



Efficient Path and Action Planning Method for Multi-Agent Pickup and Delivery Tasks under Environmental Constraints

Tomoki Yamauchi¹ · Yuki Miyashita¹ · Toshiharu Sugawara¹

Received: 31 January 2022 / Accepted: 25 October 2022 / Published online: 8 December 2022
© The Author(s) 2022

Abstract

We propose a method called *path and action planning with orientation* (PAPO) that efficiently generates collision-free paths to satisfy environmental constraints, such as restricted path width and node size, for the *multi-agent pickup and delivery in non-uniform environment* (N-MAPD) problem. The MAPD problem, wherein multiple agents repeatedly pick up and carry materials without collisions, has attracted considerable attention; however, conventional MAPD algorithms assume a specially designed environment and thus use simple, uniform models with few environmental constraints. Such conventional algorithms cannot be applied to realistic applications where agents need to move in more complex and restricted environments. For example, the actions and orientations of agents are strictly restricted by the sizes of agents and carrying materials and the width of the passages at a construction site and a disaster area. In our N-MAPD formulation, which is an extension of the MAPD problem to apply to non-uniform environments with constraints, PAPO considers not only the path to the destination but also the agents' direction, orientation, and timing of rotation. It is costly to consider all these factors, especially when the number of nodes is large. Our method can efficiently generate acceptable plans by exploring the search space via path planning, action planning, and conflict resolution in a phased manner. We experimentally evaluated the performance of PAPO by comparing it with our previous method, which is the preliminary version of PAPO, the baseline method in a centralized approach, and fundamental meta-heuristic algorithms. Finally, we demonstrate that PAPO can efficiently generate sub-optimal paths for N-MAPD instances.

Keywords Multi-agent pickup and delivery problem · Multi-agent path finding · Multi-agent planning · Robot planning · Environmental constraints

Yuki Miyashita and Toshiharu Sugawara have contributed equally to this work.

This article is part of the topical collection “Advances in Multi-Agent Systems Research: EUMAS 2021 Extended Selected Papers” guest edited by Davide Grossi, Ariel Rosenfeld and Nimrod Talmon.

✉ Tomoki Yamauchi
t.yamauchi@isl.cs.waseda.ac.jp

Yuki Miyashita
y.miyashita@isl.cs.waseda.ac.jp

Toshiharu Sugawara
sugawara@isl.cs.waseda.ac.jp

¹ Department of Computer Science and Communications Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 1698555, Japan

Introduction

In recent years, *multi-agent system* (MAS) technology has exhibited remarkable promise for automating complex and enormous tasks in real-world applications using coordinated and cooperative actions and combinations of heterogeneous abilities and skills. For example, transport robots in automated warehouses (Wurman et al. [35]), autonomous aircraft-towing vehicles in airports (Morris et al. [22]), ride-sharing services (Li et al. [15]; Yoshida et al. [40]), office robots (Veloso et al. [32]), and multiple-drone delivery systems (Krakowczyk et al. [12]). However, because of resource conflicts between agents, such as collisions and redundant actions by multiple agents, the simple increase in the number of agents often leads to inefficiency. Therefore, to improve performance and avoid negative mutual effects, cooperative and/or coordinated actions are essential for actual use. Resolving conflicts between agents, i.e., collision avoidance,

is particularly essential in our envisioned pickup and delivery system with robots that carry heavy and large materials in a constrained environment.

Therefore, the corresponding problems are often formulated as *multi-agent pickup and delivery* (MAPD) problems, where multiple carrying tasks are assigned to multiple agents with each task one after the other. Once an agent is assigned a task, it is required to travel to the material storage area, load the required material, carry it to the specified location, and unload it. From a planning viewpoint, the MAPD problem can be regarded as an iteration of *multi-agent path finding* (MAPF), wherein each agent generates a collision-free path from the starting to the ending location. Unfortunately, MAPF is regarded as a non-deterministic polynomial-time hardness (NP-hard) problem for generating optimal paths (Ma et al. [17]). Moreover, the MAPD problem is challenging and time-consuming because it requires a large number of carrying tasks. Nevertheless, it is necessary to efficiently create acceptable paths for the movement of multiple agents in the MAPD problem.

Several studies have focused on the MAPF/MAPD problem (Liu et al. [16]; Ma et al. [19]; Sharon et al. [26]). Although their proposed methods were applied to actual systems, they often assumed a specially designed environment with few constraints on agent operations. Most of these studies formulated the environments as grids without considering the different path widths, agent sizes, and material sizes. Therefore, agents can move without considering their rotation and moving direction. However, in our application, we have envisioned a transportation task in a construction site or a rescue-with-robot task in a disaster area. In contrast to the aforementioned studies, our environments are more complicated and have more constraints. For example, let us consider autonomous forklift-type carrier agents with a picker or rescue robots with an arm. In addition to the passages with various widths at a construction site, agents typically have to transport large materials that are wider than themselves. Therefore, an agent with a large material can only pass through a narrow path in a certain orientation, and in some places, an agent may not be able to rotate due to obstacles. This implies that each location and path may have their own constraints in their environment; thus, agents should generate different routes depending on whether they are transporting materials. Furthermore, at a construction site, the passage width and the topology can easily change; for example, a new wall is built which was not there the day before, or a pile of materials is placed as an obstacle in a passage and removed the next day. This indicates that learning methods that require a large amount of training data are not preferred.

In this study, first, we formulate the *multi-agent pickup and delivery in non-uniform environment* (N-MAPD) problem, an extension of the MAPD problem, to model the

aforementioned complex situations. For example, in a construction site, agents may be prohibited from actions in certain locations and passages due to constraints on their size and width. Therefore, the agents are required to consider their sizes (including the size of the materials if they carry). Instead, we assume that the agent can move in any direction (left, right, up, down) without changing its orientation. In such an environment, the agent needs to decide on a travel path that reflects collision avoidance and environmental constraints. For example, the simple shortest path without considering constraints may be inappropriate because if a part of the path is narrow, the agent with the material has to change its orientation before passing through to move sideways, but such additional rotate action may require additional time. We believe that N-MAPD formulation can also be exploited in other scenarios where physical constraints have to be considered, such as a multi-agent disaster rescue problem.

Considering this issue, we propose *path and action planning with orientation* (PAPO), which is an algorithm for solving an N-MAPD problem. It constructs collision-free paths and the associated sequence of actions in two phases: *two-stage action planning* (TSAP), wherein the agent builds several short paths to its destination and then generates the set of action sequences along each path, and *conflict resolution with candidate action sequences* (CRCAS), wherein the agent generates a conflict-free (so *approved*) action sequence by observing the previously approved plans of other agents in the *synchronized block of information* (SBI) to avoid conflicts. The PAPO algorithm considers not only the direction and timing of movement but also the orientation of the agent and the time required for each action (i.e., time cost) for solving an N-MAPD problem. Furthermore, our PAPO algorithm also considers sophisticated processes for resolving conflicts and satisfying environmental constraints, because agents have to make appropriate decisions to avoid conflicts, such as waiting (for synchronization), detouring, or changing the order of actions. PAPO can efficiently generate plans by exploring the search space for the N-MAPD problem via path planning, action planning, and conflict resolution in a phased manner. We have already reported the effectiveness of the preliminary version of PAPO (called *preliminary PAPO*) for the N-MAPD problem in our previous study (Yamauchi et al. [37]). However, in this study, we have further improved it using a more effective strategy for modifying plans.

Later, we experimented with the performance of the new PAPO by comparing its results with those of (a) a naive centralized method as a baseline method, (b) preliminary PAPO, (c) *ant colony optimization* (ACO), and (d) *simulated annealing* (SA) in a variety of N-MAPD experimental settings. Experimental results indicate that PAPO can generate sub-optimal but sufficiently acceptable paths more efficiently compared to other methods. We also conducted experimental

evaluation with various parameter settings to understand the features of the proposed algorithm. Finally, we comprehensively discuss with comparing the experimental results with the preliminary PAPO and the baseline method.

Related Work

The MAPF/MAPD problem has been studied from a variety of perspectives (Felner et al. [7]; Ma et al. [18]; Salzman and Stern [25]). Based on the coordinated and cooperative structures among agents, we can classify them as centralized and decentralized approaches. Examples of the centralized approach include the *conflict-based search algorithm* (CBS) (Sharon et al. [26]) of MAPF and its extensions (Bellusci et al. [4]; Boyarski et al. [5]; Boyrasky et al. [6]; Zhang et al. [41]; Huang et al. [10]). CBS is a planning algorithm comprising two stages: *low-level search*, wherein agents individually determine their paths, and *high-level search*, wherein a centralized planner generates sequences of actions while checking for conflicts between agents and resolving them. The decentralized approaches include those that guarantee completeness under certain restrictions (Ma et al. [19]; Okumura et al. [23]; Wang and Rubenstein [33]; Wang and Botea [34]; Okumura et al. [24]). For example, Ma, Li, Kumar, and Koenig (Ma et al. [19]) proposed a well-known decentralized MAPD algorithm called *token passing* (TP). Here, each agent can refer to the *token*, which is a synchronized shared memory block; it individually chooses a task that satisfies the certain conditions and generates collision-free paths. However, these studies assume a simplified environment and ignore constraints, such as individual agent speed, path width, agent size, and duration of each action. As a result, real-world applications are limited.

Meanwhile, there are several studies that use the model including rotation, size, and movement speed of the agent (Barték et al. [3]; Ho et al. [8]; Hönig et al. [9]; Kou et al. [11]; Li et al. [13]; Ma et al. [20]; Machida [21]; Surynek. [29]; Yakovlev et al. [36]). For instance, Ma et al. ([20]) proposed *TP-safe interval path planning with reservation table* (TP-SIPPwRT) to include the agent's rotation and movement direction into their model. However, this study assumed a custom-designed environment with normalized/fixed path width and length. Therefore, its application in our target environment is not feasible. Although some studies in the area of trajectory planning have focused on kinematic constraints during planning (Alonso-Mora et al. [1]; Bareiss and van den Berg [2]; Li et al. [14]; Tang and Kumar [30]), our study differs in that it aims to efficiently complete the MAPD iterative task in tight and cluttered environments.

Certainly, conventional algorithms can be applied to our N-MAPD problem by adding orientations to agent states and

considering environmental constraints. Since path planning using conventional algorithms requires a two-dimensional space comprising temporal and spatial dimensions, the search space for the N-MAPD problem becomes immense when the orientation dimensions related to various path widths and agent sizes are added. Therefore, if the optimal path must be found using naive search as in conventional algorithms, the computational cost increases in this three-dimensional search space. To the best of our knowledge, no study exists on path planning with conflict resolution on discrete graphs under behavioral constraints due to the shape and size of agents, spaces, and materials being carried.

Background and Problem Formulation

We have extended the conventional MAPD problem and formulated the N-MAPD problem by introducing path widths, material and agent sizes, and the required time for agent actions. To enhance the readability, we list a summary of notations in Tables 2 and 3 in Appendix A.

Problem Formulation for N-MAPD

The N-MAPD problem represented by tuple (A, \mathcal{T}, G) , where $A = \{1, \dots, M\}$ is a set of M agents, $\mathcal{T} = \{\tau_1, \dots, \tau_N\}$ is a set of N tasks, and $G = (V, E)$ is an undirected connected graph that can be embedded in a two-dimensional Euclidean space described by x - and y -coordinates. Node $v \in V$ and edge $(u, v) \in E$ ($u, v \in V$) represent a location and path that an agent can travel between u and v in the environment, respectively. L_v and W_v are the length and width of the node v , respectively. Moreover, the edge (u, v) has width W_{uv} and distance $\text{dist}(u, v)$, which is defined as the length between the centers of u and v in the Euclidean space. Our agent is, for example, a forklift-type robot with a picker in front and can carry heavy material. We assume that a material (or a pile of materials) is on a rack base, and the agent can load it (pick it up) or unload it (put it down) at one of the specified nodes using its picker towards a particular direction. We introduce discrete-time $t \in \mathbb{Z}_+$ (where \mathbb{Z}_+ is the set of non-negative integers) and assume that certain durations are required for agent actions, such as wait action, move action toward a neighbor node, rotate action, and the load and unload actions of a material. Figure 2 indicates examples of our environments.

