systems

MDPI

Article

Integration of Efficient Techniques Based on Endpoints in Solution Method for Lifelong Multiagent Pickup and Delivery Problem †

Toshihiro Matsui

Department of Computer Science, Nagoya Institute of Technology, Nagoya 466-0061, Japan; matsui.t@nitech.ac.jp

† This paper is an extended version of our paper published in Matsui, T. Investigation of Integrating Solution Techniques for Lifelong MAPD Problem Considering Endpoints. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics*; The PAAMS Collection. PAAMS 2023. Lecture Notes in Computer Science; Mathieu, P., Dignum, F., Novais, P., De la Prieta, F., Eds.; Springer: Cham, Switzerland, 2023; Volume 13955, https://doi.org/10.1007/978-3-031-37616-0_15.

Abstract: We investigate the integration of several additional efficient techniques that improve a solution method for the lifelong multiagent pickup-and-delivery (MAPD) problem to reduce the redundancy in the concurrent task execution and space usage of a warehouse map. The lifelong MAPD problem is an extended class of iterative multiagent pathfinding problems where a set of shortest collision-free travel paths of multiple agents is iteratively planned. This problem models a system in automated warehouses with robot-carrier agents that are allocated to pickup-and-delivery tasks generated on demand. In the task allocation to agents, several solution methods for lifelong MAPD problems consider the endpoints of the agents' travel paths to avoid the deadlock situations among the paths due to the conflict of the endpoints. Since redundancies are found in the problem settings themselves and the concurrency of allocated tasks, several additional techniques have been proposed to reduce them in solution methods. However, there should be opportunities to investigate the integration of additional techniques with improvements for more practical solution methods. As analysis and an improved understanding of the additional solution techniques based on endpoints, we incrementally integrate the techniques and experimentally investigate their contributions to the quality of task allocation and the paths of the agents. Our result reveals significant complementary effects of the additionally integrated techniques and trade-offs among them in several different problem settings.

Keywords: multiagent pickup-and-delivery; multiagent pathfinding; lifelong problem; endpoints

check for updates

Citation: Matsui, T. Integration of Efficient Techniques Based on Endpoints in Solution Method for Lifelong Multiagent Pickup and Delivery Problem. *Systems* **2024**, *12*, 112. https://doi.org/10.3390/systems12040112

Academic Editors: Philippe Mathieu, Dalila Durães, Alfonso González-Briones and Fernando De la Prieta Pintado

Received: 30 January 2024
Revised: 15 March 2024
Accepted: 23 March 2024
Published: 27 March 2024

Copyright: © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In this study, we investigate the integration of several additional efficient techniques that improve the solution methods for the lifelong multiagent pickup-and-delivery (MAPD) problem, which is an extended class of iterative multiagent pathfinding (MAPF) problems, to reduce the redundancy in concurrent task execution and the space usage of a warehouse map.

The MAPF problem is a fundamental problem in multiagent systems, and its goal is to find a set of shortest collision-free paths for multiple agents in a time-space graph. Two cases of agents colliding on a graph of a two-dimensional map must be avoided. A vertex collision is a situation where multiple agents simultaneously stay in the same vertex, and in an edge/swapping collision, two agents move from both ends of the same edge in opposite directions.

MAPF problems are motivated by various applications, including the navigation of mobile robots [1], autonomous vehicles in warehouses [2], a swarm of drones [3], autonomous taxiing of airplanes [4], autonomous car operation [5], and video games [6]. There are various studies of MAPF/MAPD problems and solution methods (Table 1).

The solution methods for MAPF problems are categorized as either complete or incomplete methods. The former type finds an optimal solution under a criterion of optimization. The conflict-based search (CBS) algorithm [7] is a complete solution method consisting of two levels of search processes. In a higher level, a tree-search method finds colliding situations of agents and inserts constraints that inhibit colliding moves by agents to the pathfinding problems at the lower level. Then each agent solves its pathfinding problem using the A* algorithm in a time-space graph under the inserted constraints. Since this class of solution methods requires relatively high computational cost, several efficient/approximate methods have been developed, including the priority-based search (PBS) [8] and enhanced CBS (ECBS) [9] algorithms.

