

# Scheduling and Negotiation Method for Double Synchronized Multi-Agent Pickup and Delivery Problem

Yuki Miyashita<sup>1</sup> <sup>a</sup> and Toshiharu Sugawara<sup>2</sup> <sup>b</sup>

<sup>1</sup>Shimizu Corporation, 2-16-1 Kyobashi, Chuo-ku, Tokyo, Japan

<sup>2</sup>Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan

**Keywords:** Multi-Agent System, Scheduling, Synchronization, Sequential Cooperative Task, Contract Net Protocol.

**Abstract:** We propose a multi-agent scheduling and negotiation method for pickup and delivery tasks, each of which requires two synchronizations between heterogeneous agents. Real-world applications of multi-agent systems often require synchronous cooperation at specific times while resolving conflicts between agents. Iterative multi-agent path-finding problem has recently received much attention, which is called *multi-agent pickup and delivery* (MAPD) problem. In the MAPD problem, agents move to the pickup locations, load materials, and deliver them to their respective unloading locations, by repeatedly assigning new tasks to agents consecutively. Our target is a *multi-agent pickup and delivery* (MAPD) problem in a multi-story building/warehouse, and thus, a carrier agent requires synchronizations when loading and unloading materials in elevators. We call this problem a *double synchronized MAPD* (DSMAPD). To our knowledge, the current studies in MAPD have rarely considered such complicated tasks with synchronizations. Our proposed method attempts to reduce the unnecessary free time to improve the efficiency and agents' operating time without causing collisions and disturbing already agreed-upon synchronizations. The experiments show that our method can outperform naive methods for completing DSMAPD problem instances with reasonable planning and scheduling time.


## 1 INTRODUCTION


Many real-world applications of multi-agent systems require that agents cooperate with other agents. The examples of these applications include cooperative routing of a truck-drone system (Luo et al., 2022; Das et al., 2021), manipulation of elevators for robots in multiple floors (Ge et al., 2021), and logistic processes in open-pit mines (Ahumada and Herzog, 2021; Zhang et al., 2021). Our target applications are transportation and packing of products in a multi-story warehouse and transportation of heavy construction materials to work locations for next day's task during night by self-driving carriers and elevators. In these applications, a robot (i.e., a carrier agent) transports a material on a certain floor horizontally and an elevator (an elevator agent) transports it vertically. When the material should be delivered to another floor level, the carrier agent carries it to an elevator and unloads it. Then, the elevator moves to the destination floor, and another carrier agent on that

floor loads the material inside the elevator and carries it to the required destination. This type of problem should be executed by iteratively generating schedules with two synchronizations between carrier and elevator agents on different floors for the respective tasks.

Therefore, these tasks can be considered as a *double synchronized multi-agent pickup-and-delivery* (DSMAPD) problem, which is an extension of the *multi-agent pickup-and-delivery problem* (MAPD) (Ma et al., 2017). The objective of DSMAPD is that two types of cooperative agents transport materials without collisions as efficiently as possible.

Many studies have been conducted on the MAPD problem so far, but most of them have focused on generating collision-free paths to consecutively perform the pickup and delivery in a single floor, that is, horizontal transportation (Sharon et al., 2015; Ma et al., 2017; Yamauchi et al., 2022; Miyashita et al., 2023; Okumura et al., 2022; Okumura, 2023). Although in real-world applications agents are often required to cooperate performing synchronized actions between independent and heterogeneous agents,

<sup>a</sup>  <https://orcid.org/0000-0002-1676-9346>

<sup>b</sup>  <https://orcid.org/0000-0002-9271-4507>

only few studies have focused on synchronized tasks in MAPD. An exceptional study was conducted by Zhang et al. (Zhang et al., 2022), in which they proposed a mining truck system that realizes a consecutive transportation of materials excavated by shovels and hauled by trucks at some loading locations. However, their method is excessively simple for our target applications, which require double synchronization between different pairs of heterogeneous agents, simultaneously.

Therefore, we propose a scheduling and negotiation method for the DSMAPD problem that enables agents to generate an efficient schedule through negotiation to select appropriate cooperative agents for the synchronized work. This method is an extension of the *contract net protocol* (CNP) (Smith, 1980), and a carrier agent on the start floor of a DSMAPD task instance plays the role of the manager. Thus, it initiates the negotiation to determine the elevator agent and the time and location of the first synchronization. Then, after receiving the arrival time and location on the destination floor, the manager agent further continues the negotiation with all carrier agents on that floor to determine the time and location of the second synchronization with the selected elevator.

One feature of the proposed method is that, because the schedules of an agent will likely involve free time owing to the synchronization with other agents and collision avoidance, new schedules are generated so that unnecessary free (i.e., wasting) times are reduced as much as possible without disrupting the synchronizations that have already been determined. In naive methods, a new schedule is usually added at the end of the schedule time to avoid disruption. In our method, agents attempt to insert a new schedule into the free time in the schedule timeline, while creating the shortest collision-free paths using a certain multi-agent path-finding algorithm such as cooperative A\* (CoopA\*) (Silver, 2005) to estimate the synchronization time.

We evaluated our method in simulated experimental environments by comparing the results with those obtained from the straightforward breadth-first-based search to decide the cooperative agents and synchronization time. The results indicate that our method achieves considerably efficient and effective executions of DSMAPD problem instances without collisions and deadlock/livelock situations. Finally, we state the limitations of our method and propose future extensions.

## 2 RELATED WORK

The combination of robot (vehicle) routing and scheduling is used in many applications, and thus, extensive studies have targeted this topic from these applications (Fumero and Vercellis, 1999; Bredström and Rönnqvist, 2008; Ma et al., 2017). For example, Ma et al. (Ma et al., 2017) proposed a prioritized path planning method for the MAPD problem. Kumar et al. (Kumar et al., 2023) introduced a mechanism to provide better flexibility for multiple selfish agents while improving the total performance in a taxi fleet problem. However, Ma et al. (Ma et al., 2017) focused only on generating collision-free paths for MAPD, and Kumar et al. (Kumar et al., 2023) targeted a strategic and operational decision problem in which agents maximize their profits (*profit-maximization*). The problem of synchronized scheduling for heterogeneous agents has not received much attention in the literature.

The synchronized scheduling of heterogeneous agents has also been extensively studied (Zhang et al., 2022; Kafle et al., 2017). Kafle et al. (Kafle et al., 2017) proposed an urban parcel delivery system in which the trucks respectively determine their routes and schedules to coordinate with the cyclists and pedestrians. Das et al. (Das et al., 2021) presented a new mechanism that includes multiple vehicles in cooperation with multiple drones that work synchronously with trucks as mobile launching and retrieval sites for drones. In the process of open-pit mines (Bastos et al., 2011; Alexandre et al., 2017), Ahumada et al. (Ahumada and Herzog, 2021) proposed multi-agent negotiation algorithms using a contract net protocol in which agents interact with each other to generate schedules for their represented equipment item. Alexandre et al. (Alexandre et al., 2017) presented a mathematical model for the multi-objective (cost and production) truck dispatch problem in open-pit mining operations. However, their method is too simple for our target applications, which require two synchronizations between carrier and elevator agents on different floors for their respective tasks at a construction site. Furthermore, some types of agents can be a bottleneck of work due to their small number. Therefore, we propose a scheduling and negotiation method for the DSMAPD problem in which agents can generate synchronized schedules for cooperative work without disrupting the synchronizations that have already been determined as well as reducing the unnecessary wasting period of time.