For agent $i \in A$, let us denote the (moving) *direction* $d_i^t > 0$ and the *orientation* $o_i^t > 0$ of i at time t , where $0 \leq d_i^t, o_i^t < 360$ in D increments. For example, if we set $D = 90$, there are four directions and orientations: $d_i^t, o_i^t = 0, 90, 180, 270$. Although D can be any number depending on environmental features, for the sake of simplicity, we assume $D = 90$ in this study. We set the north

orientation/direction in the environment G to 0, i.e., $o_i^t = 0$ and $d_i^t = 0$. Therefore, the set of possible orientations can be expressed by $\mathcal{D} = \{0, 90, 180, 270\}$ since $D = 90$. We also express the size of agent $i \in A$ by its width W_i and length (or depth) L_i . Thereafter, the x -axis length w_i^t and y -axis length l_i^t of i at time t are obtained as follows:

$$\begin{aligned} w_i^t &= |L_i \sin o_i^t| + |W_i \cos o_i^t|, \\ l_i^t &= |W_i \sin o_i^t| + |L_i \cos o_i^t|. \end{aligned} \tag{1}$$

The material that is requested to carry by task τ_k also has a size whose width and length are denoted by W_{τ_k} and L_{τ_k} , respectively. While i carries the material associated with τ_k , i 's size will change to

$$\begin{aligned} W_i &\leftarrow \max(W_i, W_{\tau_k}), \\ L_i &\leftarrow \max(L_i, \gamma L_i + L_{\tau_k}), \end{aligned} \tag{2}$$

where the non-negative number γ ($\gamma \leq 1$) is the ratio of the agent's length to the length of its fork part. We have indicated the size and orientation of agents with and without materials for $\gamma = 0.5$ in Fig. 1.

Agents need to determine their action sequences for carrying materials while considering the constraints on the path width and node size configured in the environment, and agent size calculated using Formulae 1 and 2. Henceforth, the constraints defined in the environment, such as the width and size of routes, nodes, and agents, will be termed as *environmental constraints*.

Agents execute the following actions: *rotate*, *move*, *load*, *unload*, and *wait*. The durations of actions, *rotate*, *move*, *load*, *unload*, and *wait* are denoted by $T_{ro}(\theta)$, $T_{mo}(l)$, T_{ld} , T_{ul} , and $T_{wa}(t)$, respectively, where $\theta \in \mathbb{Z}_+$ is the rotation angle, $l = \text{dist}(u, v)$ is the moving distance between nodes u and v , and t is the waiting time. For example, $T_{mo}(1) = 10$ when $l = 1$. Suppose at time t , agent $i \in A$ is on a node $v \in V$; then, agent i can move along edge $(u, v) \in E$ to u with the *move* action without changing the orientation of o_i^t if the edge is sufficiently wide. The *rotate* action makes i rotate D degrees clockwise (D) or counter-clockwise ($-D$) from o_i^t , i.e., $o_i^{t+T_{ro}(D)} \leftarrow o_i^t \pm D$, staying at v , if the node size is sufficient. A parking location $park_i \in V$, which is the starting

location at $t = 0$, is uniquely allocated to each agent i (Liu et al. [16]). Agents may return to their parking locations unless they have a task to perform. In Fig. 2, parking locations are indicated by red squares.

We specify the task τ_j by the tuple $\tau_j = (v_{\tau_j}^{ld}, v_{\tau_j}^{ul}, W_{\tau_j}, L_{\tau_j}, \phi_{\tau_j})$, where $v_{\tau_j}^{ld} = (v_{\tau_j}^{ld}, o_{\tau_j}^{ld}) (\in V \times \mathcal{D})$ is the location and orientation of the loading material ϕ_{τ_j} , $v_{\tau_j}^{ul} = (v_{\tau_j}^{ul}, o_{\tau_j}^{ul}) (\in V \times \mathcal{D})$ is the location and orientation of the unloading ϕ_{τ_j} , and W_{τ_j} and L_{τ_j} are the width and length of ϕ_{τ_j} , respectively. Considering the direction of the picker and the shape of the material, an agent needs to be oriented in a specific direction when loading and unloading the material. Agents have to complete all the tasks in \mathcal{T} with no collision and no violation of environmental constraints. Agent i returns to $park_i$ for recharging when it completes all tasks in \mathcal{T} .

Well-Formed N-MAPD Problem Instance

Not all MAPD instances may be able to be solved. For example, if the environment is not connected, agents cannot reach some nodes. Therefore, Ma et al. ([19]) introduced *well-formed* MAPD problem instances.

An MAPD instance is well formed if and only if the following three conditions are satisfied:

- (1) The number of requested tasks \mathcal{T} is finite.
- (2) Parking locations are different from all the pickup and delivery locations specified by tasks.
- (3) At least one path exists between any two start/goal locations such that it does not traverse any other start/goal locations.

We modify condition (3) to reflect the environmental constraints in N-MAPD as follows:

- (3') At least one *feasible path* exists between any two start/goal locations such that it does not traverse any other start/goal locations. A "feasible" path implies that a solution for the MAPF instance would meet the environmental constraints.

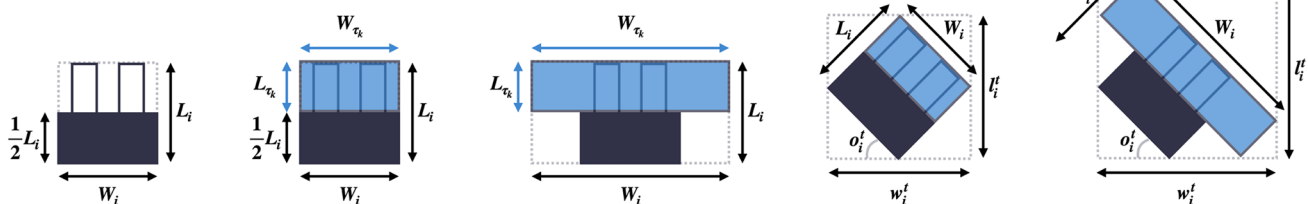


Fig. 1 Definition of agent orientation and size (including the material being carried)

Agents can return and stay at the parking locations for as long as they need at any time, in order to avoid conflicts (collisions) with other agents in the well-formed N-MAPD instances. With this return action, the number of agents in an excessively crowded environment can be reduced to mitigate congestion. As most real-world MAPD problems can be well-formed instances, including MAPD in a construction site, they belong to a realistic subclass of all MAPD instances. However, we need to discuss the condition (3') of our N-MAPD below.

Proposed Method

PAPO, which is our proposed algorithm for N-MAPD instances, generates collision-free plans, i.e., sequences of actions to reach destinations without conflict in non-uniform environments. In PAPO, the agent detects and resolves conflicts by utilizing and maintaining a shared SBI. The SBI comprises two tables, a *task execution status table* (TEST) and a *reservation table* (RT) (Silver [27]). A TEST is a set whose elements are tuples (τ, v, i) , wherein τ is the task currently being executed by agent i and v is the node of the i 's destination, which is the loading or unloading location specified by τ . Therefore, every time an agent with no task selects a task from the set of requested tasks, two tuples are stored in the TEST. The details of the TEST and the RT are explained later. The SBI is stored in centralized shared memory, and one agent can exclusively access this memory at a time, similar to a token in the TP method (Ma et al. [19]). Synchronized shared memory may become a bottleneck for performance, but we can assume that the movement of the robot is relatively slow; thus, the overhead time caused by mutual exclusion control can be ignored if the number of agents in an environment is not unrealistically high (e.g., less than 100 agents).

The PAPO algorithm comprises the following two phases. The first phase is TSAP, wherein the agent builds several short paths to its destination and then generates the set of action sequences along each path. Second, in the phase of CRCAS, the agent generates a conflict-free action sequence by observing the previously approved plans of other agents in the SBI to avoid conflicts.

TSAP

Agent $i \in A$ constructs the first $N_K \in \mathbb{Z}_+$ shortest path from its current location v_s^i to its destination v_d^i in the first stage of TSAP, where v_d^i is typically the loading, unloading, or parking location, depending on the progress of the performing task. Here, we formally define a *path* as a finite node sequence $r = \{v_0(= v_s^i), v_1, \dots\}$ where any pair of adjacent

nodes has an edge $(v_j, v_{j+1}) \in E$. The distance of path r is given by

$$r = \sum_{l=1}^{|r|} \text{dist}(v_{l-1}, v_l).$$

We use Yen's algorithm (Yen [38]) with the Dijkstra method among several algorithms for generating the first N_K shortest path. Notably, at this stage, agents only refer to the topological structure of $G = (V, E)$ and do not consider environmental constraints.

In the second stage, i generates the first $N_p \in \mathbb{Z}_+$ lowest-cost action sequences to move along the path $\forall r_k \in \{r_1, \dots, r_{N_K}\}$ obtained in the first stage without violating environmental constraints. Here, cost implies the duration to complete the action sequences and thus *wait* action is not included. We elaborate on the second stage in detail here. First, to build the action sequences along the path r_k , agent i generates the weighted state graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ (subscripts are omitted below) from G for the path r_k . Its nodes $\mathcal{V} \subset V \times \mathcal{D}$ is the set of *state nodes* $v = (v, o) \in \mathcal{V}$ and its edges $(\mu, v) \in \mathcal{E} (\mu, v \in \mathcal{V})$ is called the *transition edge* corresponding to action, *move* or *rotate*, for the transition from μ to v . We have also introduced the edge weight $\omega(\mu, v)$, which is the required duration of the corresponding action. We have denoted the search space by (v, t_v) using the time t_v when i will reach the state v , and then apply Yen's algorithm using A^* search to build the first N_p lowest-cost action sequences. Furthermore, using distance $l = \text{dist}(v_d^i, v_s^i)$ between v_d^i and the current node v_s^i and the difference θ between the orientation required by v_d^i and current orientation o_s^i , we define the heuristic function h for A^* search as

$$h(l, \theta) = T_{mo}(l) + T_{ro}(\theta). \tag{3}$$

Expectedly, the heuristic function h is admissible, as the costs of actions and environmental constraints are not considered. Furthermore, since the final node v_d^i is typically the loading or unloading node of the material, the required orientation at v_d^i is typically determined by the task of i . If v_d^i is the parking location, any orientation may be possible or restricted depending on the shape of the location.

In the second stage, agent i can prune several state nodes that violate the environmental constraints related to the size of the material being carried, path width, and its size. Suppose that i , whose state is $v = (v, o)$ at t_1 , schedules a state transition to $\mu = (u, o')$ at $t_2 = t_1 + \omega(v, \mu)$ by (v, μ) . Thereafter, if $l_u^i > L_u$ or $w_u^i > W_u$ is satisfied, μ will be pruned because the constraint between the node and agent sizes is violated. Similarly, if $W_{uv} < |l_u^i \sin d_u^i| + |w_u^i \cos d_u^i|$ is satisfied by direction d and orientation o , then *move*, (v, μ) (i.e., $(u, v) \in E$ and $o = o'$), is impossible; thus, μ after v is also pruned. Finally, if (v, μ) is *rotate* (i.e., $v = u$ but $o \neq o'$),

μ may also be pruned, since the insufficient size of v , L_v and/or W_v , make the corresponding action impossible as a violation of the rotation constraint. We can identify this situation by comparing L_v and W_v with l'_i and w'_i between t_1 and t_2 . For example, these values are maximum if the agent shape is square and the orientation of i is 45, 135, 225, or 315 (Fig. 1).