The incomplete methods find quasi-optimal solutions in exchange for relatively low computational cost. There are several types of solution methods based on dedicated problem settings for them. In the cooperative A* (CA*) algorithm [6], the A* algorithm [10,11] in a time-space graph is repeatedly performed for each agent with a predefined order of agents to find a collision-free path, where each path is reserved in such an order. This is a greedy approach where the agents simply avoid the previously reserved paths.

The priority inheritance with backtracking (PIBT) algorithm [12,13] repeatedly determines the moves of agents in each subsequent time step by reactively resolving collisions among agents. It employs a push-and-rotate operation and introduces the priority values of agents and a partial backtracking method.

The lifelong MAPD problem is an extended class of the iterative multiagent pathfinding problem [12], where new travel paths of multiple agents are repeatedly planned at the ends of their travel. This problem models a system in automated warehouses with robot carrier agents, which are allocated to pickup-and-delivery tasks generated on demand.

Although there are various classes of optimization/routing problems related to logistics, including the variants of vehicle-routing problems [14,15], several dedicated solution methods are necessary for them. A unique setting of MAPF/MAPD problems is that they extend a traditional shortest pathfinding problem so that multiple (ideally) shortest paths for multiple agents must be found without collisions of agents in a time-space graph. Although several techniques for traveling-salesperson problems (TSPs) can be employed as a part of solution methods, the variants of MAPF problems generally focus on the lower layers of routing/pathfinding problems containing obstacles than that for TSPs.

A solution method for MAPD problems consists of task allocation to agents and an MAPF method for the allocated tasks. In the case of static problems with a fixed number of tasks, the task allocation problem is often formalized as a fundamental combinatorial problem [16]. On the other hand, for lifelong MAPD problems, partially greedy task allocation methods are reasonable for tasks continuously generated on demand [17].

While PIBT can also be effectively applied to several cases of lifelong MAPD problems, it cannot correctly work with a map containing dead ends.

The token passing (TP) algorithm [17], which is a solution method for lifelong MAPD problems, is based on a greedy task allocation method and the CA* algorithm. With TP, each task is allocated to an agent in a specific order, and a travel path for the task is reserved. This class of algorithms solves well-formed MAPD problems, where the endpoints (EPs) of agents' travel paths are considered to avoid deadlock situations on the paths. A well-formed problem has several types of EPs, including pickup, delivery, and parking locations, and the condition of the feasible solutions is defined with EPs. Then a solution method exclusively allocates each task to an agent under a rule to avoid conflicts among the endpoints of the agents' paths.

Since there are redundancies in the problem settings themselves and the concurrency of the allocated tasks, several additional techniques have been proposed to reduce them in the solution methods. The multi-label A* (MLA*) algorithm [18] improves the CA* algorithm by employing the knowledge of the agents' pickup locations/times to prune and relax pathfinding for allocated tasks. The standby-based deadlock avoiding method (SBDA) [19], which is a dedicated solution for maze-like maps with just a few EPs, dynamically places standby locations of agents for narrow maps.

However, opportunities undoubtedly exist for investigating the integration of additional efficient techniques with improvements for more practical solution methods. Such an investigation will also be a useful tool to consider several modes in agents' moves. As an analysis and better understanding of the additional solution techniques based on endpoints, we incrementally integrate the techniques and experimentally investigate their contributions to the quality of task allocation and agents' paths. We integrate several techniques: (1) prioritizing task allocation, (2) generalization of EP types, (3) relaxation of constraints excluding EPs from paths, (4) dummy retreat paths, and (5) the subgoal divisions of pickup-and-delivery tasks into TP. Our result reveals some significant complementary effects of integrated additional techniques and trade-offs among them in several cases of problem settings. In particular, the full combination of our selected techniques with several improvements, excluding an experimental version, resulted in the best makespan in most cases. We believe that our experimental analysis will contribute to further developments of efficient solution methods.

This work is an extension based on a conference paper [20]. We refined its experimental results by adding new benchmark problems, improved our proposed methods including the subgoal division, and revised our paper's description by adding detailed explanations and examples.

