planning**.

KEYWORDS

Multi-load agent, Pickup and delivery problem, Task allocation

ACM Reference Format:
Anonymous Author(s). 2025. Task Group Allocation for Solving Multi-Load Agent Pickup and Delivery problem. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 8 pages.

1 INTRODUCTION

Recently, the multi-agent pickup and delivery (MAPD) problem has attracted increasing attention, because it can be applied to various fields, such as warehouse robots for handling goods, aircraft towing robots for aircraft movement, and office delivery robots [3, 4, 15]. In the MAPD problem, a set of agents need to complete a set of dynamically arriving tasks. When a task is released, the system needs to allocate it to an agent. Then, the agent needs to arrive at the pickup location to pick task up and delivery task to the delivery location. Note that the agent is single-load and can only carry one task once time, in the MAPD problem. Now, many platforms have used multi-load agents to complete tasks. Multi-load agents can carry multiple tasks simultaneously to adapt to the complex environment and improve the efficiency of the system [2, 7].

Existing works focusing on the task allocation of MLAPD problem can be divided into two main types. Fisrt, some works [9, 14, 17] focus on individual task assignments. These works find the connection between individual tasks and agents, and then use a special strategy for task allocation. However, matching between individual tasks and agents can lead the problem to a local optimum easily. Second, some works [1, 16] focus on the task group assignments, which divide the set of tasks into groups, Then allocate task groups to agents. However, these works have excessive computational time and no theoretical support.

There exits a main challenge in the MLAPD problem. When a agent is carrying multiple tasks and going to the pickup location or delivery location of a task, it maybe make a detour to pick up or deliver another task. So, tasks, which are allocated to the same agent simultaneously, affect each other's completion time. A unreasonable allocation of tasks to agents will lead to a longer completion time of all tasks. Therefore, it is difficult to assign multiple tasks to the most suitable agents and schedule the completion order of them, to minimum the completion time of all tasks.

To address this issue, we analyze the likelihood of carpooling between two task. Then, we evaluate the upper bound on the cost of the schedule of multiple tasks, and proof it. Based on it, we quantify the carpooling scores between multiple tasks, and propose the K-Capacity Hierarchical Clustering Algorithm (KCHC). KCHC algorithm divides the set of tasks into groups, where tasks in the same group have little influence on each other. Finally, we introduce the Task Group Allocation Algorithm (TGA) to allocate task groups to agents. Experimental results show that the TGA algorithm outperforms, compared to the state-of-the-art algorithms in terms of service time, makespan, and processing time.

The rest of this paper is organized as follows. In Section 2, we review the related work on the pickup and delivery problem. Section 3 presents the definition of MLAPD problem. In Section 4, we introduce the Task Group Allocation Algorithm to solve the MLAPD problem. We show the experimental results in Section 5 and conclude the paper in Section 6.

2 RELATED WORK

This section reviews the related works on the multi-agent pickup and delivery problem (MAPD) and the multi-load agent pickup and delivery problem (MLAPD).

2.1 MAPD problem

In MAPD problem, each agent can only carry one task once time. The MAPD problem has been widely studied in the existing works [5, 10, 14, 18].

To solve this problem, Ma et al. [5] present Token Passing (TP) and Token Passing with Task Swaps (TPTS). TP plans its paths for the agents in turn, and its task set contains all tasks that are not assigned to agents. Compared with TP, the task set of TPTS contains all unfinished tasks. For maximizing the completion ratio of tasks, Wu et al. [14] take an integrated approach, which allocates and plans a task at a time. They define the priority of tasks based on the release time and the deadline, and propose a prioritization-based framework for combined task assignment and path planning. Moreover, they utilize the boundary and pruning techniques in the proposed framework to greatly improve the computational efficiency. However, single-load agents cannot satisfy all the requirements of reality, since load capacity limitations. Different from other work, Zhou et al. [18] focus on the min-max balanced connected q-partition problem that seeks the early completion time of a multi-agent system while satisfying task constraints. This study presents the first exact Mixed Integer Linear Programming (MILP) in the literature, which can partition any graph into a given number of arbitrary components, and a genetic algorithm to solve the BCPq problem. In [10], an effective and efficient performance impact (EEPI) algorithm is proposed based on the performance impact (PI) algorithm. First, the travel time from each vehicle's initial position to its task position is minimized, allowing more time to be left for the vehicle to perform more tasks due to limited fuel. Second, the start time of each task should be close enough to its deadline so that tasks with earlier deadlines can be assigned earlier than tasks with later deadlines. However, these works focus on the single-load agent pickup and delivery problem, which cannot apply all the real-world scenarios. Therefore, we present our approach to solve the MLAPD problem efficiently.

2.2 MLAPD problem

In MLAPD problem, each agent can carry multiple tasks simultaneously. Therefore, MLAPD problem is more complex than MAPD, but it has more practical significance in real-world applications.