After TSAP, the ordered set of at most $N_K \cdot N_P$ action sequences that are sorted in ascending order by total duration, $P_i = \{p_1, \dots, p_{N_K \cdot N_P}\}$, is obtained. Expectedly, its first action sequence is the minimum cost plan and the best candidate. However, it might be selected due to conflicts with other plans. This kind of conflict is verified in the next phase, CRCAS. When the obtained plan p_i is generated along a path r_k , we denote this relationship as $r_k = r(p_i)$.

CRCAS

The purpose of CRCAS is that, by accessing the SBI, agent i selects the plan at the first of P_i and attempts to detect conflicts between it and the already approved plans being executed by other agents. Thereafter, i modifies the plan to resolve the detected conflicts, replaces it with the modified one, and sorts P_i again. Note that the basic policy for our strategy of planning is not to modify the already executed plans. If no conflict is detected in the first element of P_i , it is the result of CRCAS. The reservation data associated with the selected plan is then added to the SBI's RT and the plan is approved for execution. We have described the structure of the RT below.

Algorithm 1 Conflict resolution part of PAPO

```

1: //  $P_i$  is the list of the plan generated by TSAP( $N_K, N_P$ ).
2: function CRCAS( $P_i$ )
3:    $C_{max} \leftarrow$  Maximal duration of the plan in  $P_i$ 
4:   while true do
5:     if  $P_i = \emptyset$  then return false // no plan is found.
6:     end if
7:      $P_i$  is sorted by duration
8:      $p_1 \leftarrow$  the first (shortest duration) plan in  $P_i$ 
9:      $C_1 \leftarrow$  duration of  $p_1$  // Shortest duration in  $P_i$ 
10:     $c_1 \leftarrow \langle (i, j) [t'_s, t'_e], v' \rangle$  // The first conflict in  $p_1$  by comparing with
the entries in the RT.
11:    if  $c_1 = null$  then return  $p_1$  // Plan  $p_1$  will be executed.
12:    end if
13:     $C \leftarrow$  all conflicts occurring at  $v'$  // so  $c_1 \in C$ 
14:     $c_f \leftarrow$  final element  $\langle (i, k), [t_s, t_e], v' \rangle$  in  $C$  after being sorted by the
occurrence order.
15:     $u \leftarrow e_{v'}^k - s_{v'}^i + 1$  // where  $u$  is the wait length required to resolve  $c_f$ .
16:    if  $p_1$  includes wait at  $v'$  then
17:       $C_1 \leftarrow C_1 - u_{v'}$  // where  $u_{v'}$  is the wait length of wait at  $v'$ .
18:      Action wait at  $v'$  is removed from  $p_1$ .
19:    else
20:       $w(p_1) \leftarrow 0$  //  $w(p)$  may be defined for  $\forall p \in P_i$ .
21:    end if
22:     $w(p_1) \leftarrow \max\{w(p_1), u\}$ ,  $u_{max} \leftarrow w(p_1)$ ,  $C_1 \leftarrow C_1 + u_{max}$ 
23:    if  $C_1 \geq C_{max} + \beta$  then // where  $\beta$  is the tolerance parameter.
24:       $P_i \leftarrow P_i \setminus \{p_1\}$  // abandon  $p_1$ 
25:    end if
26:    wait( $u_{max}$ ) is inserted in  $p_1$  before reaching  $v'$  with the modification
strategy.
27:     $p_1$  in  $P_i$  is replaced with the modified  $p_1$ .
28:  end while
29: end function

```

We define conflict as a situation where the same node $v \in V$ is simultaneously occupied by multiple agents. If i starts *move* from v to the neighboring node u at time s_v^i , and the occupied intervals by i for v and u are denoted as $[s_v^i, e_v^i]$ and $[s_u^i, e_u^i]$, we assume $e_v^i = s_v^i + T_{mo}(\text{dist}(v, u))/2$ and $s_u^i = e_v^i$. In the same way, if i begins *rotate*, *load*, *unload* or *wait*, at time s_v^i on v , we can denote the occupied interval of the corresponding action by $[s_v^i, e_v^i = s_v^i + T_*]$, wherein T_* is the duration of the corresponding action. Additionally, we add a fixed margin $\lambda \geq 0$ to these intervals for safety, as every agent has a physical size. For example, s_v^i and e_v^i are changed using $s_v^i \leftarrow s_v^i - \lambda$ and $e_v^i \leftarrow e_v^i + \lambda$. Thereafter, i creates the following list related to the node occupancy from plan $p_k \in P_i$,

$$((v_0, [s_{v_0}, e_{v_0}], i), (v_1, [s_{v_1}, e_{v_1}], i), \dots, (v_d^i, [s_{v_d^i}, e_{v_d^i}], i)),$$

where $r(p_k) = \{v_0, v_1, \dots, v_d^i\}$ is the sorted set. This list is called the *occupancy list*.

We define that a conflict occurs at node v when two occupied intervals for v by two agents $i, j \in A$ have an intersection $[t_s, t_e]$. This conflict is represented by tuple $c = (\langle i, j \rangle, [t_s, t_e], v)$. RT in the SBI stores the occupancy lists of the approved plans of other agents. The plan of agent i may cause a conflict with the plan of another agent k , approved at the same node v , but since the plans stored in the SBI have already been approved, the intersection indicating a conflict will only appear between the two agents. Therefore, for two conflicts $c_1 = (\langle i, j \rangle, [t_s^1, t_e^1], v)$ and $c_2 = (\langle i, k \rangle, [t_s^2, t_e^2], v)$, they are always disjoint, i.e., $[t_s^1, t_e^1] \cap [t_s^2, t_e^2] = \emptyset$.

RT is the set of $(v, [s_v^i, e_v^i], i)$ in the occupancy list generated from the approved plans and has not yet expired. Therefore, the element is removed from RT when $e_v^i < t_c$, where t_c is the current time. Agent i stores all elements of the occupancy list in RT when i 's plan p is approved.

The pseudocode in Algorithm 1 is the overall flow of the CRCAS algorithm. During the execution of this function, we assume that an agent has exclusive access to RT in the SBI. First, agent i calculates the occupancy list for the first element p_1 in P_i generated by TSAP. Thereafter, according to the order of visiting nodes $r(p_1) = \{v_1, v_2, \dots\}$, i retrieves from RT a list whose first element is v_l and attempts to detect conflicts by comparing these lists. If no conflicts are detected, i will register p_1 , which is the result of CRCAS, as the approved plan into RT (Line 11). Let $c = (\langle i, j \rangle, [t_s^c, t_e^c], v')$ be the first conflict that is detected at v' in the visit order, where j is the agent with the approved plan that conflicts with p_1 and $[t_s^c, t_e^c]$ is the intersection of the time of staying at v' for both agents i and j . We define the set of all conflicts at v' as C since another conflict with another agent at v' can also occur. Subsequently, i sorts C in order of occurrence time and sets the last element of C as c_f (Line 14). Later, i inserts *wait* into p_1 using the strategy to modify plans as described

next to ensure that i arrives at v' after another agent k leaves v' (Lines 15, 22, and 26), where e_v^k is the leaving time of k from v' . Thus, i can avoid at least the detected conflicts.

However, the addition of *wait* may cause another conflict and therefore we have modified the preliminary PAPO (Yamauchi et al. [37]) to improve the overall efficiency. If *wait* has already been inserted at v' in p_1 , i eliminates it; otherwise, it initializes $w(p_1) = 0$, where $w(p_1)$ ($= u_{max}$) is the maximal length of *wait* required to resolve the conflicts in p_1 . Thereafter, i adds *wait*(u_{max}) because u_{max} is the smaller but sufficient waiting length required to resolve c_f ; thus, i can prevent an unnecessarily long wait. However, with the addition of *wait*, p_1 is abandoned if the duration of the modified plan is too large for the implementation (Lines 23 and 24). Note that β is the *tolerance parameter* for retaining the selected plan. Next, i sorts P_i again in ascending order by duration and repeats the same operation until the first element of P_i , p_1 , contains no conflict.

This process will eventually stop; otherwise, if it continues forever, the duration of plan implementation becomes larger than $C_{max} + \beta$, the plan is eliminated from P_i , and eventually, P_i will become empty. If P_i is empty, i.e., a collision-free plan cannot be generated from the P_i using the parameters β , N_K , and N_p , then the function CRCAS returns a *false* value. In this case, agent i invokes CRCAS again with a relaxed condition, i.e., using P_i that is generated by increasing the values of β , N_K , and N_p .

Strategy to Modify Plan

Agent i has to determine where to insert *wait*(u_{max}) before the node v' , where the first conflict is detected in plan p_i . Let $r(p_i) = \{v_0, \dots, v_l (= v'), \dots, v_n (= v_d^i)\}$ denote the sequence of visiting node. Agent i can add *wait*(u_{max}) just before leaving any node v_k ($0 \leq \forall k \leq l - 1$) before v_l . For example, it can be added to v_{l-1} , but another conflict may occur somewhere due to this modification. In particular, if a probability of a new conflict at v_{l-1} is expected, a further *wait* before v_{l-1} is needed to avoid it, resulting in a cascade of conflicts that leads to significant performance reduction. Furthermore, the required waiting time is likely to be longer because of the basic policy of CRCAS (not to modify already executed plans). Conversely, if i inserts *wait*(u_{max}) when or before i leaves v_0 , i can avoid conflicts without failure. This is because, according to the well-formed N-MAPD condition (3'), other agents do not pass through node v_0 , and the loading and unloading nodes of the current task τ are already stored in the TEST when i selects it in the task selection process described below. However, this strategy implies locking the actions of other agents for a while, which might reduce the entire performance. This issue may require further discussion, but we have tentatively adopted the strategy of adding *wait*(u_{max}) to v_k , where $k = \max\{l - 3, 0\}$.

Algorithm 2 Task selection

```

1: function SELECTTASK( $i$ ) // Called from  $i \in A$ 
2:    $\mathcal{T}' \leftarrow \{\tau \in \mathcal{T} \mid v_\tau^{ld}$  and  $v_\tau^{ud}$  do not contain in the TEST in the current SBI.}
3:   // Note that  $\tau = (v_\tau^{ld}, v_\tau^{ul}, W_\tau, L_\tau, \phi_\tau)$ ,
4:   //  $v_\tau^{ld} = (v_\tau^{ld}, o_\tau^{ld})$  and  $v_\tau^{ul} = (v_\tau^{ul}, o_\tau^{ul})$ 
5:   if  $\mathcal{T}' \neq \emptyset$  then
6:      $\tau^* \leftarrow \arg \min_{\tau \in \mathcal{T}'} h(l, \theta)$  //  $h$ : heuristic function (Formula 3)
7:      $\mathcal{T} \leftarrow \mathcal{T} \setminus \tau^*$ ; return  $\tau^*$ 
8:   else return false
9:   end if
10: end function

```

Algorithm 3 Path and action planning with orientation

```

1: function PAPO( $i, v_s, v_d$ ) // Called from  $i \in A$ 
2:   //  $v_d$ :  $i$ 's destination and  $v_s$ :  $i$ 's current location
3:   while true do
4:      $P_i \leftarrow$  TSAP( $N_K, N_P$ )
5:      $p \leftarrow$  CRCAS( $P_i$ )
6:     if  $p \neq \text{false}$  then return  $p$ 
7:     end if
8:      $(\beta, N_K, N_P) \leftarrow$  RELAXPARAM( $\beta, N_K, N_P$ )
9:   end while
10: end function

```

Task Selection and Process of PAPO

Agent i returns to its parking location $park_i$ if \mathcal{T} is empty; otherwise, i performs the process for task selection. The outline of the task selection process in agent i is demonstrated in Algorithm 2. During this process, i has exclusive access to the TEST in the SBI. From condition (3'), i focuses only on task $\tau = (v_\tau^{ld}, v_\tau^{ul}, W_\tau, L_\tau, \phi_\tau)$ where loading node v_τ^{ld} and unloading node v_τ^{ul} do not contain in the TEST. Thereafter, i chooses the task τ^* with the smallest value of the heuristic function $h(l, \theta)$ in the A^* -search (Formula 3), where l is the distance between the loading node v_τ^{ld} and the current node v_s^i ($l = \text{dist}(v_\tau^{ld}, v_s^i)$). If there is no such task (Line 8), agent i returns to $park_i$ and remains there for a short while. Later, if another agent completes its task while i stays there, it is probable that i can select a task that meets condition (3') and will leave $park_i$.