The rest of this paper is organized as follows. In the next section, we briefly describe related works. In Section 3, we present the background of our study, including the multiagent pickup-and-delivery problem, a solution method called token passing, and additional efficient techniques that are integrated in our study. Then we describe our proposed approach in Section 4. We complementarily integrate the additional techniques with several improvements and apply our solution methods to an environment with relatively narrower maps. We experimentally investigate our approach in Section 5, discuss several issues in Section 6, and conclude in Section 7.

Table 1. MAPF/MAPD problem and approaches.

MAPF				
Appraoch	**Solver**	**Technique**	**Variants**	**Note**
Offline MAPF				
Exact	CBS [7]	Tree search to resolve collisions A* in time-space	ECBS [9], PBS [8]	Relatively high computational cost
Greedy/ Heuristic	CA* [6]	Greedy ordering of agents A* in time-space		
Iterative MAPF				
Greedy/Heuristic	PIBT [12]	Priority of agentsPush-and-rotate/backtracking	winPIBT [13]	Graphs (maps) with cycles
MAPD				
Appraoch	Solver	Task allocation	MAPF	Note
Offline MAPD				
Heuristic	TA-Prioritized/ TA-Hybrid [16]	TSP	Heuristic/ max-flow	
Lifelong MAPD				
Greedy/Heuristic	TP [17]	Greedy	Variant of CA*	Well-formed problems
	MLA* [18]			
	PIBT (for MAPD)		PIBT	Graphs (maps) with cycles
	SBDA [19]		Variant of CA*	Maze-like maps

2. Related Works

In this section, we describe related works by extending our explanation from the previous section. We start with the multiagent path finding (MAPF) problem and briefly describe the extended classes of problems including iterative MAPF ones. Then we discuss multiagent pickup-and-delivery (MAPD) problems which are another iterative MAPF problem. For the classes of problems we refer to several solution methods. Finally we focus on a solution method for a class of MAPD problems and several additional techniques to improve such methods. Since the relationship among the classes of problems and the solution methods is slightly complicated, we summarize most of them in Table 1 in advance.

The MAPF problem is a base of the MAPD problem, and solution methods for MAPF problems are also employed for MAPD problems. The solution methods of MAPF problems are categorized as either complete or incomplete methods. The complete solution methods find optimal solutions without collisions in the agents' paths under a criterion of optimality. Typical criteria are the total number of moves for all agents and the makespan, which is the time required to complete the moves of all the agents.

The conflict-based search (CBS) [7] is a complete solution method that employs two levels of search processes. In the higher level, a tree-search method finds the conflicts of the agents' paths and resolves them by inserting constraints. On the other hand, in the lower level, a pathfinding method in a time-space graph is performed under the inserted constraints. Here each agent initially has its own shortest path ignoring other agents. The colliding situations between two agents are incrementally identified and inhibited in a tree search so that each agent recomputes its path avoiding such collisions. Since this method requires relatively high computational cost, efficient/approximate methods have been proposed [8,9]. In several studies, a MAPF problem was formalized as a satisfiability, satisfiability modulo theories, or answer set programming problem, and a dedicated solver for such a problem was employed [21–23].

There are solvable and unsolvable MAPF problems. Intuitively, if a map has no sufficient space, agents cannot avoid each other [17]. This situation also depends on the population density of the agents.

The incomplete solution methods do not assure the optimality of solutions, and they generally need relatively low computational cost. Such methods usually depend on specific feasible problem settings.

The cooperative A* (CA*) algorithm [6] finds and reserves each agent's travel path in a predefined order of agents. The collision avoidance in this approach basically depends on consistent reservations of agents' travel paths. During the pathfinding, the A* algorithm is performed in a time-space graph under the previously reserved agents' paths.

A different approach employs the push, rotate, and swap operations to resolve collisions among agents' moves [24,25]. Here the agents have their default shortest paths ignoring other agents and basically resolve their collisions on demand. This partially resembles sliding puzzles or Sokoban. The priority inheritance with backtracking method (PIBT) [12,13] based on push-and-rotate operations manages the priority values of agents and employs a limited backtracking method in the push operation. Although the solution method is designed for a specific type of problem where all the vertices in a map's graph are contained in cycles, it effectively resolves collisions among agents.