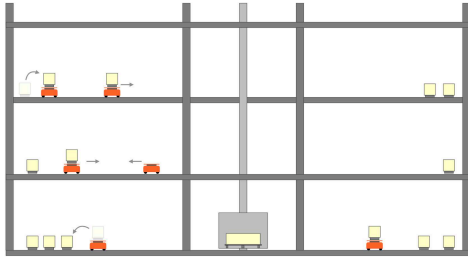


Figure 1: Example of multi-story building environment.

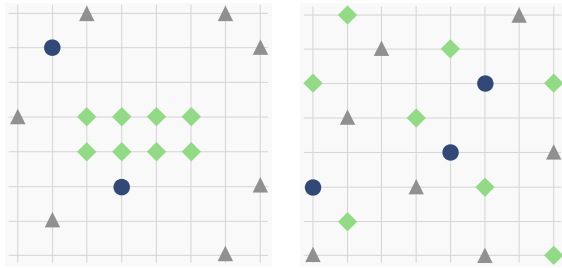


Figure 2: Example Environments for DSMAPD problems.

### 3 MODEL AND PROBLEM

#### 3.1 Model of Agents and Environment

We formulate the DSMAPD problem on the basis of our practical logistics scenario. The main difference from MAPD is that beside the delivery tasks in a floor, tasks in DSMAPD requires synchronized cooperation between heterogeneous agents. Example in a multi-story environment is shown in Fig. 1, in which two types of agents in an environment execute tasks of delivery orders performed through sequential cooperation. The first type of agent, called a *carrier agent*, has the role of loading a material, delivering it horizontally to the given destination in a specified floor, like the agent in MAPD. The second type of agent, called an *elevator agent*, transports a material vertically between floor levels. We assume that all agents can transport only one material and an elevator agents does not transport a carrier agent to another floor level.

A task in DSMAPD requires carrying material from a floor (called the *pickup floor*) to the floor of the destination point (the floor is called the *delivery floor*, hereafter). If the pickup and delivery floors are the same, a carrier agent execute it by loading a material, carrying it, and unloading it in that floor. In contrast, if the levels are different, cooperation among different agents is necessary, that is, one carrier agent on the pickup floor must carry the material to a *meeting*

*point* that can be reached by an elevator agent, the elevator must transport it to the delivery floor level, and another carrier agent on the delivery floor must load the material from the elevator at the meeting point, carry it, and unload it at the destination point. Note that each elevator agent has its own meeting points on all floors where it can transport a material. Therefore, this type of task consists of three subtasks that should be executed sequentially with synchronization by cooperation of two type of agents. the overall objective of DSMAPD is to accomplish effective cooperative transportation with reasonable computational and communication cost and increase the throughput, i.e. the number of executed tasks per unit of time.

The floor environments in Fig. 2 are described by a grid consisting of  $N \times N$  points (of intersection), where a blue circle is a carrier agent, a green diamond is an elevator agent, and a gray triangle is a task endpoint, which is a pickup or delivery location. A carrier agent takes one of the actions  $A_c = \{N(orth), S(outh), E(ast), W(est), ST(ay), L(oad), UN(load)\}$ , whereas an elevator agent can move to other floor levels by taking actions  $A_e = \{U(p), D(own), ST(ay)\}$ . When a carrier agent loads/unloads a material into/from an elevator, it moves to the point at which the elevator is located. We do not consider the details of how to avoid conflicts, that is, collisions and deadlock situations, because some centralized/decentralized algorithms that avoid conflicts between agents have already been proposed (Sharon et al., 2015; Yamauchi et al., 2022; Ma et al., 2017; Miyashita et al., 2023), and we assume that one of them is used.

#### 3.2 Problem Formulation

We introduce the discrete time  $t \geq 0$ . Let  $I_c = \{1, \dots, N\}$  be the set of  $N$  carrier agents,  $I_e = \{e_1, \dots, N_E\}$  be the set of  $N_E$  elevator agents, and  $\mathcal{T} = \{\tau_1, \dots, \tau_M\}$  be the set of  $M$  tasks required in an environment, where  $N, N_E$ , and  $M$  are positive integers. We assume that all tasks are given in advance and/or some tasks are added to  $\mathcal{T}$  in real time. Environment  $\mathcal{G}$  consists of  $N_f$  floors and the  $f$ -th floor (where  $f \in F = \{1, \dots, N_f\}$ ) is expressed by graph  $\mathcal{G}_f = (V_f, E_f)$ . Note that  $\mathcal{G}_1$  does not necessarily indicate the ground floor of a multi-story building but rather the lowest floor. A location  $o_k$  in  $\mathcal{G}$  is represented by  $(f_k, v_k) \in F \times V_{f_k}$ , a pair of the floor level and the location in that floor. A path is denoted by a sequence of locations and time,  $p = ((o_1, t_1), \dots, (o_{n_p}, t_{n_p}))$ , where  $t_k < t_{k+1}$ , and  $o_i$  and  $o_{i+1}$  indicate a horizontal move along an edge in the same floor  $G_f$  (i.e.,  $f_i = f_{i+1}$  and  $(v_i, v_{i+1}) \in E_{f_i}$ ) or

a vertical move at the same horizontal location (i.e.,  $f_i = f_{i+1} + 1$  or  $f_{i+1} - 1$ , and  $v_i = v_{i+1}$ ). Therefore, the path for a carrier agent consists only of horizontal moves, whereas a path for an elevator consists of vertical moves.

Task  $\tau_k \in \mathcal{T}$  is specified by tuple  $\tau_k = (o_k^p, o_k^d, \mu_k)$ , where  $\mu_k$  is the material to carry, while  $o_k^p = (f_k^p, v_k^p)$  and  $o_k^d = (f_k^d, v_k^d)$  are the locations of the pickup and delivery at which an agent loads to pick  $\mu_k$  up and then it or another agent unloads to finally deliver  $\mu_k$ . When  $f_k^p = f_k^d$ ,  $\tau_k$  does not require vertical transportation and  $\tau_k$  should be executed by one carrier agent on floor  $f_k^p$  by moving along the horizontal path from the current location to the delivery location  $o_k^d$  via the loading (pickup) location  $o_k^p$ ; such a task is called a *horizontal task*. Otherwise,  $\tau_k$  requires cooperation including vertical transportation of the material from  $f_k^p$  to  $f_k^d$  by an elevator agent and the horizontal transportation by two carrier agents on  $f_k^p$  and  $f_k^d$ . Thus, this type of task is called a *vertical task*. To achieve cooperative work for DSMAPD, we assume that agents can communicate with each other by exchanging messages. The aim of our problem is that all agents repeatedly perform sequential cooperation to complete all the tasks as efficiently as possible.