Agent i calls the function PAPO to generate a collision-free action sequence after selecting task τ_i . The pseudocode of PAPO is presented in Algorithm 3. Let us consider

that v_s is the current node and v_d is the current destination, which is one of v_τ^{ld} , v_τ^{ul} , and $park_i$, depending on the task progress. First, if i succeeds in generating plan p using the CRCAS and TSAP processes, i begins to move according to p and removes $(\tau, v_d, i) \in \mathcal{T} \times V \times A$ from the TEST. If not, i invokes the *relaxation function* RELAXPARAM, which modifies the parameter values β , N_K , and N_P to relax the conditions for planning and then calls the planning process again. However, if i has already called RELAXPARAM several times to relax the parameter values, the function PAPO gives up the execution of τ_i (i.e. τ_i is restored to \mathcal{T}) and sets $v_d = park_i$ to make i return to the parking location. This situation is likely to occur when the number of agents is very large compared to the current environment size.

There are several strategies to determine the initial values of β , N_K , and N_P and how we increment these values. Smaller values of β , N_K , and N_P may generate more effective plans but may fail in generating a plan. Therefore, initially, it is better to set these parameters to small values and then gradually increase them if the agent cannot generate a plan

without conflicts. Continuing to increase the parameter values results in the collision-free plan for i (i.e., completeness is guaranteed) since the constraints are gradually removed as the approved plan is executed by other agents and the occupancy list in the RT is expired over time. Of course, we also have to consider that the frequent increments of these parameter values will lead to inefficient planning.

Properties of PAPO

We analyze the properties of PAPO from the viewpoints of soundness, completeness, and complexity bounds. First of all, the soundness of PAPO is trivial from Algorithms 1, 2, and 3 because CRCAS returns the conflict-free path based on one of the basic paths, which are generated by TSAP, from the current location to the destinations for the given task without changing the sequence order of the nodes that pass through.

For the completeness of PAPO, we first note that the well-formed N-MAPD condition (3') guarantees the existence of at least one solution in a N-MAPD instance that satisfies the required environmental constraints. It is obvious that this solution consists of the finite length of action sequence. Then, TSAP in PAPO generates the first N_K shortest path and the first N_P lowest-cost action sequences in path planning and then action planning (sequences of actions) to reach the destination using Yen's algorithm. Because the values of the parameters β , N_K , and N_P are gradually increased by the relaxation function RELAXPARAM, and TSAP with Yen's algorithm will output the paths that follow the nodes appearing in the solution and CRCAS can find the collision-free action sequence of the solution. Moreover, as aforementioned in "Task selection and process of PAPO", the parameter β is continually increasing, so that the constraints are gradually removed as the approved plan is executed by other agents and the occupancy list in the RT is expired over time. The well-formed N-MAPD conditions and task selection based on them allow the agent to stay at its start location for any finite amount of time so that even in the worst case, the agent can generate a collision-free plan which moves after all other agents have completed their movements. Therefore, the completeness of PAPO is guaranteed.

Finally, we consider the complexity bounds of PAPO per collision-free action sequence generation of one agent. Since PAPO eventually covers all solutions by continually increasing the values of the parameters N_K and N_P via RELAXPARAM, the complexity bounds is $\mathcal{O}(|V \times \mathcal{D}|^2)$ in the worst case, as well as naive search in conventional studies. However, because PAPO explores the search space for the N-MAPD problem via path planning, action planning, and

conflict resolution in a phased manner, PAPO can significantly improve planning efficiency compared to naive search in many cases.

Experimental Evaluation and Discussion

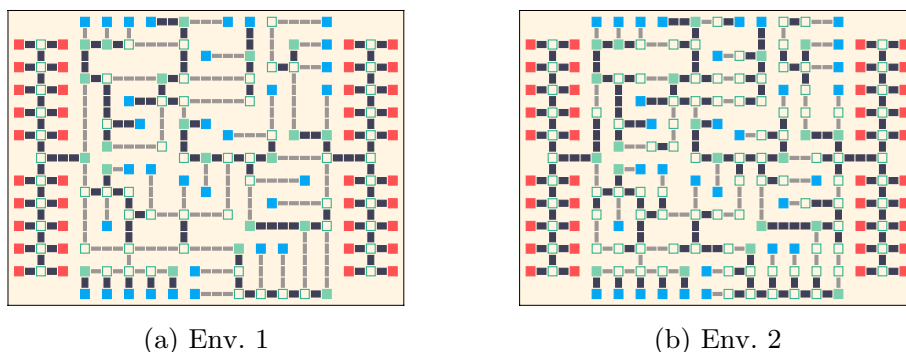
Experimental Environment and Setting

We conducted many experiments to evaluate the performance of our proposed method, PAPO, using several N-MAPD instances. Agents carry either small or large materials that are specified in each N-MAPD task $\tau \in \mathcal{T}$. We set the equal numbers of small and large materials to $N/2$. The width and length of a large material are $W_\tau = 1.0$ and $L_\tau = 0.25$ and those of a small material ϕ_τ are $W_\tau = 0.5$ and $L_\tau = 0.25$, respectively. All agents have the same size that is specified by $W_i = 0.5$, $L_i = 0.5$. The ratio of the agent's length to the length of its fork part is set to $\gamma = 0.5$.

For our experiments, we have prepared two environments, Env. 1 and Env. 2, which have maze-like structures inspired by the construction sites with obstacles such as spaces for other work, columns, and walls; thus, they have environmental constraints such as paths widths and nodes sizes. Env. 1 is a maze-like environment wherein the nodes are set at the ends of edges where agents may load and unload materials, and at intersections (Fig. 2a). Nodes are divided into two types: large nodes v with width $W_v = 1.5$ and length $L_v = 1.5$ and small nodes v with width $W_v = 1.0$ and length $L_v = 1.0$. These nodes are indicated as hollow green squares (large nodes) and green-filled squares (small nodes) in Fig. 2. Similarly, edges are divided into two types: wide edges (u, v) with width $W_{uv} = 1.0$, depicted by bold black edges, and narrow edges (u, v) with width $W_{uv} = 0.5$, depicted by gray edges. The breaks in an edge indicate a length of 1 per one block. We assume that agents can only wait and rotate at nodes, but agents with large materials cannot rotate on small nodes and may have to rotate before passing through narrow edges.

Env. 2 has essentially the same topology as that of Env. 1; however, more nodes are added on some long edges, and so it has more nodes that are not at the intersection. Typically, the narrow and wide edges are connected by these nodes, and agents carrying large materials may need to rotate at this node to pass through the narrow edge (Fig. 2b). Rotation at an intersection requires some time; thus, it might block the movement of other agents. Initially, a hundred tasks \mathcal{T} are generated, and the pickup and delivery locations of each task in \mathcal{T} are randomly selected from the blue squares. Furthermore, we randomly allocate the initial position of each agent to its parking location.

Fig. 2 Two experimental environments (blue cells: pickup and delivery locations, red cells: parking locations, hollow green cells: large nodes, green-filled cells: small nodes, black cells: wide edges, and gray cells: narrow edges)



We measured four performance indicators, the *success rate* for all runs, the *total planning time* (we call it the *planning time*) for all tasks in \mathcal{T} , *makespan*, i.e., the required time to complete all tasks in \mathcal{T} , and *operational time per task*, which is the average time to complete a task (we call it the *operational time*), of our proposed method, and the comparative methods. Success rate is mainly used for meta-heuristic optimization methods. Planning time is used for the evaluation of planning efficiency, makespan is used for the evaluation of transportation efficiency, and operational time is used for the evaluation of generated plan quality. Table 1 presents the other parameter values used in our experiments. Our experiments were conducted on a machine with the following specifications: 3.20-GHz Intel 16-Core Xeon W with 112-GB RAM. The following experimental results are the averages of 100 independent runs using 100 different random seeds. Note that the experimental results are plotted as the average of all successful runs, except for the success rate.

Performance Comparison — Exp. 1

We have compared the performance of the proposed method, PAPO, with those of preliminary PAPO (Yamauchi et al. [37]), the naive centralized method (Yamauchi et al. [37]), which will be called NAIVE henceforth, and ACO and SA as the meta-heuristic algorithms, i.e., general incomplete optimization methods in Env. 1 and Env. 2. NAIVE is a centralized planner that assumes that the nodes reserved

by the plans of other agents are temporary obstacles for a certain time interval and generates an optimal (shortest) and collision-free action sequence whose elements are $\{load, unload, move, rotate, wait\}$ in turn. Therefore, each state is denoted by tuple $v = (v, o, t)$ of node v , agent orientation o , and time t when the agent reaches node v with orientation o . We describe the ACO and SA algorithms for the N-MAPD in Appendix B. The values of the parameters for ACO are $\mathcal{N} = 10$, $I = 25$, $S = 1000$, $\rho = 0.9$, $\psi = 0.2$, and $\delta = 0.8$, and those for SA are $T_{ini} = 10^{-4}$, $T_{ter} = 10^{-7}$, $\alpha = 0.97$, $\zeta = 50$, and $S = 1000$. We set these parameter values so that the planning times of ACO and SA were less than or equal to that of NAIVE, and the number of iterations of ACO and SA per agent's plan were almost the same.

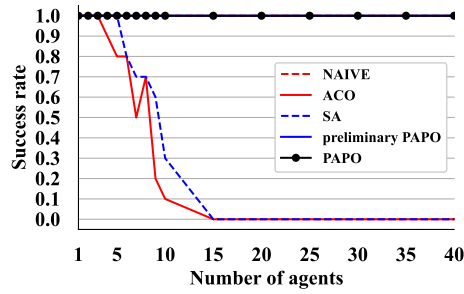
All durations of actions in preliminary PAPO, NAIVE, ACO, and SA are identical to those in PAPO (Table 1). Under the assumption that the plans already in execution will not be modified, NAIVE generates an optimal plan. The initial values of the parameters for PAPO and preliminary PAPO are $\beta = 100$, $N_K = 3$, and $N_p = 3$, and for each call to the relaxation function RELAXPARAM, the values of the parameters increase to $\beta \leftarrow \beta * 2.0$, $N_K \leftarrow N_K + 1$, and $N_p \leftarrow N_p$. We have plotted the experimental results of Env. 1 in Fig. 3 and the results of Env. 2 in Fig. 4. In Figs. 3b and 4b, the makespan for NAIVE, preliminary PAPO, and PAPO are too small to be seen graphically. Therefore, we have plotted only the makespan for NAIVE, preliminary PAPO, and PAPO in Figs. 3c and 4c. We have also plotted the operational time and planning time only for some of the methods in Figs. 3e, 3g, 4e, and 4g to improve visibility. Because the completion of all tasks using NAIVE, ACO, and SA were very time-consuming, those results are an average of 10 runs.