There are several extended classes of MAPF problems. An important example is the iterative MAPF problem [12], where each agent updates its new goal location at its current goal. Other extended problem settings addressed various more practical situations, including delays in agents' moves [26,27], continuous time in the moves [28,29], the asynchronous execution of tasks [30], moves to any direction [31], and agents with a specific type of bodies [32]. Dedicated solution methods were also developed for the extended problems.

The multiagent pickup-and-delivery (MAPD) problem is a class of iterative MAPF problems where agents are allocated to transportation tasks, and the lifelong MAPD problem addresses tasks generated on demand [17]. The lifelong MAPD problem is a class of iterative MAPF problems, since each agent has its sequence of goal locations for

pickup-and-delivery tasks that are sequentially allocated to the agent. A single pickup-and-delivery task itself of a MAPD problem can also be considered an iterative MAPF problem because it has a subgoal of a pickup location.

Several solution methods for iterative MAPF problems and MAPD problems basically perform iterative processes for a sequence of (sub-)goals. However, the details of the solution processes are various.

The typical criteria of optimality in lifelong MAPD problems are the service time to complete each task and the makespan. A solution method for MAPD problems is decomposed into task allocation to agents and the MAPF for agents with allocated tasks. For a static problem with a fixed number of tasks, task allocation can be solved with a complete search method, although the issue of computational complexity remains. Task allocation can be represented as several types of combinatorial optimization problems, including the traveling-salesperson problem (TSP) [16].

We note that TSP is employed not to solve the routing of agents but to sort their allocated tasks as a cycle graph. Although perhaps MAPF/MAPD problems can be integrated with several vehicle-routing problems (VRPs) [14,15], a relatively large gap appears to exist between them in both the problem definitions and the solution methods. For these classes of methods, several solution methods for TSPs and VRPs are employed, including local search methods [33], genetic algorithms [34], and machine learning approaches [35–37]. Extended classes with multiple agents contains combinatorial problems of agents and routes, and dedicated extended solution methods are employed [37,38]. On the other hand, MAPF/MAPD problems are extensions of a traditional shortest pathfinding problem and generally focus on the lower layers of routing/pathfinding problems containing obstacles and other colliding agents. Several demand-responsive transport (DRT) systems [39] also employ multiple vehicles, while their collisions are generally excluded from problem settings. Here a transport simulation itself is also a major interest.

When tasks are generated on demand, task allocation is solved using (partially) greedy approaches. In addition, greedy approaches for iterative MAPF problems are often reasonable for the scalability of solution methods.

The token passing (TP) algorithm [17] is a solution method for lifelong MAPD problems. It employs a greedy task allocation method and the CA* algorithm. This solution method depends on the condition of well-formed problems [17,40] and employs a set of rules considering the endpoints of agents' travel paths in the task allocation to avoid deadlock situations among agents. While PIBT can also be employed for MAPD problems, the conditions of feasible problems are different from that of TP. In particular, PIBT cannot be applied to maps containing dead ends.

Since a solution method based on well-formed problems suffers some redundancy in the parallel execution of tasks, several methods, including additional efficient techniques (Table 2), have been proposed to reduce the redundancy.

The multi-label A* (MLA*) algorithm [18], which has been used instead of the CA* algorithm, considers agents' pickup locations/times in the pathfinding to prune the search and relax the restriction of paths. This previous method also employs an ordering of agents in the task allocation that is based on the minimum heuristic path length from the location of a currently available agent to the pickup location of each new task. However, it determines whether the allocation of each task to an agent is possible by executing the MLA* algorithm. On the other hand, the effect of the MLA* algorithm resembles that of another method employed in our study. Moreover, our selected method considers the estimated pickup time of each new task for all the agents, including currently working agents, and do not require a pathfinding process to verify whether the allocation of each task is possible.

Several methods that improve TP partially relax the conditions of the well-formed problems that ensure feasible solutions. The standby-based deadlock avoiding method (SBDA) [19] is a relatively aggressive approach that employs dynamically prepared standby locations of agents. However, it mainly addresses a special case of maze-like maps with a

small number of parking, pickup, and delivery locations. We address the case of warehouse maps that generally contains a sufficient number of such locations.