A carrier agent can move to a neighboring location in one timestep and load/unload in  $T_{lu} (\geq 1)$  timesteps. An elevator agent can move to a next level from the current floor in  $T_f (\geq 1)$  timesteps. We assume that it takes more time for vertical movement and loading/unloading than time for moving next location.

### 3.3 Agents' Schedule and Issues

One solution to achieve tailored cooperation would be to set up and maintain the schedules of individual agents' activities so that they are consistent with those of other agents (Ahumada et al., 2020). In our problem, timely synchronization is required for efficient and effective work, i.e., an elevator agent should arrive at approximately the same time as a cooperating carrier agent arrives at the meeting point and vice versa. Furthermore, if the number of elevator agents is relatively small, it is desirable for the carrier agents to arrive at the meeting point slightly earlier. To achieve these activities, each agent has its own schedule list in order of time and must maintain the start and end times of the assigned tasks so that all agents can work coherently.

When agents maintain their schedule lists, a new schedule for an assigned task will usually be added to the end of schedule list, although this often causes inefficiency; otherwise, the other agreed-upon (so ap-

proved) schedules in the list will be affected, and the ensuing changes will further affect the other agents' schedules in a cascading fashion. For example, in DSMAPD, if a new schedule is put in the middle, the starting location for the subsequent task will be changed, and thus, the starting time for that task will also change. This may cause a loss of synchronization with other cooperative agents. Furthermore, to avoid collisions, the path to the loading location will change accordingly, which will affect the paths that are already approved for other agents. Therefore, we address this issue and propose a negotiation and scheduling method to allocate the next task to an agent for efficient cooperative task execution without disturbing synchronization.

## 4 PROPOSED METHOD

The objective of DSMAPD is to accomplish effective cooperative transportation with reasonable computational and communication cost and with effective synchronization. In this section, we describe the proposed scheduling and negotiation method in which agents interact with other agents to create efficient schedules for cooperation by incorporating advance path-planning for a new task.

### 4.1 Scheduling Method

#### 4.1.1 Schedule Element

Agent  $i$  takes actions according to the associated schedule list  $S_i = [s_1^i, \dots, s_{|S_i|}^i]$  whose element, called *schedule element* (or simply, *schedule*)  $s_n^i$  in  $S_i$  is specified by  $s_n^i = \{t_n^s, t_n^l, t_n^e, o_n^s, o_n^l, o_n^e, p_n^{s \rightarrow l}, p_n^{l \rightarrow e}, \mu_k, \tau_k\}$ , where  $\tau_k$  is the task to execute in this schedule element and  $\mu_k \in \tau_k$  denotes a material requested to carry in  $\tau_k$ . Parameter  $t_n^s$  is the time when  $i$  starts moving from the anticipated starting location,  $o_n^s$ , to the loading location,  $o_n^l$ , along the planned path  $p_n^{s \rightarrow l}$ .  $t_n^l (> t_n^s)$  is the time when  $i$  starts moving from  $o_n^l$  to its unloading location  $o_n^e$  along the path  $p_n^{l \rightarrow e}$  after loading, and  $t_n^e (> t_n^l)$  is the time when agent starts unloading  $\mu_k$  at  $o_n^e$  specified in  $\tau_k$ . We define the *length* of the schedule element as  $(T_{lu} + t_n^e) - t_n^s$ . We introduce the maximum time length,  $T_{max} > 0$ , of the schedule list to avoid scheduling in the distant future, i.e., agents do not generate a schedule element whose end time  $t_n^e > t_c + T_{max}$ , where  $t_c$  is the *current time*. Note that if  $i$  is the carrier agent,  $s_n^i$  must satisfy  $o_k^p = o_n^l$  or  $o_k^d = o_n^e$  (or both).

For the first element  $s_1^i$  of  $S_i$ ,  $o_1^s$  is the current location and  $p_1^{s \rightarrow l}$  is the path from  $o_1^s$  to  $o_1^l$ . Otherwise,

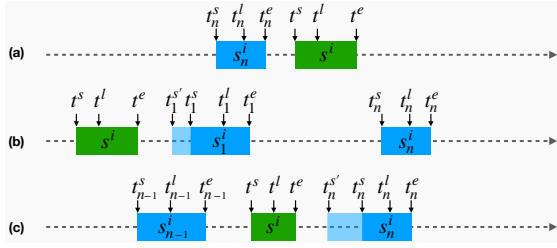


Figure 3: Cases for adding a schedule.

$p_n^{s \rightarrow l}$  is the path whose start location  $o_n^s$  is the destination of the previous schedule element,  $o_{n-1}^e$  (or a location where the agent  $i$  can stop temporarily by using a standby location (Yamauchi et al., 2022) near  $o_{n-1}^e$ ). All paths,  $p_n^{s \rightarrow l}$  and  $p_n^{l \rightarrow e}$ , must be collision-free and are generated by an appropriate algorithm such as *cooperative A\** (Silver, 2005) by referring to the approved (scheduled) paths of other agents. When all paths and synchronizations of the new schedule element have been approved after a negotiation process (Section 4.2), it is stored in the schedule list. Note that we assume that an elevator agent never collides with other elevator agents because of the characteristics of normal elevators. Any  $s_n^i \in S_i$  whose  $t_n^e$  is smaller than  $t_c$  is removed; this means that the first element of  $S_i$  is the schedule for the current or next activity for task execution. Moreover, when  $\tau_k$  is completed, it is removed from  $\mathcal{T}$ .

One main feature of our scheduling method is that we retain the times for synchronizations,  $t_n^l$  and  $t_n^e$  in  $s_n^i \in S_i$ , if another schedule element is added to the list. This avoids the cascade effects to the schedules of other agents. We also allow inserting another schedule element  $s_n^i$  before  $s_n^j$ , if there is sufficient time between  $s_{n-1}^i$  and  $s_n^j$ . This means that the start time  $t_n^s$  and the associated path  $p_n^p$  could change if a new schedule element is inserted; the details of how another schedule is inserted are described in Section 4.1.2.

Initially ( $t = 0$ ), agent  $i \in I (= I_c \cup I_e)$  starts from the initial location, and it keeps stays there until  $t = t_1^s$ , where  $t_1^s$  is the start time of the first schedule element  $s_1^i (\in S_i)$ . Then,  $i$  starts moving to  $o_1^l$  along  $p_1^l$  at  $t_1^s$  in  $s_1^i$ . We assume that carrier agent  $i_c \in I_c$  begins to load/unload inside  $i_e \in I_e$  after the cooperating elevator agent  $i_e$  has arrived at the meeting location on the same floor. Note that agent  $i$  cannot stop in the middle of its movement to the next location/floor. After completing the transportation specified by  $s_1^i$ ,  $s_1^i$  is removed from  $S_i$ , and then  $i$  moves on to the next schedule,  $s_2^i$ , which was the second schedule  $s_2^j$  before removing the previous  $s_1^i$ .