First, we have compared the performance of the meta-heuristic optimization methods, ACO and SA, with those of other methods in N-MAPD. Figures 3a and 4a indicate that the success rate of NAIVE, preliminary PAPO, and our PAPO was always 1.0 for both Envs. 1 and 2, regardless of the number of agents. However, the success rates of ACO and SA decreased with increasing number of agents for both Envs. 1 and 2, falling to 0.0 after $M \geq 15$ (ACO and SA)

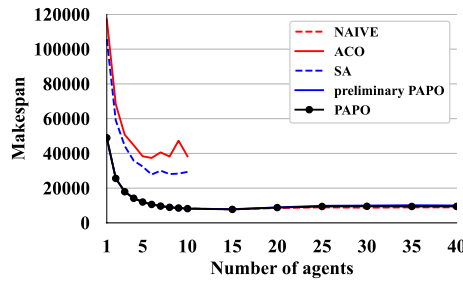
Table 1 Parameter values used in the experiments

Description	Parameter	Value
No. of agents	M	1 to 40
No. of tasks	N	100
Orientation/direction increments	D	90
Duration of <i>move</i> per length 1	$T_{mo}(1)$	10
Duration of <i>rotate</i>	$T_{ro}(D)$	20
Durations of <i>load</i> and <i>unload</i>	T_{ld}, T_{ul}	20
Duration of <i>wait</i>	$T_{wa}(t)$	t
Safety intervals	λ	5

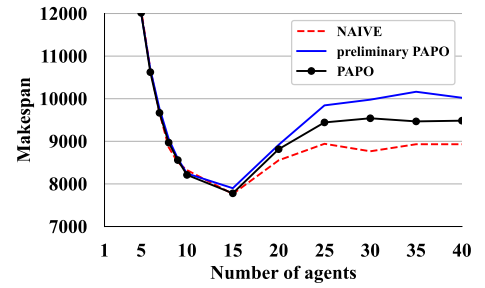
Fig. 3 Performance comparison for Env. 1



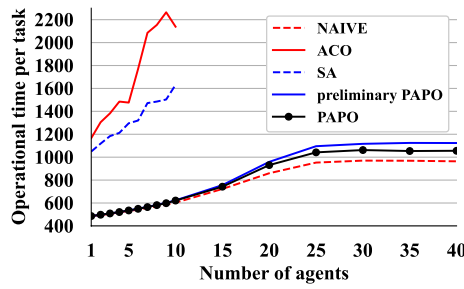
(a) Success rate



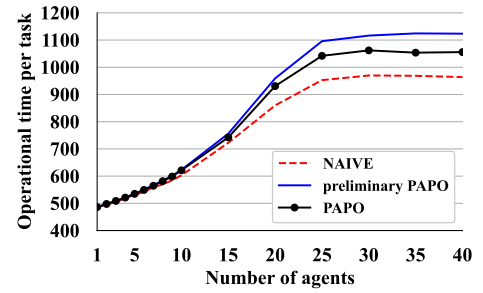
(b) Makespan



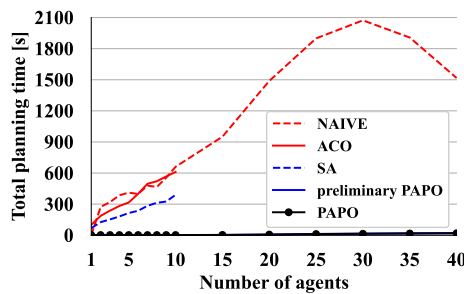
(c) Makespan



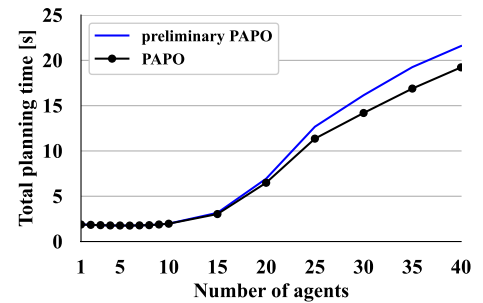
(d) Operational time



(e) Operational time



(f) Planning time



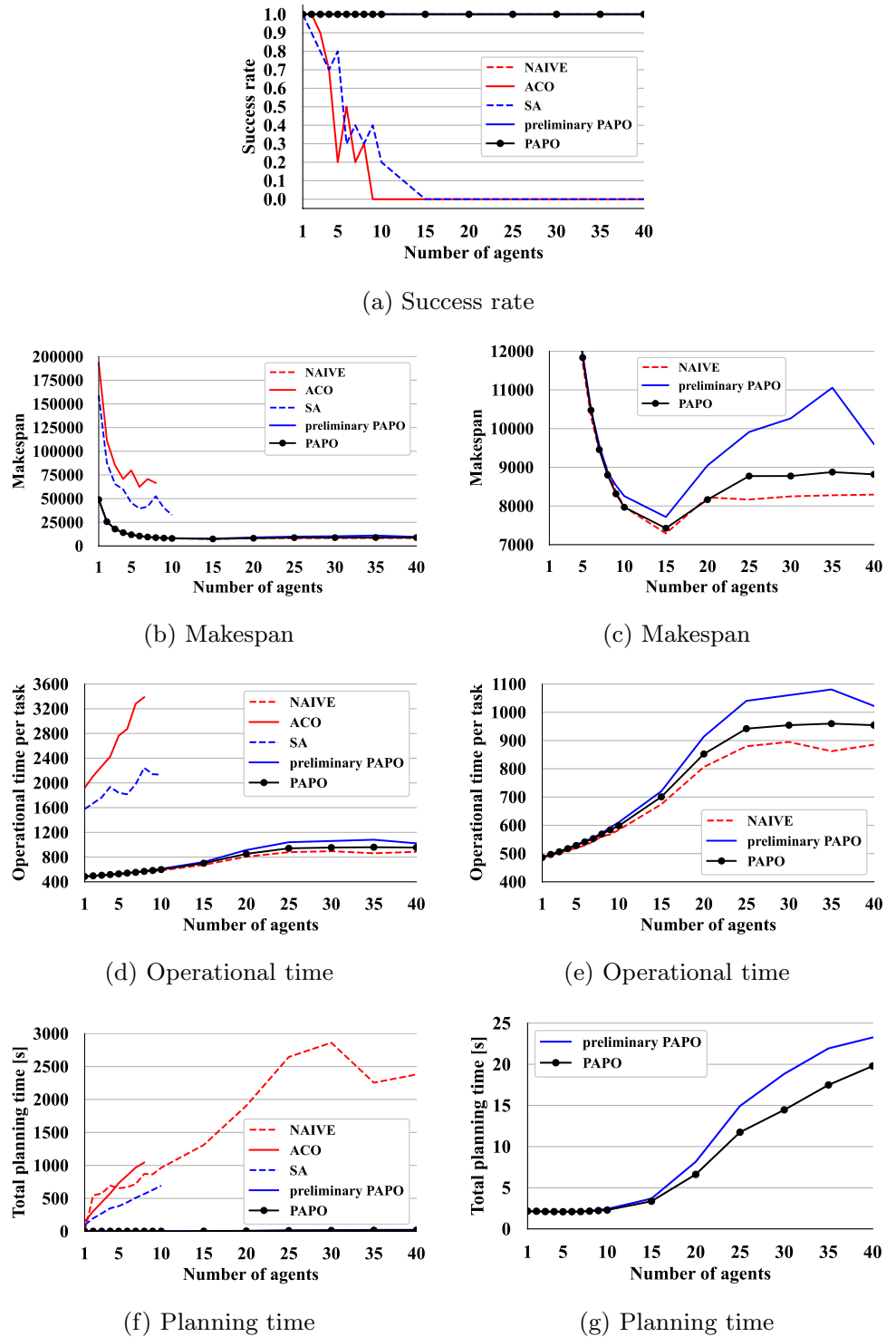
(g) Planning time

in Env. 1, and $M \geq 9$ (ACO) or $M \geq 15$ (SA) in Env. 2. In ACO and SA, there was no guarantee that agents continued to wait for conflict avoidance at the planning start location, including the unique parking location of each agent because they selected their next transition state probabilistically. Therefore, even if the environment met the conditions for

well-formed N-MAPD, ACO and SA could not necessarily solve the N-MAPD instance.

Figures 3b, 4b, 3d, and 4d indicate that the quality of the generated plans by ACO and SA was lower than that of the other methods, and that the transportation efficiencies were also poorer. This is because the N-MAPD problem

Fig. 4 Performance comparison for Env. 2



adds orientation to agent states to account for environmental constraints, so the number of states is enormous, and probabilistic state transitions in ACO and SA may cause agents to redundantly transition to the same state. This phenomenon is also supported by a performance comparison of Envs. 1 and 2. As we will be discussed in more detail below, the makespan and operational time tend to decrease in Env. 2

compared to Env. 1 in NAIVE, preliminary PAPO, and our PAPO. In contrast, the makespan and operational time in Env. 2 increased more than those in Env. 1 for ACO and SA. This is because Env. 2 added more nodes to some of the longer edges of Env. 1 to reduce the occurrence of conflicts, especially at intersections, and to improve transportation efficiency. However, the number of states was further

increased, resulting in an increased probability of agents making unnecessary state transitions in ACO and SA.

We may be able to improve the quality of the generated plans and the transportation efficiency of ACO and SA by further increasing the parameter value of the number of iterations. However, because there is a tradeoff between solution quality and planning efficiency, from Figs. 3f and 4f, any further deterioration in planning efficiency is impractical for use in an actual environment. Finally, we have compared the results of ACO and SA and found that the performance of SA outperformed ACO in all evaluation indicators for Envs. 1 and 2. This is because when generating the initial solution, the agent uses heuristic values in SA, but in ACO there are no pheromones, so the ants make state transitions completely at random. With completely random transitions, it is not easy to reach the goal state in an N-MAPD problem with a large number of states, so the improvement in the quality of the plan through iterations was limited.

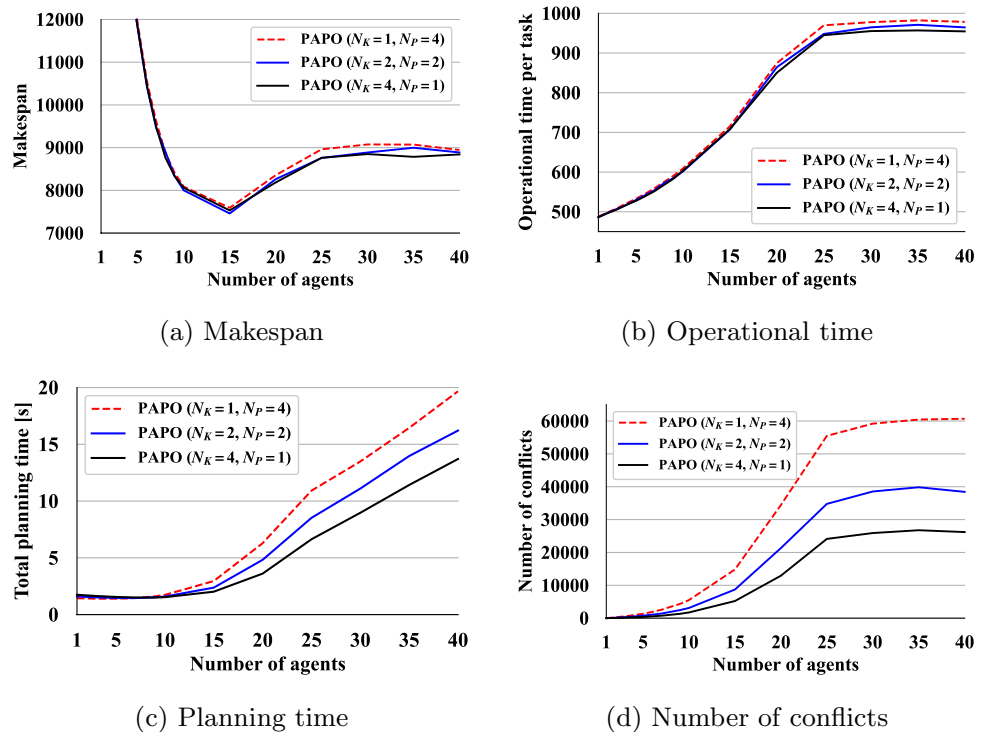
We can observe from Figs. 3c and 4c, Figs. 3e and 4e that the makespan and operational time of PAPO increased by approximately 7% compared to those of NAIVE, while from Figs. 3f and 4f, the planning time of PAPO in both Envs. 1 and 2 was significantly reduced compared to that of NAIVE. The quality of the solution obtained using NAIVE was very high because it could generate an optimal plan. However, the planning time was immense, making its use impractical in an actual environment. In contrast, PAPO can efficiently

generate sub-optimal but acceptable paths. We suppose that the transportation efficiency of PAPO can be improved by adjusting the parameter values, which we have comprehensively discussed in the next section.