Sarkar et al. [6] proposed a heuristic algorithm nearest-neighbor based Clustering and Routing (nCAR) to solve the MLAPD problem. nCAR algorithm first clusters tasks based on the nearest neighbor, and then constructs a route for each cluster by mapping it to a traveling salesman problem. Zhang et al. [17] considering introduce an improved iterated greedy (IIG) algorithm. IIG algorithm first generates an initial solution by a greedy strategy, and then iteratively improves the solution by a local search strategy. Sepcially, IIG algorithm designs two rules to fast identify unfeasible solutions. In [1], two group-based distributed auction algorithms are designed to solve this task allocation problem. Guided by the auction algorithms, the agents first distributedly compute the feasible parcel groups they can serve, and then communicate to find parcel group assignments. However, group-based distributed auction algorithms calculate all possible parcel groups, which is time-consuming. To cope with uncertain scenarios, Wilde et al. [13] present an adaptive sampling algorithm that finds MAPD solutions that approximate the expected Pareto front and ensure that the solutions are statistically distinguishable. Wang et al. [9] define a novel recurrent open shop scheduling (ROSS) problem variant with unique sequence structure. New sequence and scheduling models are extended to accommodate it. Then, problem-specific additive beam Christofides (ABC) construction heuristics, greedy local search (GLS) and simulated annealing (SA) meta-heuristics are designed accordingly. You et al. [16] transform the MLAPD problem into a multi-traveler salesman problem (MTSP) with time windows, using a multi-angle K-means clustering algorithm. Then, they propose a two-stage load-balancing strategy, in which the first stage is making a large adjustment to unbalanced task set, and the second stage is making a small adjustment to the balanced task set. While these works have made significant progress in the MLAPD problem, they do not consider the impact of tasks on each other's completion time. Therefore, we present our approach to solve the MLAPD problem efficiently.

3 PROBLEM DEFINITION

In this section, we introduce the multi-load agent pickup and delivery problem, which we address in this paper. Before this, we first give the definitions of map, task, agent, comparative metrics, and MLAPD problem.

Definition 3.1 (Map). A map $G = (V, E)$ is the, whose vertices set V correspond to locations and edges set E represent routes between adjoining locations. The shortest distance of the path between two locations v_i and v_j is denoted as $dis(v_i, v_j)$.

Definition 3.2 (Task). A task $\tau \in \Gamma$ is a tuple $< v_\tau^s, v_\tau^g, t_\tau^r, t_\tau^c >$, where v_τ^s and v_τ^g are the pickup location and delivery location, respectively. t_τ^r and t_τ^c are the release and completed time, respectively.

Definition 3.3 (Agent). An agent $a \in A$ is represented as a tuple $< v_a, c_a, \Gamma_a, S_{\Gamma_a} >$, where v_a is location, c_a represents the maximum capacity, Γ_a is the set of tasks, and S_{Γ_a} is the schedule of Γ_a. The $S_{\Gamma_a} =< v_1, v_2, ..., v_n >$, where v_i is a pickup or delivery location of task $\tau \in \Gamma_a$, and the cost of schedule S_{Γ_a} is denoted as $Cost(S_{\Gamma_a}) = \sum_{i=1}^{n-1} dis(v_i, v_{i+1})$.

A task $\tau \in \Gamma$ is released at t_τ^r. When system assigns τ to an agent a, a needs to reach v_τ^s, and then deliver τ to v_τ^g while recording the completed time t_τ^c.

Definition 3.4 (Comparative metrics). In this paper, we demonstrate algorithmic superiority through three comparative metrics, service time (ST), makespan (MS) and processing time (PT). $ST = \sum_{\tau_i \in \Gamma} (t_{\tau_i}^c - t_{\tau_i}^r)$, is defined as the sum of differences between the completed times and the release times of all tasks. $MS = max\{t_\tau^c\}$ $(\forall \tau \in \Gamma)$ is time when last task has completed. PT is the average processing time within each batch.

Definition 3.5 (MLAPD). Given a map G, a set of agents A, and a set of tasks Γ, in each batch, the goal of the multi-load agent pickup and delivery problem (MLAPD) is to find the allocation of tasks to agents, which optimize three comparative indicators, (ST, MS, PT). Time is discretized into timesteps, and agents can only move to adjacent locations or wait at its current location in one timestep. To focus on the task allocation, we ignore the time required to avoid conflicts.

4 SOLUTION

In this study, Task-Group Allocation Algorithm is developed to solve MLAPD problem efficiently. The TGA method first analyzes the likelihood of carpooling between each task, and then quantifies the carpooling scores between multiple tasks to evaluate the impact of tasks on each other's completion time. Based on this, we present K-Capacity Hierarchical Clustering Algorithm (KCHC) to divide the set of tasks into groups, where tasks in the same group have little influence on each other. Finally, we introduce the Task Group Allocation Algorithm to allocate task groups to agents.

4.1 Carpooling Scores

The biggest difference between multi-load agent and single-load agent is that multi-load agent can load multiple tasks simultaneously. This means that multi-load agent performing a task affects the completion time of the "co-passenger" tasks which are also executed to the same agent at same time.

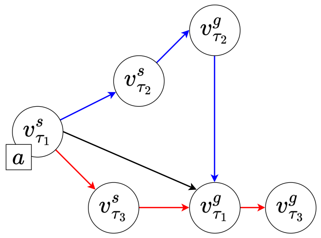


Figure 1: The impact of co-passenger tasks

Example 4.1 (Co-passenger tasks). As shown as in Figure 1, there is an agent, a, and three tasks, τ_1, τ_2 and τ_3. The agent a is located at pickup location of τ_1. The blue lines are the best schedule for agent a to complete τ_1 and τ_2. The red lines are the best schedule for agent a to complete τ_1 and τ_3. When we allocate τ_1 and τ_2 to the agent a, the agent a needs to take the blue paths to complete τ_1 and τ_2 by greedy strategy, and it will not impact the completion time of τ_2 but lengthen the completed time of τ_1. However, if we allocate τ_1 and τ_3 to the agent a, a needs to take the red paths to complete τ_1 and τ_3. While it also lengthens the completed time of τ_1, it is obvious that τ_3 has less of an effect on τ_1 as compared to τ_2. Therefore, we need to consider the impact of "co-passenger" tasks when allocating tasks to agents.

We first analyze the likelihood of carpooling between two tasks. As shown in Figure 2, there are 6 schedules to the completion of τ_i and τ_j, ignoring the location of agent. The schedules are $S^1_{i,j}$, $S^2_{i,j}$, $S^3_{i,j}$, $S^4_{i,j}$, $S^5_{i,j}$ and $S^6_{i,j}$. Therefore, we define the carpooling scores between two tasks τ_i and τ_j as follows.