#### 4.1.2 Incorporating New Schedule Elements

Because DSMAPD tasks require synchronization with other agents and this is considered as a constraint, the time space between schedule elements may not be completely filled, and long blank periods of time with no work may arise. Therefore, an agent should consider to add a new schedule between the existing schedule elements to shorten unnecessary waiting periods as long as it does not disrupt the scheduled synchronized activities. However, because the actual scheduling time can only be determined after a negotiation with the cooperating agents, the agent has to identify as many as possible scheduling times for reaching an agreement with them. Note that the loading and/or unloading locations  $o^l$  and/or  $o^e$  are determined in advance because they are in the part of the new task  $\tau = (o^p, o^d, \mu)$ .

When agent  $i$  adds a new schedule element  $s^i = \{t^s, t^l, t^e, o^s, o^l, o^e, p^{s \rightarrow l}, p^{l \rightarrow e}, \mu, \tau\}$  to a possible time interval, we can consider three possible cases, as shown in Fig. 3, in which the horizontal bars represent the start and end times of schedule elements in the current schedule list, the blue boxes represent the existing schedules, and the green boxes are new schedule elements that  $i$  tries to add to the schedule list. In the first case (Case (a)),  $i$  will add the new schedule to the end of the list. In Cases (b) and (c),  $i$  tries to add it before a schedule element that is already approved. We assume that the loading and unloading locations,  $o^l$  and  $o^e$ , are also determined when  $i$  is required to schedule for a cooperative task.

**Case (a):** Agent  $i$  can set the synchronization time later because it has no subsequent scheduled activities. That is,  $i$  can insert the new schedule element  $s^i$  at the end of  $S_i$  by setting  $t^s = t_{n_0}^e + T_{lu}$  (where  $n_0 = |S_i|$ ). Then,  $i$  can generate a collision-free path  $p^{s \rightarrow l}$  from  $o^s (= o_{n_0}^e)$  to the loading location  $o^l$  for  $\tau$  and thus can determine its (earliest) loading time  $t_{ear}^l$ . Next,  $i$  generates a collision-free path  $p^{l \rightarrow e}$  from  $o_k^l$  at  $t_{ear}^l + T_{lu}$  to  $o^e$  and can determine the earliest unloading time  $t_{ear}^e$ , which is the possible earliest time of  $t^e$  in the new schedule element. Then, suppose that  $i$  is a carrier agent. By introducing the maximal interval size  $L > 0$ , if  $o^l$  is the synchronization location, the possible synchronization time can be  $[t_{ear}^l, t_{ear}^l + L]$ . If  $o^e$  is the synchronization location, the possible synchronization time can be  $[t_{ear}^e, t_{ear}^e + L]$ . If  $i$  is an elevator agent,  $i$  will first select the synchronization time  $t^l$  from  $[t_{ear}^l, t_{ear}^l + L]$  through a negotiation, as discussed in Section 4.2.

**Case (c):** We now consider Case (c) (Case (b) is a special case of Case (c)). In Case (c),  $i$  decides whether  $s^i$  can be inserted between  $s_{n-1}^j$  and  $s_n^j$ . First,

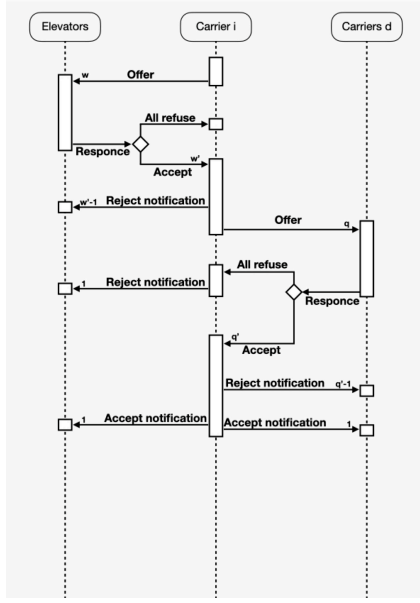


Figure 4: Interaction between agents.

$i$  builds a collision-free path  $p^{s \rightarrow l}$  starting from  $o_{n-1}^e$  at  $t_{n-1}^e$  to  $o^l$  and then  $i$  can determine the earliest time  $t_{ear}^l$  to arrive at  $o^l$  moving along  $p^{s \rightarrow l}$ . Next,  $i$  builds path  $p^{l \rightarrow e}$  starting from  $o^l$  at  $t^l + T_{lu}$  to  $o^e$ . Then, using  $p^{s \rightarrow l}$ ,  $i$  can determine the earliest time  $t_{ear}^e$  when  $i$  reaches  $o^e$ . Subsequently,  $i$  also creates another  $\tilde{p}^{s \rightarrow l}$  from  $o^e$  at  $t_{ear}^e + T_{lu}$  to  $o_n^l$  so that  $i$  arrives there before  $t_n^l$ ; if  $i$  cannot create such a path,  $i$  renounces to insert  $s^i$  between  $s_{n-1}^i$  and  $s_n^i$ .

If  $i$  is a carrier agent, it attempts to load/unload  $\mu$  synchronizing with an elevator agent. The time interval of the possible synchronization times can be calculated by considering the earliest and latest loading or unloading time  $t_{ear}^l$  and  $t_{ear}^e$ ; the details of algorithm are presented in Alg. 1 and Alg. 2. If  $i$  is an elevator agent, because  $i$  requires the synchronization at two locations,  $o^l$  and  $o^e$ , two resulting time intervals in  $FindSyncTime(i, s_n, t^s, o^s, o^l, o^e)$  are used.

**Case (b):** By setting  $t^s = t_c$  (current time) and  $o^s = o_c^i$  ( $i$ 's current location), we can identify Case (b) from Case (c). For example, if  $i$  is the elevator agent,  $i$  can determine them by  $FindSyncTime(i, s_1, t_c, o_c^i, o^l, o^e)$ , where  $s_1$  is the first element of  $S_i$ . Note again that loading and unloading locations  $o^l$  and  $o^e$  are automatically determined depending on the synchronization with agent  $i$ .

## 4.2 Negotiation Process

To perform a task in DSMAPD that includes both vertical and horizontal transportation, we propose a task negotiation process by extending the CNP. Figure 4

shows the entire flow of the negotiation process between two carrier agents on different floors and one elevator agent for a DSMAPD task.