Subsequently, for further elucidation, we would like to compare the improvement of PAPO against preliminary PAPO. Figures. 3 and 4 indicate that PAPO generated plans closer to the optimal solution obtained using NAIVE, in terms of the makespan and operational time, compared to those generated by preliminary PAPO. Furthermore, it reduces the planning time compared to preliminary PAPO. This is the contribution of the improved method of carefully adding *wait* by determining the short wait time to resolve a conflict as described in Sects. 4.2 and 4.3. Therefore, it resulted in the improvement in both overall transportation and planning efficiency. Actually, the number of conflicts detected in Env. 1 was 86086.4 for preliminary PAPO and 75545.97 for PAPO when $M = 25$, reducing the occurrence of conflicts by approximately 12%.

Comparing the results of Env. 1 and Env. 2, the planning time of PAPO in Env. 2 was almost identical to that in Env. 1; however, the makespan and operational time were smaller than those in Env. 1. This is owing to the additional nodes to some edges in Env. 2. Agents will stay at the current node when executing *rotate* and *wait*; therefore, if they execute these actions at intersection nodes, they may obstruct the movement of other agents for a while. However, they

Fig. 5 Impact of the combinations of the initial values of N_K and N_p



could reduce the number of such obstructive states by adding nodes where they could rotate and wait. The PAPO with 25 agents ($M = 25$) realized approximately 24% fewer possible conflicts 75545.97 in Env. 1 and 57466.89 in Env. 2. This led to smoother movement of all agents. Moreover, in Env. 2 when $M = 25$, for example, the planning time only increased by approximately 0.4 s (3%), while the makespan and operational time were improved by approximately 670 (7%) and 100 (10%), respectively. Note that $T_{mo}(1) = 10$; thus, an improvement of 100 in the operational time corresponds to a reduction of 10 blocks of movement per task. This is a considerable improvement in performance.

When 25 agents used NAIVE ($M = 25$), the makespan was approximately 775 (9%) shorter (Figs. 3c and 4c) and the operational time was approximately 73 (8%) shorter (Figs. 3e and 4e) in Env. 2 than those in Env. 1, while the planning time increased by approximately 746 s (39%) (Figs. 3f and 4f). Even if there were a few agents, such as $M = 2$ and 3, the planning time increased significantly when using NAIVE. This implies that an increase in the number of nodes has a significant impact on planning efficiency. Since our N-MAPD problem deals with a three-dimensional search space comprising temporal, spatial position, and orientation, the computational cost rapidly becomes high if the environments have more nodes. In contrast, we can say that PAPO has sufficient robustness against the increase in the number of nodes

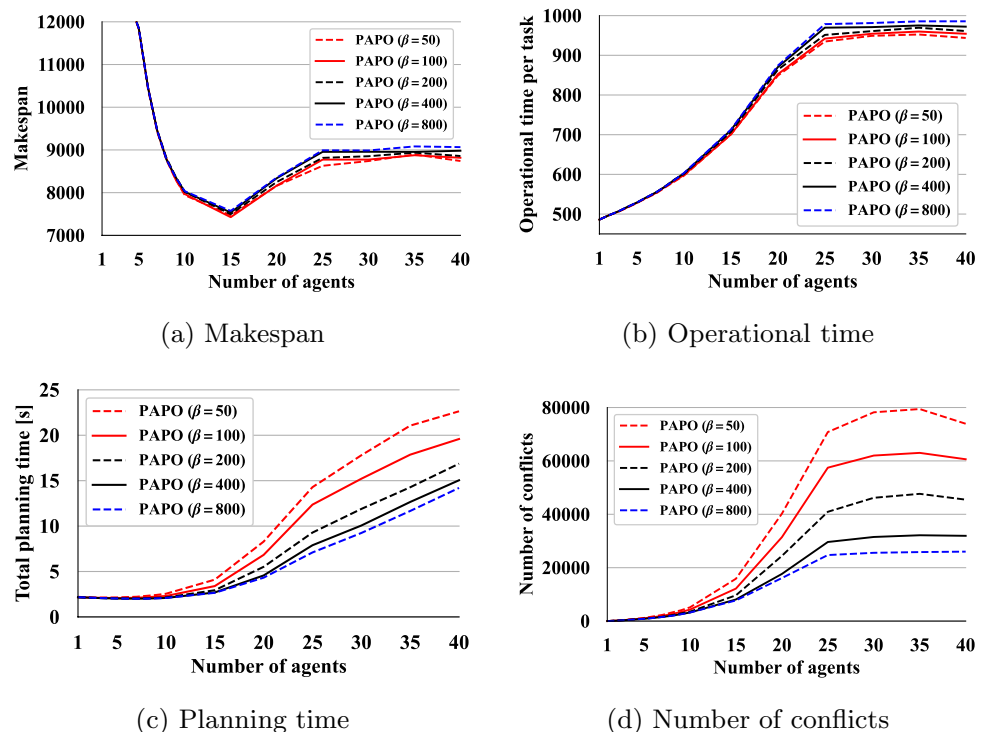
because PAPO could improve the makespan and operational time with quite a small increase in the planning time.

Characteristics of PAPO — Exp. 2

The purpose of the second experiment (Exp. 2) is to investigate how the initial values of the three parameters β , N_K , and N_P affect the performance of PAPO. First, we measured the performance when using the different combinations of initial values for (N_K, N_P) whose multiplications $N_K \times N_P$ are identical, such as (1, 4), (2, 2), and (4, 1). Note that the relaxation function RELAXPARAM was identical to the one used in Exp. 1, except that the initial value of $\beta = 100$ was fixed.

Figure 5 presents the plots of makespan, operational time, planning time, and the number of conflicts detected. From these results, we observe that the best combination of (N_K, N_P) was (4, 1) in Env. 2. This reveals that parameter N_K has more influence on performance compared to parameter N_P . When the agent is carrying no materials or small materials, it is not under any environmental constraint even in Env. 2; therefore, the only conflicts with the plans of other agents are subject to investigation. Hence, if agents generate more paths, they can prevent conflicts from occurring. Clearly, if we fix the parameter value N_P , the larger the N_K , the smaller the makespan and operational time.

Fig. 6 Impact of the initial value of β



The agent may not be able to generate a detour to the destination even if N_K is large. Perhaps, in specially designed environments, such as automated warehouses, it makes sense to set N_K to be large because there are many detours to the destination. However, such parameter setting is not always appropriate because the detours are limited depending on the environmental structure. As our experimental environment, inspired from a construction site, is a maze-like environment with limited detours, we set $N_K = 4$, which is not excessively large, to allow the agent to generate N_K different paths. Therefore, although there is an upper limit of N_K that depends on the environmental structure, a larger N_K could avoid the occurrence of conflicts and contribute to the reduction in makespan and operational time. Meanwhile, the agent can generate a variety of action sequences along each path generated in the first stage of TSAP by simply changing its rotation timing, without taking the long waiting time or detours if we set a large N_p . Therefore, by setting a larger N_p , it is possible to obtain a shorter collision-free action sequence. Note that this is dependent on the number of nodes that agents can rotate/wait and the number of detours.

Moreover, we conducted similar experiments with initial values of β by setting to 50, 100, 200, 400, and 800; however, the initial values $N_p = 3$ and $N_K = 3$ were fixed. We used the same relaxation function RELAXPARAM as in Exp. 1. It can be observed from Fig. 6 that the number of conflicts and the planning time decrease as the value of β increases. As β increases, the agents attempt to resolve conflicts by adding more *wait* to the candidate action sequences generated by TSAP; therefore, the frequency of calls to the relaxation function RELAXPARAM decreases. In Env. 2, the agent is likely to wait at the start node of a plan (i.e., the pickup or delivery location) or at the node that is added for Env. 2 before the intersection at which conflicts may occur because the wait is inserted before the node where the conflict is detected. As we described in Exp. 1, waiting at a node that is not an intersection reduces the possibility of interfering with the actions of other agents. Therefore, agents could reduce the occurrence of conflicts, resulting in reduced planning time. However, the excessive use of the synchronization strategy tends to increase the makespan and operational time. Thus, we can weigh the importance of the planning efficiency (planning time), the transportation efficiency (makespan), and the quality of the generated plans (operational time) by changing the initial value of β .

Discussion

From these experiments, we found that the proposed method could significantly improve planning efficiency compared to NAIVE and transportation and planning efficiency compared to preliminary PAPO, ACO, and SA. In our N-MAPD

problem, which deals with a three-dimensional search space of temporal, spatial, and orientation, the computational cost of obtaining a solution is very high, especially when the number of nodes in the environment is large. However, unlike NAIVE, which naively explores the search space, PAPO can efficiently explore the search space by performing path planning, action planning, and conflict resolution in a phased manner; therefore, very high planning efficiency was achieved. Furthermore, if we observe the results in Figs. 3 and 4 when the number of agents $M = 15$ and the makespan is minimum, PAPO achieved almost the same makespan and operational time as those of NAIVE, with a significant reduction in planning time. Therefore, depending on the structure and size of the environment and the number of agents, PAPO can achieve similar high transportation efficiency as that of NAIVE, which obtain the optimal solution, in situations where excessive congestion does not occur.

Moreover, unlike preliminary PAPO, PAPO prevents excessively long waits by carefully adding waits to the plan. This not only improves the quality of the generated plans for the agent in the planning process but also has a positive impact on subsequent planning for other agents. This implies that if an agent resolves conflicts with a minimum waiting time, it can prevent the occurrence of conflicts during the planning of other agents subsequently, and the increase in the waiting time to resolve them, in advance. As a result, PAPO could better improve the overall performance compared to preliminary PAPO.

Conclusion

We first introduced a model, the *MAPD in non-uniform environment* (N-MAPD) problem, inspired by our target applications at construction sites. An N-MAPD problem is quite complicated because the orientations and actions of agents are strictly restricted by the sizes of agents, materials, and nodes (locations), as well as the width of paths; therefore, it requires a high computational cost to generate optimal paths. Thereafter, we developed a planning method, called *path and action planning with orientation* (PAPO), which can efficiently build a non-optimal but acceptable collision-free plan for N-MAPD problems. The proposed method was compared with a naive centralized method, meta-heuristic optimization methods, and preliminary PAPO, our previous version of the PAPO algorithm (Yamauchi et al. [37]), and evaluated. Experimentally, we demonstrated that the proposed method can efficiently generate sub-optimal paths of moderate lengths for real-world applications, and that it has sufficient robustness to the increase in the number of nodes. Furthermore, by analyzing the characteristics of the proposed method, we observed that it can determine the weight

of the planning and transportation efficiencies and the quality of the generated plans by appropriately setting the initial values of parameters.