Definition 4.2 (Carpooling Scores). The carpooling scores between two tasks τ_i and τ_j is defined as

$$\delta_{\tau_i,\tau_j} = min\{Cost(S^q_{i,j})\} - dis(v^g_{\tau_i}, v^s_{\tau_i}) - dis(v^g_{\tau_j}, v^s_{\tau_j}) \qquad (1)$$

where $q \in [1,6] \wedge q \in \mathbb{N}_+$. $\delta_{i,j}$ is difference between the best schedule of two tasks and the sum of the distances between the pickup and delivery locations of the two tasks. The small $\delta_{i,j}$ indicates that τ_i and τ_j being executed by the same agent has less impact on each other's completed time, which can efficiently optimize ST.

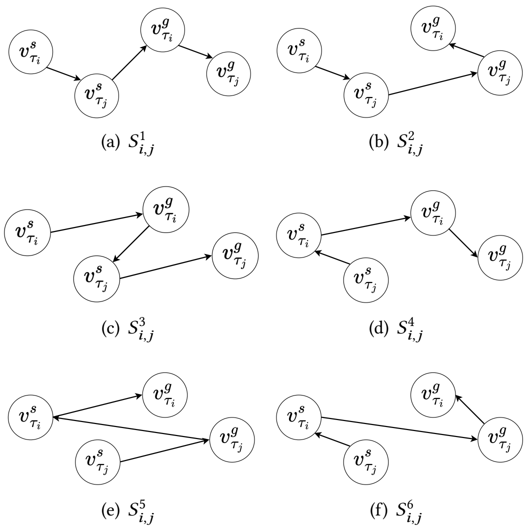


Figure 2: All possible schedules of two tasks

Howerver, in this paper, we focus on multi-load agents, which can carry multiple tasks simultaneously, not just two tasks. Therefore, we need to consider the carpooling scores between multiple tasks. It is worth noting that we cannot find the best schedule to complete multiple tasks, which is a NP-hard problem. Therefore, we propose a novel method to schedule there tasks. Then, we evaluate the upper bound on the cost of the schedule of multiple tasks. Finally, we quantify the carpooling scores between multiple tasks.

We first introduce the novel schedule method. As shown in Figure 3(a), there is a set of task Γ_K, which contains K tasks, τ_1, τ_2, ..., τ_K. The schedule S'_{Γ_K} of K tasks is shown in Figure 3(b). This schedule starts at the pickup location of a task in Γ_K, then goes to the pickup location of other tasks in Γ_K, and finally to the delivery location of all tasks in Γ_K. More significantly, in S'_{Γ_K}, the path between two pickup or delivery locations about two tasks in Γ_K is is part of paths of the best schedules of two tasks. For example, if path $v^s_{\tau_i}$ and $v^s_{\tau_j}$ is part of the best schedule of two tasks, then the path between $v^s_{\tau_i}$ and $v^s_{\tau_j}$ is part of the path of S'_{Γ_K}. Therefore, Theorem 4.3 is proposed to find the upper bound on cost of the schedule of S'_{Γ_K}, and prove it.

Theorem 4.3. *Given a set of tasks Γ_K, there exists a schedule S'_{Γ_K} shown in Figure 3(b) such that the following bound holds.*

$$Cost(S'_{\Gamma_K}) \leq 2 * \sum_{\tau_i \in \Gamma_K} dis(v^s_{\tau_i}, v^g_{\tau_i}) + 2*(K-1)\delta \qquad (2)$$

where δ is the carpooling scores between two tasks.

Proof. Firstly, we need to prove that there is a path to complete Γ_K. As shown in Figure 3(b), the schedule first connects the pickup locations of all tasks in Γ_K, then connects the delivery locations of all tasks in Γ_K. Focusing on the pickup locations set of Γ_K and connecting the pickup locations, we can get a tournament [12], which is a directed graph with exactly one edge between each

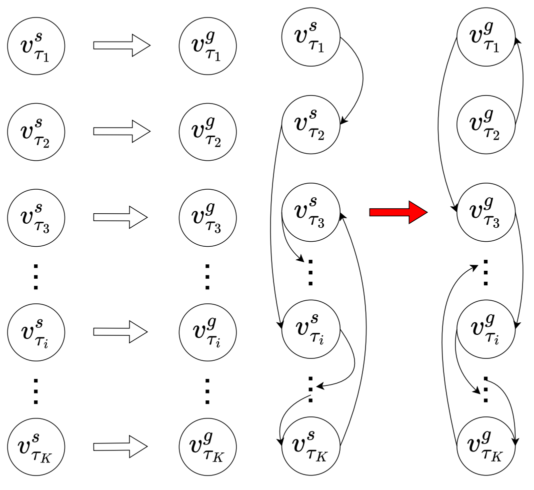


Figure 3: Schedule of K Tasks

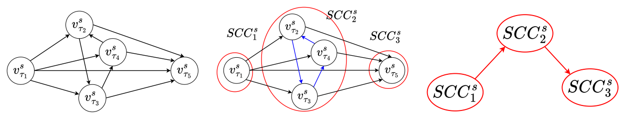


Figure 4: The tournament and SCC

two vertices, in one of the two possible directions. As shown in Figure 4(a), there are 5 pickup locations, and a path between each two locations. Accroding to the definition of tournament, there is a Hamiltonian path in the tournament [12], that means there exists a path to connect all pickup locations of Γ_K. Similarly, there is a path to connect all delivery locations of Γ_K. Then, we need to find a path to connect the pickup locations and delivery locations. Therefore, we introduce the Strongly Connected Component (SCC) [11], which is a subgraph that is strongly connected. As shown in Figure 4(b), $v^s_{\tau_2}, v^s_{\tau_3}$ and $v^s_{\tau_4}$ combine to form a SCC where there is a path between each two locations. When we shrink the SCC to a single vertex and focus on the last SCC of the pickup locations and first SCC of the delivery locations, we can always find two locations, $v^s_{\tau_i}$ and $v^g_{\tau_j}$, in two SCCs respectively, which connects the pickup locations and delivery locations. Moreover, $dis(v^g_{\tau_j}, v^s_{\tau_i})$ is less than or equal to $dis(v^g_{\tau_j}, v^s_{\tau_j}) + dis(v^g_{\tau_i}, v^s_{\tau_i})$. (The proof is shown in appendix).