A carrier agent  $i$  that is not negotiating with other agents and whose time length from  $t_c$  to the last element in the schedule list is shorter than the threshold,  $T_{max}$  ( $> 0$ ), selects one task  $\tau = (o^p, o^d, \mu) \in \mathcal{T}$  whose pick up floor is identical to  $i$ 's floor. If  $\tau$  is horizontal task,  $i$  adds the schedule element  $s_\tau^i$  for  $\tau$  somewhere in its schedule list  $S_i$  without using a negotiation process and following the method in Section 4.1.2. If  $i$  can insert  $s_\tau^i$  into multiple time slots,  $i$  selects the time slot whose start time  $t_\tau^i$  is the earliest.

### 4.2.1 Negotiation with Elevator Agents

If  $f^p \neq f^d$ , carrier agent  $i$ , which is shown as "Carrier  $i$ " in Fig. 4, also works as the manager agent to construct the group of cooperative agents for  $\tau$  and it decides its own schedule  $s_\tau^i = \{t_\tau^{s,i}, t_\tau^{l,i}, t_\tau^{e,i}, o_\tau^{s,i}, o_\tau^{l,i}, o_\tau^{e,i}, p_\tau^{s \rightarrow l,i}, p_\tau^{l \rightarrow e,i}, \mu_\tau, \tau\}$  by determining the time to synchronize with the appropriately selected elevator agent and unload  $\mu$  in floor  $f^p$ . For this purpose,  $i$  initiates the negotiation with all elevator agents in  $I_e$ . First,  $i$  sends them *offer messages* that contain the current  $\tau$  and the set of the possible delivery time slots  $S_{UTI}$ , which is the set of possible intervals of time in which  $i$  can deliver the material  $\tau$  to the location of each elevator; this means that it is also the set of possible times for synchronization between them. These time intervals can be calculated by considering Cases (a) to (c) and using the method described in Section 4.1.2. Note that the unload location  $o_\tau^{e,i}$  is different for each elevator  $i_e \in I_e$  because  $o_\tau^{e,i}$  is the loading location  $o_\tau^{l,i_e} = (f^p, v_{i_e})$  of the cooperating elevator agent  $i_e$ .

After receiving the offer message from  $i$ , elevator agent  $i_e$  decides whether it can create the synchronized schedule by referring to its content. Therefore,  $i_e$  invokes the method in Section 4.1.2 to find the *common synchronization times* (CSTs) for loading  $\mu$  by calculating possible pickup time slots of  $i_e$ . We denote this set of CSTs between  $i$  and  $i_e$  by  $S_{CSTi,i_e}$ . If  $i_e$  can find them, it sends back to  $i$  an *acceptance message* with  $S_{CSTi,i_e}$  and the time  $c_{i_e}^{l \rightarrow e}$  required for  $e$  to travel from  $f^p$  to the delivery floor  $f^d$ . The value  $c_{i_e}^{l \rightarrow e}$  is used to calculate the CST for unloading  $\mu$  at  $f^d$ . Otherwise, it returns a refusal message (Fig. 4). Note that if  $i_e$  receives multiple offer messages at the same time,  $i_e$  sends the acceptance message to one carrier agent  $i$  with which  $i_e$  has the best possibilities for synchronization, that is,  $S_{CSTi,i_e}$  is the largest. If  $i_e$  is participating in another negotiation in which  $i_e$  is still waiting for the answer from a carrier,  $i_e$  also returns a

Algorithm 1: Find synchronization time (Case (c)).

---

**Function** *FindSyncTime*( $i, s_n, t^s, o^s, o^l, o^e$ ):

**In:**  $i$ : Agent,  $s_n$ :  $n$ -th element in the schedule list  
 // Agent  $i$  tries to insert a new schedule element between  $s_{n-1}$  and  $s_n$

**In:**  $t^s$ : Earliest start time ( $= t_{n-1}^e$ );  $o^s$ : the endpoint  $o_{n-1}^e$  in  $s_{n-1}$ ;  $o^l, o^e$ : Required (un)load locations  
 Generate the shortest collision-free path  $p^{s \rightarrow l}$  from  $o^s$  at  $t^s$  to  $o^l$  by using e.g., CoopA\*;  
 $t_{ear}^l \leftarrow$  the earliest arrival time to  $o^l$  by moving along  $p^{s \rightarrow l}$ .;  
 Generate the shortest collision-free path  $p^{l \rightarrow e}$  from  $o^l$  at  $t_{ear}^l + T_{lu}$  to  $o^e$ ;  
 $t_{ear}^e \leftarrow$  the time when  $i$  arrives  $o^e$  along  $p^{l \rightarrow e}$ .;  
 Generate the shortest collision-free path  $\tilde{p}^{s \rightarrow l}$  from  $o^e$  at  $t_{ear}^e + T_{lu}$  to  $o_n^l$  in  $s_n$ ;  
**if**  $i$  arrives at  $o_n^l$  at or after  $t_n^l$  **then return** nil // Fail to insert  
 ;  
**if** the synchronization location is the place to unload **then**  
 | **return** Time interval from  $t_{ear}^e$  to  $\text{LatestStartTime}(i, o^e, t_{ear}^e + T_{lu}, o_n^l, t_n^l) - T_{lu}$ ;  
**else**  
 |  $t_s = \text{LatestStartTime}(i, o^e, t_{ear}^e + T_{lu}, o_n^l, t_n^l) - T_{lu}$ ;  
 | **return** Time interval from  $t_{ear}^l$  to  $\text{LatestStartTime}(i, o^l, t_{ear}^l + T_{lu}, o^e, t_s)$ ;  
**end**

---

Algorithm 2: Decide the latest start time.

---

**Function** *LatestStartTime*( $i, o^1, t^1, o^2, t^2$ ):

**In:**  $i$ : agent;  $o^1$ : start location  
**In:**  $t^1$ : earliest possible start time from  $o^1$   
**In:**  $o^2$ : destination;  $t^2$  due time to  $o^2$   
 // Agent  $i$  tries to decide the latest time to  
 // start from  $o^e$  to  $o_n^l$  before  $t_n^l$  in  $s_n$ .  
 $L > 0$ : Upper limit of interval length;  
**for**  $t = t^1$  **to**  $t^1 + L$  **do**  
 | **if**  $i$  can create a collision-free path  
 | from  $o^e$  to arrive at  $o_n^l$  earlier than  $t_n^l$   
 | **then**  
 | | do nothing  
 | **else**  
 | | **return**  $t - 1$  // Latest time  
 | **end**  
**end**  
**return**  $t^1 + L$  // Latest time defined by  $L$ .

---

refusal message; then,  $i$  decides that the current negotiation for  $\tau$  has failed and declines to perform it.

After agent  $i$  receives all responses from the elevator agents,  $i$  determines the best acceptance response, which is the largest  $S_{CST_{i,i_e}}$ , from the elevator agent  $i_e$ . Next,  $i$  sends rejection messages to the other elevator agents, and the elevator agents receiving the rejection messages finish negotiation with  $i$ . Then,  $i$  generates the schedule  $s_\tau^l$  so that its unloading time is the earliest in  $S_{CST_{i,i_e}}$ . If  $i$  receives no acceptance messages, it finishes the negotiation for  $\tau$  and sends the rejection messages to all elevator agents.