Table 2. Additional techniques to improve TP algorithm.

Name	Summary/Effect
Pt	Estimation of pickup times of other agents Reduction of pickup path length (Sections 3.3 and 4.2)
Te*W*	Shortcut paths through EPs Reduction of traversal path length (Sections 3.4 and 4.3)
Ge	Generalization of task and non-task EPs Reduction of space usage of warehouse maps (Section 4.1)
DP/DPc-*T*-*P*	Dummy path Improvement of concurrent task allocation (Sections 3.5 and 4.4)
Sg	Division of task into subgoals Reduction of long reserved paths (Sections 3.6 and 4.5)

In this study, we focus on additional techniques to improve the performance of TP. While there are several existing works on such components, opportunities must be found to investigate the effect of integrating such additional techniques to analyze the redundancy in the solution process that can be reduced by the integrated techniques.

We employ several additional techniques with the TP algorithm to mainly improve the concurrency of tasks and the space usage of warehouse maps. To investigate integrated solution methods based on additional techniques, we carefully selected a set of fundamental ones that have different and complementary effects (Table 2) and incrementally combine them. Table 3 shows the possible full combinations of them in our result.

Table 3. Combination of additional techniques to be incrementally integrated.

Possible Full Combinations of Additional Techniques
TP + Pt + Te*W* + Ge + Dp-*T*-*P*
TP + Pt + Te*W* + Ge + Dpc-*T*-*P*
TP + Pt + Te*W* + Ge + Sg

3. Background

In this section, we present the background of our study. We first introduce the MAPD problem and the TP algorithm. Then we describe our selected additional techniques (partially listed in Table 2) for improving the TP algorithm as the basis of our integrated solution methods.

3.1. Lifelong Multiagent Pickup-and-Delivery Problems

The lifelong multiagent pickup-and-delivery (MAPD) problem [17] is an extended class of multiagent pickup-and-delivery problems and iterative multiagent pathfinding (MAPF) problems. Here multiple pickup-and-delivery tasks are generated on demand during a certain period and repeatedly allocated to agents. For these allocated tasks, collision-free travel paths of agents are determined. A problem consists of the following:

- undirected graph $G = (V, E)$ representing a warehouse's map,
- a set of agents $\mathcal{A}$, and
- a set of currently generated pickup-and-delivery tasks $\mathcal{T}$.

Each task $\tau_i \in \mathcal{T}$ has the information of its pickup-and-delivery locations (s_i, g_i), where $s_i, g_i \in V$. An agent assigned to a task travels from its current location to the pickup location and then to the delivery location.

Let $l_i(t) \in V$ denote the location of agent $a_i \in A$ in discrete time step t. Agent a_i moves from $l_i(t)$ to an adjacent location $l_i(t+1) \neq l_i(t)$ or stays in its current location $l_i(t+1) = l_i(t)$ at the end of each time step t. An agents' path is the sequence of its moves for a period of time steps.

There are two cases of colliding agents' moves to be avoided. First, two agents a_i and a_j collide if they simultaneously stay in the same location (vertex conflict): $l_i(t) = l_j(t)$. Second, the agents collide if they simultaneously move on the same edge from both ends of the edge (edge/swapping conflict): $l_i(t+1) = l_j(t) \wedge l_j(t+1) = l_i(t)$. The paths of two agents collide if there are at least one colliding move on them.

The condition of a well-formed MAPD problem that guarantees feasible solutions without deadlock situations in task allocation and multiple pathfinding has been presented [17,40]. Here the vertices, which can be the first and last positions of each path, are called endpoints (EPs). In the case of lifelong MAPD problems, an EP can be pickup, delivery, and parking locations.

In a fundamental well-formed problem, EPs $V^{EP} \subset V$ are categorized into non-task EPs V^{NTSK}, which are initial and parking locations, and task EPs V^{TSK}, which are pickup-and-delivery locations (Figure 1). Here $V^{TSK} \cup V^{NTSK} = V^{EP} \wedge V^{TSK} \cap V^{NTSK} = \emptyset$. With these EPs, the conditions for a well-formed problem are defined as follows (Definition 1 in [17]).