Secondly, we need to evaluate the cost of the path. In the beginning, we evaluate the cost of the path between the pickup locations. $dis(v^s_{\tau_j}, v^s_{\tau_i})$ is the distance between the pickup locations of τ_i and τ_j, and all possible schedules of two tasks are shown in Figure 2. If the best schedule of two tasks is $S^1_{i,j}$, we can get Equation 3.

$$\begin{aligned} Cost(S^1_{i,j}) &= dis(v^s_{\tau_j}, v^s_{\tau_i}) + dis(v^g_{\tau_i}, v^s_{\tau_j}) + dis(v^g_{\tau_j}, v^g_{\tau_i}) \\ &\geq dis(v^s_{\tau_j}, v^s_{\tau_i}) + dis(v^g_{\tau_j}, v^s_{\tau_j}) \quad (3) \\ dis(v^g_{\tau_i}, v^s_{\tau_i}) + \delta_{\tau_i,\tau_j} &\geq dis(v^s_{\tau_j}, v^s_{\tau_i}) \quad (4) \end{aligned}$$

Then, based on this Definition 4.2, we can get Equation 4. Finally, we check all the possible schedules of two tasks in Figure 2, and we ensure the Equation 4 holds. Similarly, we can evaluate the cost of the path between the delivery locations by the same method and get Equation 5. (The proof is shown in appendix).

$$dis(v^g_{\tau_j}, v^s_{\tau_j}) + \delta_{i,j} \geq dis(v^g_{\tau_j}, v^g_{\tau_i}) \quad (5)$$

Therefore, we can evaluate the cost of the path between the pickup locations and delivery locations. The cost of paths in pickup locations less than or equal to $\sum_{\tau_i \in \Gamma_K \setminus \tau_i} dis(v^s_{\tau_i}, v^g_{\tau_i}) + (K-1)\delta$, which τ_i is the last pickup location in S'_{Γ_K}. The cost of paths in delivery locations less than or equal to $\sum_{\tau_i \in \Gamma_K \setminus \tau_j} dis(v^s_{\tau_i}, v^g_{\tau_i}) + (K-1)\delta$, which τ_j is the first delivery location in S'_{Γ_K}. The cost of the path $dis(v^g_{\tau_j}, v^s_{\tau_i})$, which connects the pickup locations and delivery locations in S'_{Γ_K}, is less than or equal to $dis(v^g_{\tau_j}, v^s_{\tau_j}) + dis(v^g_{\tau_i}, v^s_{\tau_i})$. Therefore, the cost of the schedule is less than or equal to $2{*}\sum_{\tau_i \in \Gamma_K} dis(v^s_{\tau_i}, v^g_{\tau_i}) + 2{*}(K-1)\delta$. □

Next, we can introduce the Get Carpooling Scores Algorithm (GCS) to quantify the carpooling scores between multiple tasks Γ_K as follows. Accroding to the Theorem 4.3, to be more efficient, we fisrt use Tarjan algorithm [8] to get the SCCs of the set of tasks, and topologize it to get chain of SCC like Figure 4(c). Then, in order to be more precise about δ, we calculate the carpooling scores in each SCC by Equation 6, and between two SCCs by Equation 7. Finally, we get the carpooling scores between two task groups by Algorithm 1. The result of this algorithm is the part of Theorem 4.3, $2 * (K-1)\delta$. the smaller the result leads to the lower the upper bound on cost of schedule, and the less the impact between tasks.

$$\Delta_{SCC} = \frac{2{*}\sum_{\tau_i \in SCC}\sum_{\tau_j \in SCC}\delta_{\tau_i,\tau_j}}{|SCC|} \quad (6)$$

$$\Delta_{SCC_i,SCC_{i+1}} = \frac{\sum_{\tau_i \in SCC_i}\sum_{\tau_j \in SCC_{i+1}}\delta_{\tau_i,\tau_j}}{|SCC^i|{*}|SCC^{i+1}|} \quad (7)$$

where SCC is the Strongly Connected Component, $|SCC|$ is the number of tasks in SCC. Equation 6 is average δ in SCC, which contains $|SCC|-1$ carpooling scores between tasks. Equation 7 is average δ between SCC_i and SCC_{i+1}, which contains one carpooling score between two tasks in SCC_i and SCC_{i+1}.

Algorithm Detail: The pseudo-code of Get Carpooling Scores Algorithm is shown in Algorithm 1, where the input is a set of tasks Γ_K containing K tasks, and the output is the carpooling scores Δ_{Γ_K}. First, we initialize Δ_{Γ_K} to 0. Then, we call Tarjan algorithm [8] to get the SCCs of Γ_K (lines 1-2). For each SCC, we calculate the carpooling scores in the SCC and between two SCCs (lines 3-10). Finally, we return the carpooling scores Δ_{Γ_K} (line 11).