To further improve transportation efficiency, we plan to study the relaxation of the well-formed N-MAPD condition (3'). Because of condition (3'), agents cannot traverse any other start/goal locations in the N-MAPD problem; therefore, the upper bound on the number of agents that can simultaneously execute tasks in an environment depends on the number of nodes that can be the pickup and delivery locations of tasks. Furthermore, real-world transport robot agents occasionally need to recharge or replace their batteries. Therefore, we need to consider that agents have limited-capacity batteries (i.e., time limit for movement) in future

N-MAPD studies, as in the studies on the *multi-agent cooperative patrolling problem* (Yoneda et al. [39]; Sugiyama et al. [28]). Finally, our proposed method utilizes and maintains a shared SBI to detect and resolve conflicts, similar to a token in the TP method (Ma et al. [19]) which can be easily extended to a fully distributed version; a more detailed study of a fully distributed design is planned for future work.

Appendix A Summary of Notations

To enhance the readability, we list a summary of notations in Tables 2 and 3.

Table 2 Summary of notations used in this paper (No. 1)

Notations	Meanings
A	Set of M agents i , $A = \{1, \dots, M\}$
T	Set of N tasks τ_j , $T = \{\tau_1, \dots, \tau_N\}$
V, E	Set of nodes v and set of edges (u, v) , $u, v \in V$
G	Undirected connected graph, $G = (V, E)$
W_v, L_v	Width and length of the node v
W_{uv}	Width of the edge (u, v)
$\text{dist}(u, v)$	Length between the centers of u and v in the Euclidean space
t	Discrete-time, $t \in \mathbb{Z}_+$ (where \mathbb{Z}_+ is the set of non-negative integers)
d_i^t, o_i^t	(Moving) direction and orientation of agent i at time t
D	Orientation/direction increments
\mathcal{D}	Set of possible orientations
W_i, L_i	Width and length (or depth) of agent i
w_i^t, l_i^t	x -axis length and y -axis length of agent i at time t
γ	Ratio of the agent's length to the length of its fork part
$T_{ro}(\theta)$	Duration of <i>rotate</i> per rotation angle $\theta \in \mathbb{Z}_+$
$T_{mo}(l)$	Duration of <i>move</i> per length $l = \text{dist}(u, v)$
T_{ld}, T_{ul}	Durations of <i>load</i> and <i>unload</i>
$T_{wa}(t)$	Duration of <i>wait</i> per waiting time t
$park_i$	Parking location of agent i , $park_i \in V$
ϕ_{τ_j}	Material that is requested to carry by task τ_j
$v_{\tau_j}^{ld}, o_{\tau_j}^{ld}$	Location and orientation of the loading material ϕ_{τ_j}
$v_{\tau_j}^{ul}, o_{\tau_j}^{ul}$	Location and orientation of the unloading material ϕ_{τ_j}
W_{τ_j}, L_{τ_j}	Width and length of material ϕ_{τ_j}
τ_j	Task specified by $(v_{\tau_j}^{ld}, v_{\tau_j}^{ul}, W_{\tau_j}, L_{\tau_j}, \phi_{\tau_j})$, where $v_{\tau_j}^{ld} = (v_{\tau_j}^{ld}, o_{\tau_j}^{ld})$, $v_{\tau_j}^{ul} = (v_{\tau_j}^{ul}, o_{\tau_j}^{ul})$
(τ, v, i)	Task data being executed by agent i , where v is the i 's destination specified by task τ
v_s^i, o_s^i	Current location and orientation of agent i
v_d^i	Destination of agent i
N_K	No. of shortest path from v_s^i to v_d^i
r	Path, which is a finite node sequence $r = \{v_0(= v_s^i), v_1, \dots\}$
N_p	No. of lowest-cost action sequences to move along the path $\forall r_k \in \{r_1, \dots, r_{N_k}\}$
$\mathcal{V}_i, \mathcal{E}_i$	Set of state nodes $v = (v, o)$ and set of transition edges (μ, v) , $\mu, v \in \mathcal{V}_i$
\mathcal{G}_i	Weighted state graph generated from G for the path r_k by agent i , $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$
$\omega(\mu, v)$	Transition edge weight, which is the required duration of the corresponding action
t_v	Time when agent i will reach the state v
(v, t_v)	Search space, where t_v is the time when agent i will reach the state v

Table 3 Summary of notations used in this paper (No. 2)

Notations	Meanings
$h(l, \theta)$	Heuristic function using distance l and angle θ for A* search
P_i	Ordered set of at most $N_K \cdot N_P$ action sequences, $P_i = \{p_1, \dots, p_{N_K \cdot N_P}\}$
$r_k = r(p_i)$	Relationship that the obtained plan p_i is generated along a path r_k
$[s_v^i, e_v^i]$	Occupied intervals from time s_v^i to e_v^i of node v by agent i
λ	Safety intervals due to the physical size of agent
$[t_s, t_e]$	Intersection of occupied intervals by two agents
c	Conflict of intersection $[t_s, t_e]$ at node v between agents i and j , $c = \langle \langle i, j \rangle, [t_s, t_e], v \rangle$
$(v, [s_v^i, e_v^i], i)$	Reservation data of interval $[s_v^i, e_v^i]$ for node v by agent i
t_c	Current time
C_{max}	Maximal duration of the plan in the ordered set P_i
p_1	First (shortest duration) plan in the ordered set P_i
C_1	Shortest duration of the plan in the ordered set P_i
c_1	First conflict in plan p_1
C	Set of all conflicts at node v
c_f	Last element of the set C sorted in order of occurrence time
u	Wait length required to resolve conflict c_f
u_v	Wait length of <i>wait</i> at node v
$w(p_1)$	Maximal length of <i>wait</i> required to resolve the conflicts in plan p_1 , $w(p_1) = u_{max}$
$wait(u_{max})$	Waiting action for length u_{max}
β	Tolerance parameter for retaining the selected plan
\mathcal{T}	Set of candidate tasks for selection by agent i , $\mathcal{T} \subset \mathcal{T}$
\mathcal{A}	Set of \mathcal{N} ants, $\mathcal{A} = \{a_1, \dots, a_{\mathcal{N}}\}$
η_v	Amount of pheromone on the state node $v \in \mathcal{V}_i$
$dur(p_j)$	Total duration of p_j
ρ	Pheromone evaporation rate
\mathcal{V}^j	Set of states to which ant a_j (or agent j) can transition from state v
p_μ^j	Probability that a_j (or j) selects $\mu \in \mathcal{V}^j$ as the next transition state
ξ_μ	Heuristic value of state μ
ψ, δ	Parameters that weights the pheromone and heuristic value
I	No. of iterations of path and action planning in ACO
S	Maximam No. of state transitions
T, T_{ini}, T_{ter}	Current, initial, and terminate temperature
α	Cooling rate
$f(p)$	Evaluation function for the solution (plan) p
ζ	Maximum No. of times for generating initial solution in SA
p_{bst}	Best solution (plan) of agent in SA

Appendix B Meta-Heuristic Algorithms Used in the Experiments

Ant Colony Optimization for N-MAPD

We describe the ACO algorithm used for path and action planning for agent $i \in A$ in N-MAPD in this appendix section. We define a set of \mathcal{N} ants as $\mathcal{A} = \{a_1, \dots, a_{\mathcal{N}}\}$. Agent i generates the weighted state graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ from a graph $G = (V, E)$. Note that in ACO, unlike PAPO (Section. 4.1), i does not use a path to generate \mathcal{G}_i ; thus, generates state nodes for all nodes in V . Ant $a_j \in \mathcal{A}$ moves on \mathcal{G}_i

to generate a plan from the current state $v_s^j = (v_s^j, o_s^j) \in \mathcal{V}_i$ of i to the destination state $v_d^j = (v_d^j, o_d^j) \in \mathcal{V}_i$. We denote the amount of pheromone on the state node $v \in \mathcal{V}_i$ as η_v (≥ 0). For $\forall v \in \mathcal{V}_i$, η_v is initialized with 0. We denote the total duration by $dur(p_j) = \sum_{l=1}^{|p_j|} \omega(v_{l-1}, v_l)$ for the plan $p_j = (v_0 (= v_s^j), v_1, \dots, v_d^j)$ generated by ant a_j . For $\forall v \in \mathcal{V}_i$, the amount of pheromone η_v is updated by

$$\eta_v \leftarrow \rho \cdot \eta_v + \Delta\eta, \text{ and } \Delta\eta^j = \begin{cases} \frac{1}{dur(p_j)} & (p_j \neq \text{false and } v \in p_j) \\ 0 & (\text{otherwise}), \end{cases}$$

where $\Delta\eta = \sum_{a_j \in \mathcal{A}} \Delta\eta^j$ and ρ ($0 < \rho < 1$) is the pheromone evaporation rate. Note that $p_j = \text{false}$ means that a_j failed to generate a plan.

Let $\mathcal{V}^v \subset \mathcal{V}_i$ be the set of states to which ant a_j can transition from state v , considering environmental constraints and conflicts with other agents. The probability p_μ^j that a_j selects $\mu \in \mathcal{V}^v$ as the next transition state is denoted by

$$p_\mu^j = \frac{(\eta_\mu)^\psi (\xi_\mu)^\delta}{\sum_{\mu' \in \mathcal{V}^v} (\eta_{\mu'})^\psi (\xi_{\mu'})^\delta},$$

where ξ_μ is the heuristic value of state μ and ψ, δ are non-negative real number parameters that weights the pheromone and heuristic value. We use as heuristic value the inverse of Formula 3, using the Manhattan distance as the distance l in terms of computational cost. Ants perform the following process for plan generation:

- (1) Initialize the amount of pheromone on \mathcal{V}_i and set the initial state v_s^i and the destination state v_d^i for each ant. Then, repeat (2) to (4) following I (> 0) times.
- (2) $\forall a_j \in \mathcal{A}$ selects the next transition state μ from \mathcal{V}^v according to probability p_μ^j and sets μ to the current state of a_j .
- (3) Repeat (2) for all ants until the destination state v_d^i is reached, $\mathcal{V}^v = \emptyset$, or the number of state transitions reaches S , where S is a threshold parameter to prevent ants from endlessly selecting the same state. This means that if ant a_j cannot reach v_d^i in S times transitions, then false is set to p_j . Furthermore, let $p_j = \text{false}$ also for $\mathcal{V}^v = \emptyset$.
- (4) Update the amount of pheromone η_v in $\forall v \in \mathcal{V}_i$ based on the plan p_j of $\forall a_j \in \mathcal{A}$.
- (5) Among all the generated plans, the shortest duration plan p is output as the plan to be executed by agent i . If all plans fail to generate, the run is counted as a failure (i.e., the success rate is reduced).