#### 4.2.2 Negotiation with Carrier Agents on Delivery Floor

Next, manager agent  $i$  starts the negotiation with carrier agents  $I_d$  on the delivery floor  $f^d$  (these agents are shown as “Carriers\_d” in Fig. 4). Before starting it,  $i$  calculates the set of possible synchronization times,  $S_{CST_{i_e, I_d}}$  with carrier agents on  $f^d$  by considering  $C_{i_e}^{l \rightarrow e}$  and  $S_{CST_{i, i_e}}$ . Then,  $i$  starts to negotiate with carrier agents  $i_d$  on  $f^d$  by sending them offer messages with  $S_{CST_{i_e, I_d}}$  and the location to synchronize with  $i_e$  on  $f^d$ , which is naturally determined based on  $i_e$  with which it is cooperating.

After receiving this offer message, agent  $i_d$  decides whether it can create the schedule including synchronization. If  $i_d$  can find the CSTs,  $S_{CST_{i_e, i_d}}$ , to load  $\mu$  from  $i_e$  using the method presented in Section 4.1.2, it sends back to  $i$  an acceptance message with only the earliest loading time,  $t_{ear, i_d}^l$  in  $S_{CST_{i_e, i_d}}$ ; otherwise, it returns a refusal message. If  $i_d$  is already negotiating with other agents,  $i_d$  also returns a refusal message.

After manager agent  $i$  receives all the responses from  $I_d$ ,  $i$  determines the best acceptance message with the earliest  $t_{ear, i_d}^l$  from agent  $i_d$  and sends rejection messages to the others. Then,  $i$  calculates the earliest load time  $t_{ear, i}^l$  ( $= t_\tau^l$  in  $s_\tau^l$ ) and earliest unload time  $t_{ear, i}^e$  ( $= t_\tau^u$ ) backwards from  $t_{ear, i_d}^l$  using  $C_{i_e}^{l \rightarrow e}$  to decide its schedule  $s_\tau^l$ . Note that the other parameter values in  $s_\tau^l$  are naturally decided based on the cooperating elevator agent  $i_e$ . If  $i$  cannot receive any acceptance messages,  $i$  sends rejection messages to the

selected elevator  $i_e$ , and  $i$  finishes this negotiation for  $\tau$ . After receiving the rejection message, agent  $i_e$  also finishes the negotiation with  $i$ .

Subsequently,  $i$ , which is also a carrier agent on floor  $f^p$ , adds  $s_c^i$  to its schedule list  $S_i$  as the approved schedule element and sends the contract messages with  $i_e$ 's loading time  $t_{i_e}^l (= t_i^l)$  and unload time  $t_{i_e}^e (= t_{ear, i_d}^l)$  to the elevator agent  $i_e$  and carrier  $i_d$ . Finally, the agents receiving the contract messages from the manager agent insert  $s_c^i$  and  $s_c^{i_d}$  into their own schedule list as the approved elements. Then, the negotiation process ends.

#### 4.2.3 Remark

We assume that our environment holds the condition

$$\min_{f \in F} |\{i_c \in I_c \mid i_c \text{ on floor } f\}| > |I_e/2|. \quad (1)$$

to prevent a deadlock/livelock during the negotiation process. If we assume that the number of carrier agents on each floor is the same, then this condition can be

$$|I_c|/|F| > |I_e/2|, \quad (2)$$

We explain the possibility of a livelock situation using a simple example in which the environment consists of two floors, two carrier agents on each floor, and four elevator agents. Suppose that all carrier agents start to negotiate with elevator agents to deliver materials at all floor levels. After this, all carrier agents will receive acceptance messages from elevators and then send offer messages to carrier agents on the delivery floors. Because all carrier agents have already started a negotiation, they return refusal messages, and thus, they also receive refusal messages. Then, they simultaneously restart negotiation processes but these will also result in failure. However, our assumption is plausible and realistic in actual applications because the number of elevator is usually small, and it could stop some elevators to hold Condition (2).

## 5 EXPERIMENTS AND DISCUSSION

### 5.1 Experimental Setting

We evaluated the proposed method using DSMAPD instances under two different environments and compared the results with those using the synchronized schedule searching algorithm (SSS) as a baseline, which will be briefly explained in Section 5.2.

Our experiments were conducted in the two environments shown in Fig. 2. In the first environment

Table 1: Experimental parameter values.

Parameter description and symbol	Value
Size of environment, $N$	7
Number of tasks, $M =  \mathcal{T} $	30
No. of carrier agents in each floor, $N_C$	4 to 11
No. of elevator agents, $N_E$	6
No. of floors, $N_f$	4
Maximum length of schedule list, $T_{max}$	50
Upper limit of search time, $L$	50
Duration for moving to a next floor, $T_f$	4
Duration for loading/unloading, $T_{lu}$	2
Negotiation process per timestep, $C$	4
Upper limit of the length of instance, $H$	400

(Env. 1), each floor, as shown in Fig. 2a, had seven task endpoints (gray triangles) that could be load and unload locations of tasks, and eight elevator agents (green diamonds) where carrier agents (black circles) could load or unload materials. For simplicity, all floors in the building (Fig. 1) had the same structure as shown in Fig. 2a. In Env. 1, the elevators were arranged in two rows because actual transport elevators are usually installed together in a building. In the second environment (Env. 2), each floor also had seven task endpoints and eight elevators, but their locations were placed randomly, as shown in Fig. 2b. This scheme was used to investigate the performance of the proposed method in general cases.

Both environments consisted of  $N_f (> 0)$  floors, the initial locations of  $N_C (> 0)$  carrier agents on each floor were randomly placed, and  $N_E (> 0)$  elevators were placed on randomly selected floors. Here, the number of carrier agents on each floor was the same, thus  $|F| \times N_c = |I_c|$ . We assume that  $|\mathcal{T}|$  is constant  $M$ , and if one task is completed, it is removed from  $\mathcal{T}$  and another randomly generated task is added to  $\mathcal{T}$ . Agents continue to execute tasks until  $H > 0$  timesteps; this experiment until  $H$  is an instance of an experimental run. Then, another experimental run starts after placing endpoints, agents, and elevators randomly.

We introduced the factor  $C > 0$  for the ratio of communication time (with acknowledgement) to one timestep in which any carrier agent can move to a neighboring point. Usually, the physical movement takes relatively a much longer time than that required for communication with the associated computational processing, but we set for it a relatively large value, that is,  $C = 4$ , so that agents can confirm the received messages, select one task to cooperate, and send/reply messages to negotiate in  $1/C$  timestep. Therefore, agents can negotiate with other agents  $C$  times in one timestep to attempt to generate a synchronized sched-



ule but in parallel; thus, several negotiation processes for generating schedules can be performed simultaneously.