A MAPD is well-formed iff

1. the number of tasks is finite,
2. non-task endpoints are no fewer than the number of agents, and
3. for any two endpoints, there exists a path between them that traverses no other endpoints.

Under the following rules, tasks can be sequentially allocated to agents without deadlock situations among the paths for the allocated tasks.

1. Each path of an agent only contains EPs for the first and last locations of the agent (and a pickup location), and
2. the last location of each path is not included in any other paths.

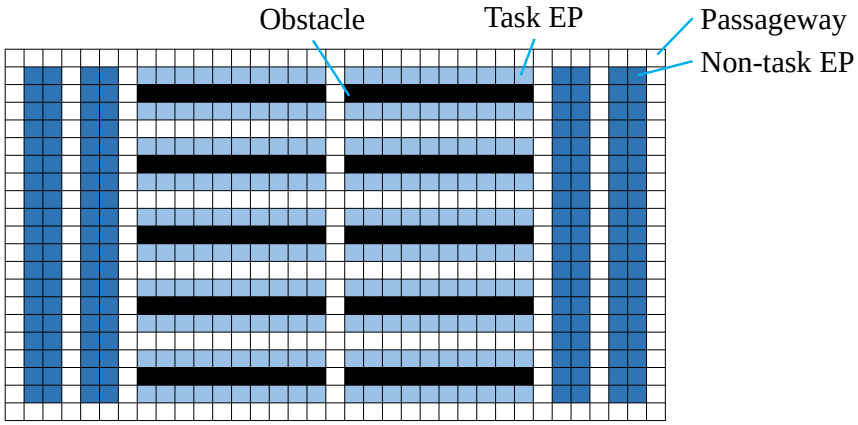

Figure 1. Env. 1: Well-formed MAPD [17] (white: passageway, black: shelf (obstacle), light-blue: task EP, blue: non-task EP). A well-formed problem of a grid-like warehouse map consists of cells that are categorized into passageways, task/non-task EPs, and obstacles. The obstacle cells also represent storage shelves. Only EPs can be the ends of a path (including a parking state) of each agent. Task EPs can be pickup-and-delivery locations of transport tasks; non-task EPs can only be the initial or parking locations of agents. For each pair of EPs, at least one path that contains no other EPs must exist.

The maps of warehouses are generally represented using grids in fundamental studies, and we interchangeably use a vertex in a graph, a cell in a grid world, and a location in a map.

3.2. Solution Method Based on Endpoints

The token passing (TP) algorithm [17] is a fundamental method to solve well-formed MAPD problems (Figure 2). Here a cooperative system is assumed to consists of multiple agents and a centralized process that adds new tasks.

In the pseudo code in Figure 2, we employ the following notations:

- $loc(a)$: agent a's location;
- s_j and g_j: pickup-and-delivery locations of task τ_j;
- $h(v, v')$: heuristic distance from v to v';
- $Path1(a_i, \tau, token)$ and $Path2(a_i, token)$: pathfinding and reservation methods for a pickup-and-delivery path of task τ and a retreat path.

```
1  Initialize token with path [loc(a_i)] for each agent a_i.
2  until a termination condition do // All generated tasks in a period are completed.
3    Add new tasks generated on demand to task set T.
4    foreach agent a_i that determines its next path in token do
5      T' ← {τ_j ∈ T s.t. no other agents' paths in token end in s_j or g_j}.
6      if T' ≠ ∅ then do
7        τ ← arg min_{τ_j ∈ T'} h(loc(a_i), s_j).
8        Assign a_i to τ.
9        Remove τ from T.
10       Update a_i's pickup-and-delivery path in token with Path1(a_i, τ, token).
11       done
12     else if no task τ_j ∈ T s.t. g_j = loc(a_i) exists then
13       Update a_i's stay path in token with path [loc(a_i)].
14     else
15       Update a_i's retreat path in token with Path2(a_i, token).
16     done
17   All agents move along their paths in token for one time step.
18   done
```

Figure 2. Token passing (TP) [17].