Analysis: The time complexity of GCS algorithm is $O(K^2)$, where K is the number of tasks in Γ_K. The Tarjan algorithm [8] has a time complexity of $O(|V| + |E|)$, where $|V|$ is the number of

Algorithm 1 Get Carpooling Scores Algorithm

```
Input: Set of tasks $\Gamma_K$
Output: Carpooling scores $\Delta_{\Gamma_K}$
1: Initialize $\Delta_{\Gamma_K} = 0$
2: Call Tarjan algorithm to get $SCC^s$ and $SCC^g$ of $\Gamma_K$
3: for $SCC^s_i \in SCC^s$ do
4:     Get $\Delta_{SCC^s_i}$ by Equation 6
5:     if $SCC^s_i$ is not last SCC then
6:         Get $\Delta_{SCC^s_i, SCC^s_{i+1}}$ by Equation 7
7:     end if
8:     $\Delta_{\Gamma_K} += \Delta_{SCC} + \Delta_{SCC^s_i, SCC^s_{i+1}}$
9: end for
10: Do the same for $SCC^g$
11: return $\Delta_{\Gamma_K}$
```

vertices and $|E|$ is the number of edges. Therefore, $O(K^2)$ is the time complexity of call Tarjan algorithm to get the SCCs of Γ_K (line 2). The time complexity of calculating the carpooling scores in each SCC and between SCCs is $O(K)$ (lines 3-10). Therefore, the time complexity of GCS algorithm is $O(K^2)$.

4.2 K-Capacity Hierarchical Clustering

Based on Theorem 4.3, we can quantify the carpooling scores between multiple tasks. However, the schedule of multiple tasks needs to go to the pickup locations of all tasks first. In MLAPD problem, the maximum load capacity of the agent limits the size of the set of tasks that we can quantify the carpooling scores. Therefore, the K-Capacity Hierarchical Clustering Algorithm (KCHC) is proposed to divide the set of tasks into groups by carpooling scores, which contain at most K tasks. K is the maximum load capacity of the agent. The KCHC algorithm is shown in Algorithm 2.

Algorithm 2 K-Capacity Hierarchical Clustering Algorithm

```
Input: Set of tasks $\Gamma$, the max capacity of agents $K$
Output: Set of task groups $\Pi$
1: Initialize $\Pi = \emptyset$
2: for $\tau_i \in \Gamma$ do
3:     Treat $\tau_i$ as a task group $\Gamma_i$, and add $\Gamma_i$ into $\Pi$
4:     for $\Gamma_j \in \Pi$ and $\tau_j \in \Gamma_j$ do
5:         $\Delta_{\Gamma_i \cup \Gamma_j} = \delta_{\tau_i, \tau_j}$
6:     end for
7: end for
8: while $\Pi$ has changed do
9:     Select minimum carpooling scores $\Delta_{\Gamma_p \cup \Gamma_q}$ from $\Pi$
10:    Merge $\Gamma_p$ and $\Gamma_q$ into $\Gamma_{p+q}$
11:    Remove $\Gamma_p$ and $\Gamma_q$ from $\Pi$ and add $\Gamma_{p+q}$ into $\Pi$
12:    for $\Gamma_j \in \Pi$ do
13:        if $|\Gamma_j| + |\Gamma_{p+q}| \leq K$ then
14:            Get $\Delta_{\Gamma_{p+q} \cup \Gamma_j}$ by Algorithm 1
15:        end if
16:    end for
17: end while
18: return $\Pi$
```

Algorithm Detail: The pseduo-code of K-Capacity Hierarchical Clustering Algorithm is shown in Algorithm 2, where the input is a set of tasks Γ and the max capacity of agents K, and the optput is a set of task groups Π. First, we initialize Π to an empty set (line 1). Then, we treat each task as a task group and add it to Π, and calculate the carpooling scores between each pair of task groups (lines 2-7). Next, we iteratively merge the task groups with the minimum carpooling scores. If the size of the merged task group is less than or equal to K, we merge the task groups and calculate the carpooling scores between them, and calculate the carpooling scores between the merged task group and other task groups (lines 8-18). Finally, we return the set of task groups Π (line 19).

Analysis: If the number of tasks in Γ is N, the time complexity of KCHC algorithm is $O(N^2)$. The time complexity of calculating the carpooling scores between each pair of task groups is $O(N^2)$ (lines 2-7). The time complexity of merging the task groups is $O(N^2)$ (lines 8-18). In each iteration, we merge two task groups, calculate the carpooling scores between the merged task group and other task groups, and the number of task groups is reduced by 1, so we need to iterate $(N - N/K)$ times at most and calculate the carpooling scores with N task groups at most. Therefore, the time complexity of KCHC algorithm is $O(N^2)$.

4.3 Task Group Allocation Algorithm

Before we introduce the Task Group Allocation Algorithm, we first define the detour cost of the schedule of agent a to complete Γ, which is the difference between the cost of the schedule and the sum of the distances between the pickup and delivery locations of the tasks in Γ. The detour cost is used to evaluate the rationality of the schedule of agent a to complete Γ.

$$DCost(S_{\Gamma_a}) = Cost(S_{\Gamma_a}) - \sum_{\tau_i \in \Gamma_a} dis(v^s_{\tau_i}, v^g_{\tau_i}) \tag{8}$$

where S_{Γ_a} is the schedule of agent a to complete Γ. Therefore, we introduce the Get Schedule of Task Group Algorithm (GSTG) to the schedule of task group by Equation 8.

Next, we introduce Get Schedule of Task Group (GSTG) Algorithm to get the schedule of task group by the detour cost. The GSTG algorithm is shown in Algorithm 3.