For evaluation, we measured the number of completed tasks and the total CPU time in an experimental instance. The number of completed tasks indicates the transportation efficiency, whereas the total CPU time indicates the planning efficiency including communication time. We list other parameter values in Table 1. All experimental data are the average of 30 independent trials using an Apple M1 Max CPU with 64 GB RAM.

## 5.2 Baseline Method

To evaluate the performance of our proposed methods for DSMAPD, we implemented the *synchronized schedule search algorithm* (SSS) as baseline because we could not find conventional studies for double synchronized tasks like ours. We briefly explain the SSS in this section.

SSS is a breadth first search algorithm that starts from the centralized root node with a DSMAPD task,  $\tau = (o^p, o^d, \mu)$ , and generates the lower (child) nodes, which correspond to carrier agents on floor  $f^p$ , and all the nodes are simultaneously requested to calculate the earliest delivery (unload) time after the already scheduled activities in the scheduled list. This is performed by generating the shortest collision-free path using CoopA\* by sharing the already approved schedule elements on floor  $f^p$ . Therefore, we do not consider an insertion between the already scheduled elements to avoid a cascade of conflicts. Then, immediately upon receiving a reply (i.e., in the arrival order of replies), the root node generates the next lower nodes, which correspond to elevator agents, below each carrier agent node and request them to find the earliest synchronization time to load the material carried by the upper node agent. Finally, it determines the earliest arrival time at floor  $f^d$  and returns the results to the root node.

In the same way, upon receiving a reply from an elevator agent, the root node generates the further lower nodes, which corresponds to the carrier agents on  $f^d$ , and requests them to decide the shortest path to the corresponding elevator agents and the earliest loading time inside the elevator. The reply to this request can generate the complete schedules. Then, the root node assigns  $\tau$  to the agents that return the earliest reply.

We assume that the SSS algorithm can generate the agents' synchronized schedules for a task in one timestep although it requires many messages between agents, and this assumption is a slight advantage over

the baseline. Therefore, for comparison, we also implemented a more efficient baseline and called it *double SSS*, which can generate synchronized schedules for two tasks in one timestep. We will show that our method is still more efficient than the baseline methods under this assumption.

## 5.3 Performance Comparison

### 5.3.1 Number of Executed Tasks

Figure 5 plots the average number of completed tasks per experimental instance with different numbers of carrier agents and six elevator agents. In Env. 1, Fig. 5a indicates that the proposed method outperformed the baselines, SSS and double SSS, in any cases with different numbers of carrier agents. For example, the proposed method increased the number of completed tasks by approximately 21% and 11% over those of SSS and double SSS, when the number of carrier agents on each floor was  $N_C = 11$ .

In the baseline methods, new schedule elements were added at the end of the agents' schedule lists to prevent potential conflicts with the approved schedule elements. In particular, when the number of carrier agents was relatively large compared to that of the elevator agents, many schedule elements were added to the lists of the elevator agents, thus forcing them to defer many synchronization times. By contrast, because agents with the proposed method could insert new schedule elements between the approved schedule elements, they could reduce the wasted free time and could execute tasks more efficiently. In Env. 2, as shown in Fig. 5b, with the proposed method the agents could also improve the performance more than with the baseline methods. Therefore, we consider that our proposed method can achieve efficient transportation for DSMAPD in more general cases.

### 5.3.2 Planning CPU Time

We also investigated the averaged planning CPU time for all agents per instance with a different number of carrier agents in Env. 1. The result is plotted in Fig. 6. As can be observed, the planning time of SSS is much smaller than those of other methods regardless of the number of agents. Actually, compared to the proposed method, SSS reduced the planning time by approximately 62% when  $N_C = 4$  and by approximately 42% when  $N_C = 11$ . This was achieved because SSS generated schedule elements in each agent by simply adding them at the end of the schedule list, starting or ending at the synchronization point and time. All conflicts (i.e., collision in our experiments) could be avoided by CoopA\*.

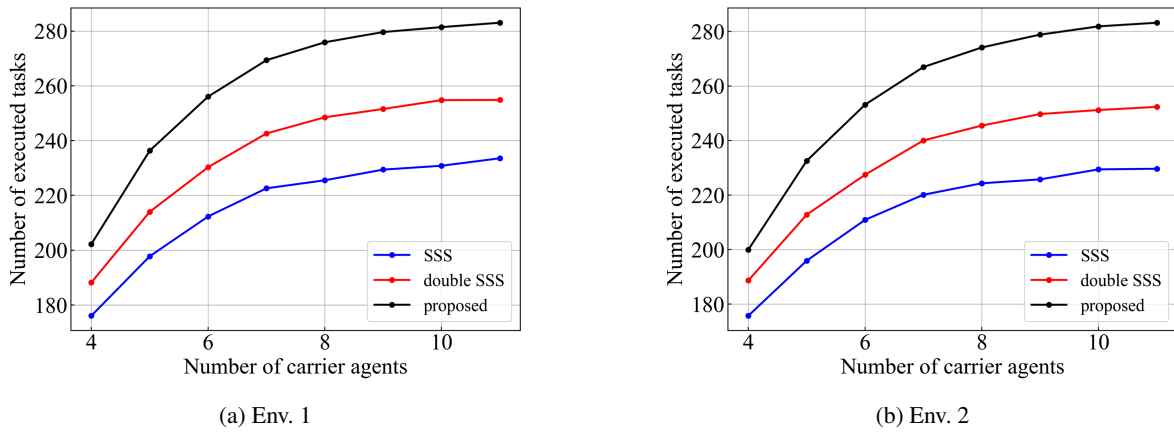


Figure 5: Number of completed tasks.

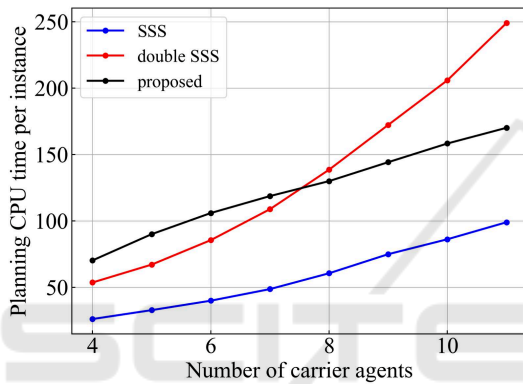


Figure 6: Planning time per instance (second) in Exp. 1.

This figure also reveals that the planning time with the proposed method was longer than that with the other methods when the number of agents was  $N_C < 8$ , but the planning time with double SSS became longer than that with the proposed method when  $N_C \geq 8$ . In double SSS, after an agent generated synchronized schedules for the first task, the root node attempted to generate synchronized schedules for the second task. To generate synchronized schedules for the second task, it was forced to search a larger number of nodes than those in the first task because it found the synchronized time with less wasting time right after the schedule for the first task. Therefore, the computational cost was higher than that with SSS and the proposed method. Although we omit the figure for the averaged planning CPU time in Env. 2, a similar tendency was observed.