Algorithm 4 Simulated Annealing for N-MAPD

```

1: function SA( $i, v_s, v_d$ ) // Called from  $i \in A$ 
2:   //  $v_s$ :  $i$ 's current state and  $v_d$ :  $i$ 's destination state
3:   //  $T_{ini}$ : initial temperature and  $T_{ter}$ : terminate temperature
4:   //  $\zeta$ : Maximum number of times for generating initial solution
5:    $T \leftarrow T_{ini}$ ,  $p \leftarrow \text{false}$ ,  $p_{bst} \leftarrow \text{false}$ ,  $b \leftarrow 0$ 
6:   while  $p = \text{false}$  and  $b < \zeta$  do
7:      $p \leftarrow \text{GENSOLUTION}(i, v_s, v_d)$ 
8:      $b \leftarrow b + 1$ 
9:   end while
10:  if  $p = \text{false}$  then return false // plan generation failure
11:  end if
12:  while  $T > T_{ter}$  do
13:     $p' \leftarrow \text{GENNEIGHBOR}(i, v_d, p)$  //  $p'$  is a neighborhood solution
14:    if  $p' \neq \text{false}$  then
15:       $\Delta E \leftarrow f(p') - f(p)$  // where  $f$  is the evaluate function.
16:      if  $\Delta E > 0$  then
17:         $p \leftarrow p'$  // Update current solution
18:        if  $f(p) \geq f(p_{bst})$  then
19:           $p_{bst} \leftarrow p$  // Update best solution
20:        end if
21:      else if  $\text{rand}(1) \leq \exp(\Delta E/T)$  then // where  $\text{rand}(1)$  is a
uniform random number from 0 to 1.
22:         $p \leftarrow p'$ 
23:      end if
24:    end if
25:     $T \leftarrow \text{SCHEDULE}(T)$  // Annealing Schedule
26:  end while
27:  return  $p_{bst}$ 
28: end function

```

Simulated Annealing for N-MAPD

We describe path and action planning using SA in N-MAPD. Agent $i \in A$ generates the weighted state graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ from a graph G without using a path similar to ACO, and generates a plan from the current state $v_s^i = (v_s^i, o_s^i) \in \mathcal{V}_i$ to the destination state $v_d^i = (v_d^i, o_d^i) \in \mathcal{V}_i$ on \mathcal{G}_i . The pseudocode of SA is presented in Algorithm 4. We describe the GENSO-LUTION function (Line 7). Let $\mathcal{V}^v \subset \mathcal{V}_i$ be the set of states to which agent i can transition from state v , considering environmental constraints and conflicts with other agents, then i selects $\mu \in \mathcal{V}^v$ as its next transition state according to probability $p_\mu^i = \xi_\mu / \sum_{\mu' \in \mathcal{V}^v} \xi_{\mu'}$, where ξ_μ is the heuristic value of the state μ and is obtained in the same way as in ACO. Similarly to ACO, a threshold parameter S is introduced for the number of state transitions, so that if v_d^i cannot be reached after S times transitions, the GENSO-LUTION function returns *false* as plan generation failure. If the GENSO-LUTION function returns *false* when generating the initial solution (plan), the run is considered a failure (i.e., the success rate is reduced). We use $f(p) = 1/dur(p)$ as the evaluation function f for the solution p if $p \neq false$; otherwise, $f(p) = -1$. Agent i performs the following GENNEIGHBOR function to generate a neighborhood solution p_i' for the current solution $p_i = (v_0(= v_s^i), v_1, \dots, v_d^i)$ (Line 13).

- (1) Using a uniform random number $k \in \mathbb{Z}_+$ from 0 to $|p_i| - 1$, generate $p_i^1 = (v_0, v_1, \dots, v_{k-1})$ from p_i .
- (2) Generate a plan $p_i^2 = (\mu_0(= v_k), \mu_1, \dots, v_d^i)$ from $v_k \in p_i$ to v_d^i by the GENSO-LUTION function.
- (3) Combine p_i^2 at the tail of p_i^1 to generate a neighborhood solution $p_i' = (v_0(= v_s^i), v_1, \dots, v_{k-1}, \mu_0(= v_k), \mu_1, \dots, v_d^i)$.

Among the many methods available for the SCHEDULE function to schedule annealing (Tsuzuki et al. [31]), we apply the geometric cooling. Therefore, using the cooling rate α ($0 < \alpha < 1$), the temperature T is updated by the expression $T \leftarrow \alpha T$ (Line 25).

Funding This study was funded by JSPS KAKENHI Grant Numbers 20H04245 and 17KT0044.

Availability of Data and Materials All data presented here are generated from the simulated environment.

Code Availability Not applicable.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing,

adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Alonso-Mora J, Beardsley P, Siegwart R. Cooperative Collision Avoidance for Nonholonomic Robots. *IEEE Trans Rob.* 2018;34(2):404–20. <https://doi.org/10.1109/TRO.2018.2793890>.
2. Bareiss D, van den Berg J. Generalized reciprocal collision avoidance. *Int J Robot Res.* 2015;34(12):1501–14. <https://doi.org/10.1177/0278364915576234>.
3. Barták R, Švancara J, Škopková V, et al. Multi-agent path finding on real robots. *AI Commun.* 2019;32(3):175–89. <https://doi.org/10.3233/AIC-190621>.
4. Bellusci M, Basilico N, Amigoni F. Multi-Agent Path Finding in Configurable Environments. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020;pp 159–167.
5. Boyarski E, Felner A, Stern R, et al. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence* 2015.
6. Boyarski E, Felner A, Sharon G, et al. Don't split, try to work it out: bypassing conflicts in multi-agent pathfinding. In: *Twenty-Fifth International Conference on Automated Planning and Scheduling* 2015.
7. Felner A, Stern R, Shimony SE, et al. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In: *Tenth Annual Symposium on Combinatorial Search* 2017.
8. Ho F, Salta A, Geraldès R, et al. Multi-agent path finding for UAV traffic management. In: *Proceedings of the 18th international conference on autonomous agents and multiagent systems, international foundation for autonomous agents and multiagent systems*, 2019;pp 131–139.
9. Hönig W, Kumar TS, Cohen L, et al. Multi-agent path finding with kinematic constraints. In: *Twenty-sixth international conference on automated planning and scheduling* 2016.
10. Huang T, Dilkina B, Koenig S. Learning Node-Selection Strategies in Bounded-Suboptimal Conflict-Based Search for Multi-Agent Path Finding. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021;pp 611–619.
11. Kou NM, Peng C, Yan X, et al. Multi-agent Path Planning with Non-constant Velocity Motion. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2019;pp. 2069–2071.
12. Krakowczyk D, Wolff J, Ciobanu A, et al. Developing a distributed drone delivery system with a hybrid behavior planning system. In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, Springer, 2018;pp 107–114.
13. Li J, Surynek P, Felner A, et al. Multi-agent path finding for large agents. *Proc AAAI Conf Artif Intell.* 2019;33(01):7627–34. <https://doi.org/10.1609/aaai.v33i01.33017627>.

14. Li J, Ran M, Xie L. Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization. *IEEE Robot Autom Lett.* 2021;6(2):405–12. <https://doi.org/10.1109/LRA.2020.3044834>.
15. Li M, Qin Z, Jiao Y, et al. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. In: *The World Wide Web Conference, ACM, 2019b*;pp 983–994. <https://doi.org/10.1145/3308558.3313433>.
16. Liu M, Ma H, Li J, et al. Task and Path Planning for Multi-Agent Pickup and Delivery. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019*;pp 1152–1160
17. Ma H, Tovey C, Sharon G, et al. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In: *Thirtieth AAAI Conference on Artificial Intelligence 2016*.
18. Ma H, Koenig S, Ayanian N, et al. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint 2017a* [arXiv:1702.05515](https://arxiv.org/abs/1702.05515).
19. Ma H, Li J, Kumar T, et al. Lifelong multi-agent path finding for online pickup and delivery tasks. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2017b*;pp 837–845.
20. Ma H, Hönig W, Kumar TS, et al. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. In: *Proceedings of the AAAI Conference on Artificial Intelligence, 2019*;pp 7651–7658. <https://doi.org/10.1609/aaai.v33i01.33017651>.
21. Machida M. Polynomial-Time Multi-Agent Pathfinding with Heterogeneous and Self-Interested Agents. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019*;pp 2105–2107.
22. Morris R, Pasareanu CS, Luckow K, et al. Planning, scheduling and monitoring for airport surface operations. In: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence 2016*.
23. Okumura K, Machida M, Défago X, et al. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, 2019*;pp 535–542. <https://doi.org/10.24963/ijcai.2019/76>.
24. Okumura K, Tamura Y, Défago X. Time-Independent Planning for Multiple Moving Agents. In: *Proceedings of the AAAI Conference on Artificial Intelligence, 2021*;pp 11,299–11,307.
25. Salzman O, Stern R. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 2020*;pp 1711–1715.
26. Sharon G, Stern R, Felner A, et al. Conflict-based search for optimal multi-agent pathfinding. *Artif Intell.* 2015;219:40–66. <https://doi.org/10.1016/j.artint.2014.11.006>.
27. Silver D. Cooperative Pathfinding. In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI Press, AIIDE'05, 2005*;pp 117–122.
28. Sugiyama A, Sea V, Sugawara T. Emergence of divisional cooperation with negotiation and re-learning and evaluation of flexibility in continuous cooperative patrol problem. *Knowl Inf Syst.* 2019;60(3):1587–609. <https://doi.org/10.1007/s10115-018-1285-8>.
29. Surynek P. On Satisfiability Modulo Theories in Continuous Multi-Agent Path Finding: Compilation-based and Search-based Approaches Compared. In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, INSTICC. SciTePress, 2020*;pp 182–193. <https://doi.org/10.5220/0008980101820193>.
30. Tang S, Kumar V. Safe and complete trajectory generation for robot teams with higher-order dynamics. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016*;pp 1894–1901. <https://doi.org/10.1109/IROS.2016.7759300>.
31. Tsuzuki MdSG, de Castro Martins T, Takase FK. ROBOT PATH PLANNING USING SIMULATED ANNEALING. *IFAC Proceedings Volumes.* 2006;39(3):175–80. <https://doi.org/10.3182/20060517-3-FR-2903.00105>, <https://www.sciencedirect.com/science/article/pii/S1474667015358250>, 12th IFAC Symposium on Information Control Problems in Manufacturing.
32. Veloso M, Biswas J, Coltin B, et al. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In: *Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press, IJCAI'15, 2015*;pp 4423–4429.
33. Wang H, Rubenstein M. Walk, Stop, Count, and Swap: Decentralized Multi-Agent Path Finding With Theoretical Guarantees. *IEEE Robotics and Automation Letters.* 2020;5(2):1119–26. <https://doi.org/10.1109/LRA.2020.2967317>.
34. Wang KHC, Botea A. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *J Artif Intell Res.* 2011;42:55–90.
35. Wurman PR, D'Andrea R, Mountz M. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magn.* 2008;29(1):9–9. <https://doi.org/10.1609/aimag.v29i1.2082>.
36. Yakovlev K, Andreychuk A, Rybecký T, et al. On the Application of Safe-Interval Path Planning to a Variant of the Pickup and Delivery Problem. In: *Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO, INSTICC. SciTePress, 2020*;pp 521–528. <https://doi.org/10.5220/0009888905210528>.
37. Yamauchi T, Miyashita Y, Sugawara T. Path and Action Planning in Non-uniform Environments for Multi-agent Pickup and Delivery Tasks. In: *European Conference on Multi-Agent Systems, Springer, 2021*;pp 37–54. https://doi.org/10.1007/978-3-030-82254-5_3.
38. Yen JY. Finding the k shortest loopless paths in a network. *Manag Sci.* 1971;17(11):712–6. <https://doi.org/10.1287/mnsc.17.11.712>.
39. Yoneda K, Sugiyama A, Kato C, et al. Learning and relearning of target decision strategies in continuous coordinated cleaning tasks with shallow coordination I. *Web Intell.* 2015;13(4):279–94. <https://doi.org/10.3233/WEB-150326>.
40. Yoshida N, Noda I, Sugawara T. Multi-agent Service Area Adaptation for Ride-Sharing Using Deep Reinforcement Learning. In: *International Conference on Practical Applications of Agents and Multi-Agent Systems, Springer, 2020*;pp 363–375. https://doi.org/10.1007/978-3-030-49778-1_29.
41. Zhang H, Li J, Surynek P, et al. Multi-Agent Path Finding with Mutex Propagation. In: *Proceedings of the International Conference on Automated Planning and Scheduling, 2020*;pp. 323–332.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.