Algorithm 3 Get Schedule of Task Group

```
Input: Set of tasks $\Gamma$, Agent $a$
Output: The detour cost of schedule $DCost(S_\Gamma)$
1: Initialize $Cost(S_\Gamma) = 0$
2: while $\Gamma$ is not empty do
3:     for $\tau_i \in \Gamma$, adjacent locations $v_i$ and $v_j$ in $S_{\Gamma_a}$ do
4:         Insert $v^s_{\tau_i}$, $v^g_{\tau_i}$ between $v_i$ and $v_j$, respectively
5:         Calculate the detour cost $DCost(S_{\Gamma_a})$.
6:     end for
7:     Select task $\tau$ with minimum detour cost
8:     Remove $\tau$ from $\Gamma$
9: end while
10: return $DCost(S_\Gamma)$
```

Algorithm Detail: The pseduo-code of Get Schedule of Task Group Algorithm is shown in Algorithm 3, where the input is a set of tasks Γ and an agent a, and the output is the detour cost of the schedule $DCost(S_\Gamma)$. First, we initialize the cost of the schedule $Cost(S_\Gamma)$ to 0 (line 1). Then, we iteratively insert the pickup and delivery locations of tasks in Γ into the schedule S_{Γ_a}, and calculate the detour cost of the schedule $DCost(S_{\Gamma_a})$ (lines 2-6). Select the task with the minimum detour cost and remove it from Γ, until Γ is empty (lines 7-9). Finally, we return the detour cost of the schedule $DCost(S_\Gamma)$ (line 10).

Then, we allocate task groups to agents. However, we cannot indirectly allocate task groups to agents, since we do not consider the location of agents, while dividing tasks into groups. It maybe lead to the situation that agents schedule task groups inappropriately.

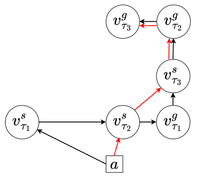


Figure 5: The schedule of agent a to complete Γ

Example 4.4. As shown in Figure 5, there is an agent, a, and a task group Γ, which contains three tasks, τ_1, τ_2 and τ_3. The black lines are the best schedule S_Γ of a to complete Γ. Obviously, in black lines, a requires a detour to pickup location of τ_1, which is irrational schedule.

Therefore, to consider the location of agents, we split the task group Γ into multiple subgroups,

$$\Pi_\Gamma = \{\forall \Gamma_i | \Gamma_i \subset \Gamma\} \tag{9}$$

where Π_Γ is the set of subgroups of Γ.

Based on there, we propose the Task Group Allocation Algorithm to allocate task groups to agents.

Algorithm Detail: The pseudo-code of Task Group Allocation Algorithm is shown in Algorithm 4, where the input is a set of tasks Γ and a set of agents A, and the output is the schedules of all agents. First, we initialize the maximum load capacity of agents K, and then divide the set of tasks Γ into task groups by the K-Capacity Hierarchical Clustering Algorithm (lines 1-2). For each task group Γ, we split it into subgroups by Equation 9 and extend the set of subgroups Π_Γ (lines 3-6). Then, we calculate the detour cost of the schedule of each agent to complete each task group by the GSTG Algorithm (lines 7-9). Next, we iteratively select the agent and task group with the minimum detour cost, remove the task group and the task groups that intersect with it from $\tilde{\Pi}$, and update the schedule of the agent to complete the task group (lines 11-13). Moreover, we need to update the detour cost of the schedule of the agent to complete the remaining task groups (lines 14-16). Finally, we return the schedules of all agents (line 18).

Analysis: If the number of tasks in Γ is N and the number of agents is M, the time complexity of TGA algorithm is $O(N^2)$. The time complexity of dividing the set of tasks Γ into task groups by the K-Capacity Hierarchical Clustering Algorithm is $O(N^2)$ (lines 1-2). Then, we split each task group into subgroups and extend the set of subgroups Π_Γ, which contains at most $K * N$ task groups (lines 3-6). Due to K is a constant, the time complexity of calculating the detour cost of the schedule of each agent to complete each task group is $O(N*M)$ (lines 7-9). Next, $O(N)$ is the time complexity of selecting the agent and task group with the minimum detour cost, removing the task group and the task groups that intersect with it from $\tilde{\Pi}$, and updating the schedule of the agent to complete the task group (lines 11-13). Moreover, the time complexity of updating the detour cost of the schedule of the agent to complete the remaining task groups is $O(N * N)$ (lines 14-16). Therefore, the time complexity of TGA algorithm is $O(N^2)$.

Algorithm 4 Task Group Allocation Algorithm

```
Input: Set of tasks $\Gamma$, set of agents $A$
Output: Schedules of all agents
 1: $K$ = the max capacity of agents
 2: Get task groups $\Pi$ by Algorithm 2
 3: for $\Gamma \in \Pi$ do
 4:     Get $\Pi_\Gamma$ by Equation 9
 5:     $\tilde{\Pi}$ extends $\Pi_\Gamma$
 6: end for
 7: for $a_i \in A$, $\Gamma_j \in \tilde{\Pi}$ do
 8:     Get detour cost $DCost(S_{\Gamma_{a_i}})$ by Algorithm 3
 9: end for
10: while $\tilde{\Pi}$ is not empty do
11:     Select agent $a$ and task group $\Gamma$ with minimum detour cost
12:     Remove $\Gamma$ and $\{\forall \Gamma' | \Gamma' \cap \Gamma \neq \phi\}$ from $\tilde{\Pi}$
13:     Update the schedule of $a$ to complete $\Gamma$
14:     for $\Gamma' \in \tilde{\Pi}$ do
15:         Update the detour cost of schedule of $a$ to complete $\Gamma'$
16:     end for
17: end while
18: return Schedules of all agents
```

5 EXPERIMENTAL ANALYSIS

5.1 Experimental Settings

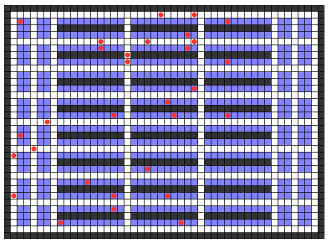


Figure 6: The map

In this section, we first introduce the experimental settings, which include the map, the evaluation metrics, and the compared algorithms. Then, we present the experimental results and analyze the performance of the proposed algorithm.