## 5.4 Discussion

The proposed method outperformed the baseline methods (SSS and *double SSS*) in two environments regardless of the number of carrier agents. To investigate the factors contributing to the performance, we

Table 2: Occupancy rate of agents (%).

Algorithm	Number of carrier agents $N_C$							
	4	5	6	7	8	9	10	11
Elevator agent								
Proposed	.87	.91	.93	.95	.96	.97	.97	.98
SSS	.78	.84	.88	.90	.91	.92	.93	.93
double SSS	.81	.87	.90	.92	.93	.94	.95	.96
Carrier agent								
Proposed	.76	.71	.64	.58	.52	.48	.43	.39
SSS	.67	.60	.54	.49	.44	.39	.36	.33
double SSS	.69	.64	.58	.52	.48	.43	.39	.36

analyzed the occupancy rate of agents' schedules with different numbers of agents in Env. 1 and list the results in Table 2, where the high rate means that the cumulative waiting time between schedule elements in one experimental instance is small. Table 2 indicates that the occupancy rates of elevator and carrier agents with the proposed method were always higher than those of agents with the baselines, probably because the elevator agents with the proposed method could avoid long free periods of time. This is the reason for the higher efficiency of the proposed method.

The table also indicates that the occupancy rates of elevator agents increased with increasing number of carrier agents regardless of the method employed, whereas the rates of the carrier agents decreased. Because the number of elevator agents is fixed and relatively small, the number of offer messages from carrier agents increased and naturally the waiting periods of elevator agents decreased. In contrast, the offer messages from the carrier agents became likely to be refused because of the simultaneous offer messages to elevator agents, and thus, the carrier agents lost the possibility to add new schedules, causing the relatively longer free time. This also means that elevator

agents are the bottleneck of performance. However, in actual applications, it is more difficult to increase the number of elevators owing to the much higher cost and effort required for their installation.

Interestingly, we found that the planning time of double SSS exceeded that of the proposed method when  $N_c \geq 8$ . Furthermore, the difference in planning time between SSS and the proposed method became gradually smaller with an increase in the number of carrier agents  $N_c$ . This occurs because the baseline methods need to generate a larger number of lower (child) nodes as the number of carrier agents increase, and thus, the root node in SSS and double SSS is required to visit all the nodes before moving on to the nodes at the next depth level. This discussion also suggests that many messages are necessary in the baseline methods. Meanwhile, agents using the proposed method could discriminate better proposals from appropriate agents through negotiation, and this resulted in higher efficiency and fewer messages.

## 5.5 Conclusion

We propose a scheduling and negotiation method for tasks that require double synchronization between heterogeneous agents. To achieve efficient task executions, our method enables agents to generate their schedule by determining the effective synchronization times and reducing the unnecessary idle time through a negotiation process, which is an extension of the CNP. It also decides the appropriate cooperative agents step by step, thus reducing the number of messages during the negotiation process. We experimentally showed that the proposed method outperformed the baseline methods, SSS and double SSS, for DSMAPD problem instances. It also involved a reasonable computational cost in environments where many agents are required to cooperate with other agents to complete DSMAPD tasks.

In the future, we plan to extend our method to more complex tasks, such as tasks that need more synchronization times and locations and tasks that have their own deadlines. Another extension is that for tasks in which agents cannot move at a constant speed. We also want to integrate our method with one of the recent path-finding algorithms for MAPD (Sharon et al., 2015; Ma et al., 2017; Yamauchi et al., 2022; Okumura et al., 2022; Miyashita et al., 2023).

## REFERENCES

- Ahumada, G. I. and Herzog, O. (2021). Application of multiagent system and tabu search for truck dispatching in open-pit mines. In 0001, A. P. R., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence, ICAART 2021, Volume 1, Online Streaming, February 4-6, 2021*, pages 160–170. SCITEPRESS.
- Ahumada, G. I., Riveros, E., and Herzog, O. (2020). An agent-based system for truck dispatching in open-pit mines. In *ICAART (1)*, pages 73–81.
- Alexandre, R. F., Campelo, F., and Vasconcelos, J. a. A. (2017). Multiobjective evolutionary algorithms for operational planning problems in open-pit mining. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '17*, pages 259–260, New York, NY, USA. Association for Computing Machinery.
- Bastos, G. S., Souza, L. E., Ramos, F. T., and Ribeiro, C. H. C. (2011). A single-dependent agent approach for stochastic time-dependent truck dispatching in open-pit mining. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1057–1062.
- Bredström, D. and Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1):19–31.
- Das, D. N., Sewani, R., Wang, J., and Tiwari, M. K. (2021). Synchronized truck and drone routing in package delivery logistics. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):5772–5782.
- Fumero, F. and Vercellis, C. (1999). Synchronized development of production, inventory, and distribution schedules. *Transportation science*, 33(3):330–340.
- Ge, H., Matsui, M., and Koshizuka, N. (2021). An open-iot approach on elevator for enabling autonomous robotic vertical mobility. In *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, pages 139–141.
- Kafle, N., Zou, B., and Lin, J. (2017). Design and modeling of a crowdsourcing-enabled system for urban parcel relay and delivery. *Transportation Research Part B: Methodological*, 99:62–82.
- Kumar, R. R., Varakantham, P., and Cheng, S.-F. (2023). Strategic planning for flexible agent availability in large taxi fleets. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '23*, pages 552–560, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- Luo, Q., Wu, G., Ji, B., Wang, L., and Suganthan, P. N. (2022). Hybrid multi-objective optimization approach with pareto local search for collaborative truck-drone routing problems considering flexible time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):13011–13025.
- Ma, H., Li, J., Kumar, T. S., and Koenig, S. (2017). Life-long multi-agent path finding for online pickup and

- delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS '17*, pages 837–845, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- Miyashita, Y., Yamauchi, T., and Sugawara, T. (2023). Distributed planning with asynchronous execution with local navigation for multi-agent pickup and delivery problem. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 914–922.
- Okumura, K. (2023). Lacam: Search-based algorithm for quick multi-agent pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):11655–11662.
- Okumura, K., Machida, M., Défago, X., and Tamura, Y. (2022). Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752.
- Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66.
- Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, pages 117–122.
- Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, 29(12):1104–1113.
- Yamauchi, T., Miyashita, Y., and Sugawara, T. (2022). Standby-based deadlock avoidance method for multi-agent pickup and delivery tasks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1427–1435.
- Zhang, X., Chen, L., Ai, Y., Tian, B., Cao, D., and Li, L. (2021). Scheduling of autonomous mining trucks: Allocation model based tabu search algorithm development. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 982–989.
- Zhang, X., Guo, A., Ai, Y., Tian, B., and Chen, L. (2022). Real-time scheduling of autonomous mining trucks via flow allocation-accelerated tabu search. *IEEE Transactions on Intelligent Vehicles*, 7(3):466–479.