We conduct experiments in a 33×46 grid map [4, 5], which is shown in Figure 6. In Figure 6, the black grid is an obstacle, the

Table 1: Experimental Settings

Parameter	Value
Number of tasks	2k,4k,**6k**,8k,10k
Number of agents	30,40,**50**,60,70
Max capacity of agents	2,**3**,4,5

blue grid is the pickup location or delivery location of a task, the white grid is the road, and the red circle presents an agent.

The experimental settings are shown in Table 1. We vary the number of tasks from 2k to 10k, the number of agents from 30 to 70, and the max capacity of agents from 2 to 6. The bold values are the default settings. The evaluation metrics include service time (*ST*), makespan (*MS*), and processing time (*PT*). Finally, we compare the proposed algorithm with the following algorithms:

- nCAR algorithm [6], which clusters tasks based on the nearest neighbor, and constructs a route for each cluster by mapping it to a TSP.
- IIG algorithm [17], which generates an initial solution by a greedy strategy, and iteratively improves the solution by a local search strategy.
- Multi-angle K-means algorithm [16], which uses the multi-angle K-means algorithm to divide the set of tasks into groups, and uses GA algorithm to perform path planning.
- TGA algorithm, which uses KCHC Algorithm to divide the set of tasks into groups, and allocates task groups to agents.

5.2 Experimental Results

Performance of task number $|\Gamma|$: As shown in Figure 7(a), 7(b) and 7(c), we can see the impact of the number of tasks on *ST*, *MS* and *PT*. When the number of tasks increases, the *ST*, *MS* and *PT* increase, since the agents need more time to complete more tasks, in each batch. The proposed TGA algorithm outperforms the other algorithms on *ST* and *MS*, since it can allocate task groups to agents more efficiently. As shown in Figure 7(a), when the number of tasks is 6k, TGA algorithm reduces *ST* by 20.6% compared with nCAR algorithm, and by 15.2% compared with IIG algorithm.

Performance of agent number $|A|$: figure 7(d), 7(e) and 7(f) illustrate the impact of the number of agents on *ST*, *MS* and *PT*. When the number of agents increases, the *ST* is reduced, and the trend is gradually decreasing. Because, starting with a small number of agents, the waiting time for the task is extended, but as the number of agents increases and approaches full saturation, the decreasing trend of ST slowly reduces. In Figure 7(e), *MS* has a similar trend to *ST*, even slightly higher. Although TGA algorithm has a higher *PT* than other algorithms, it has a lower *ST* and *MS*.

Performance of capacity c: The impact of the max capacity of agents on *ST*, *MS* and *PT* is shown in Figure 7(g), 7(h) and 7(i). A higher capacity of agents can lead to a lower *ST* and *MS*. As the capacity of agents increases, the *ST* and *MS* decrease. The increase in the capacity of agents means that the agents can carry more tasks simultaneously, which leads that all algorithms have better performance on *ST* and *MS*. Sepcially, all algorithms have a similar performance on *PT*, since the capacity of agents does not affect the processing time of tasks. Howerver, *PT* of the TGA algorithm has obviously increased by capacity, since the more capacity, the more complexity of the task group allocation.

6 CONCLUSION

In this paper, we propose the Task Group Allocation Algorithm to solve the MLAPD problem. We first evaluate the upper bound on the cost of the schedule of multiple tasks, and quantify the carpooling scores between multiple tasks. Next, we propose the K-Capacity Hierarchical Clustering Algorithm to divide the set of tasks into groups, in which tasks have less impact on each other's completion time. Then, we introduce the Task Group Allocation Algorithm to allocate task groups to agents. Finally, we conduct experiments to evaluate the performance of the proposed algorithm. In the future, we will consider the heterogeneous agents and the dynamic environment in the MLAPD problem, which is more practical in real-world applications.

REFERENCES

[1] Xiaoshan Bai, Andres Fielbaum, Maximilian Kronmüller, Luzia Knoedler, and Javier Alonso-Mora. 2022. Group-based distributed auction algorithms for multi-robot task assignment. *IEEE Transactions on Automation Science and Engineering* 20, 2 (2022), 1292–1303.

[2] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. 2021. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5816–5823.

[3] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11272–11281.

[4] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7651–7658.

[5] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868* (2017).

[6] Chayan Sarkar, Himadri Sekhar Paul, and Arindam Pal. 2018. A Scalable Multi-Robot Task Allocation Algorithm. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Brisbane, Australia, 5022–5027. https://doi.org/10.1109/ICRA.2018.8460886

[7] Dingyuan Shi, Yongxin Tong, Zimu Zhou, Ke Xu, Wenzhe Tan, and Hongbo Li. 2022. Adaptive task planning for large-scale robotized warehouses. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 3327–3339.

[8] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1, 2 (1972), 146–160.

[9] Hanfu Wang and Weidong Chen. 2024. Task scheduling for heterogeneous agents pickup and delivery using recurrent open shop scheduling models. *Robotics and Autonomous Systems* 172 (2024), 104604.

[10] Shengli Wang, Youjiang Liu, Yongtao Qiu, Simin Li, and Jie Zhou. 2023. An efficient distributed task allocation method for maximizing task allocations of multirobot systems. *IEEE Transactions on Automation Science and Engineering* (2023).

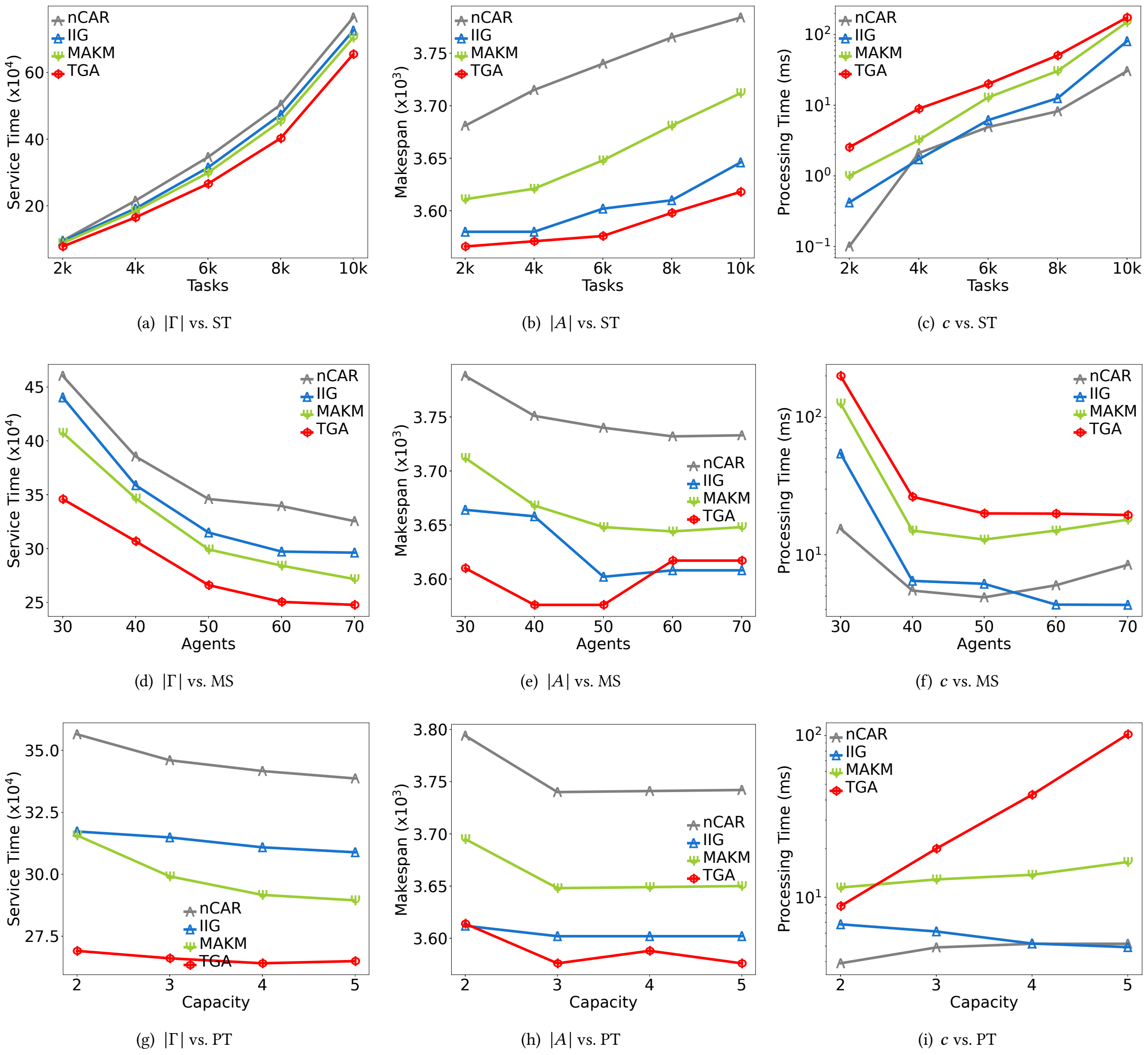
[11] Wikipedia contributors. 2024. Strongly connected component — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Strongly_connected_component&oldid=1237069718. [Online; accessed 16-September-2024].

[12] Wikipedia contributors. 2024. Tournament (graph theory) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Tournament_(graph_theory)&oldid=1234378036. [Online; accessed 15-September-2024].

[13] Nils Wilde and Javier Alonso-Mora. 2024. Statistically distinct plans for multi-objective task assignment. *IEEE Transactions on Robotics* (2024).

[14] Xiaohu Wu, Yihao Liu, Xueyan Tang, Wentong Cai, Funing Bai, Gilbert Khonstantine, and Guopeng Zhao. 2021. Multi-agent pickup and delivery with task deadlines. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. 360–367.

[15] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29, 1 (2008), 9–9.



(a) $|\Gamma|$ vs. ST (b) $|A|$ vs. ST (c) c vs. ST

(d) $|\Gamma|$ vs. MS (e) $|A|$ vs. MS (f) c vs. MS

(g) $|\Gamma|$ vs. PT (h) $|A|$ vs. PT (i) c vs. PT

Figure 7: Performance of $|\Gamma|$ vs. ST, MS and PT

[16] Jiangwei You, Jianfang Jia, Xiaoqiong Pang, Jie Wen, Yuanhao Shi, and Jianchao Zeng. 2023. A Novel Multi-Robot Task Assignment Scheme Based on a Multi-Angle K-Means Clustering Algorithm and a Two-Stage Load-Balancing Strategy. *Electronics* 12, 18 (2023). https://doi.org/10.3390/electronics12183842

[17] Xu-jin Zhang, Hong-yan Sang, Jun-qing Li, Yu-yan Han, and Peng Duan. 2022. An effective multi-AGVs dispatching method applied to matrix manufacturing workshop. *Computers & Industrial Engineering* 163 (2022), 107791.

[18] Xing Zhou, Huaimin Wang, Bo Ding, Tianjiang Hu, and Suning Shang. 2019. Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications* 116 (2019), 10–20.