The agents sequentially allocate their tasks and reserve corresponding travel paths in a greedy order (line 4) by referring and updating a shared memory called a token (lines 1, 4-5, 10, 13, 15, and 17 in Figure 2). A token contains the information of a list of tasks waiting to be allocated and the paths reserved by the agents.

Each agent without an assigned task tries to allocate a new task that do not cause a deadlock situation under the currently reserved paths. If such a task is allocated, the agent finds and reserves a path that corresponds to its assigned task by the A* algorithm on a time-space graph (lines 6–11).

Here the tasks are prioritized by the distance from the agent's current location to the pickup location for each task (line 7).

For heuristic function $h(v, v')$, we employ the distance in a two-dimensional map with fixed obstacles (shelves) and without agents. The distance value can be cached for the end of the current travel path of each agent and each endpoint, and a cached distance value is updated when a corresponding agent's reservation is updated.

If any waiting tasks cannot be assigned to an agent, the agent remains at its current EP or retreats to one of other EPs. If its current location is the delivery location of a task on the task list, the agent retreats to a free EP that does not conflict with the delivery locations of the tasks on the task list and the reserved paths (line 15).

Otherwise, the agent temporarily stays at its current location (line 13). The agent also reserves a retreat or a stay path (lines 10, 13, and 15). At the end of each time step, each agent moves or stays, as it reserved (line 17).

There is some redundancy in this basic solution method, and several techniques below have been proposed to address this problem.

3.3. Prioritizing Task Allocation Considering Estimated Pickup Times

The task allocation process can employ several ordering criteria to select a waiting task to be assigned to a free agent. Although there are many possible heuristic criteria, there might be some trade-offs among them in different problem instances [18]. Here we focus on a heuristic based on the estimated pickup time of tasks [41] as a reasonable example.

Since TP is based on the reservation of paths, the time of the last location on each reserved path is known. Therefore, to estimate agent a_i's pickup time $\hat{t}^i_{s_j}$ of task τ_j, the heuristic distance from the agent's current goal location $end(a_i)$ to the task's pickup location s_j is added to the (future) time at the agent's goal location: $\hat{t}^i_{s_j} =$ (reserved time at $end(a_i)) + h(end(a_i), s_j)$.

Each agent a_i, which selects its new task to be allocated, compares the estimated pickup time $\hat{t}^i_{s_j}$ of each task τ_j with that of other agents. If the estimated pickup time of one of the other agents is earlier than that of the selecting agent, the allocation of the task is postponed.

In the example shown in Figure 3, idle agent a_i without a task can pick up new task τ_k in eight time steps, while delivering agent a_j can pick it up in five time steps including its current delivery time. Therefore, agent a_i does not immediately select τ_k and tries to select the nearest task among other new ones.

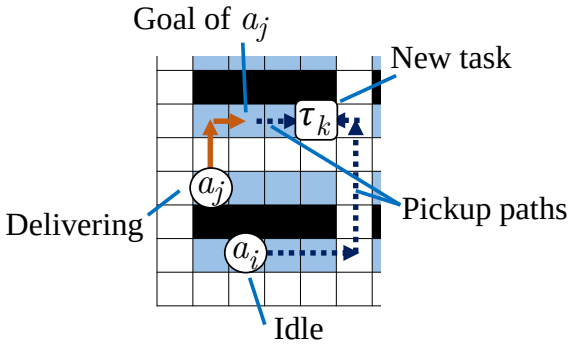

Figure 3. Considering estimated pickup time.

3.4. Relaxation of Paths Containing Arbitrary EPs

To reduce the redundancy in path lengths, an additional technique was proposed, where paths are allowed through EPs [41]. While paths can contain any EPs in this extension, the deadlock situations of paths are avoided by considering conflicting EPs in task allocation.

Here all the EPs in all the reserved paths are locked, and the conflict between the locked EPs and the delivery location of each new task in the task list is checked. Each agent also temporarily reserves its additional 'staying' path at the end of its current path until it cancels the staying reservation and reserves a new path. This information is referred to in the pathfinding for a newly assigned task to avoid the locations at which agents are staying. The effect of this technique partially resembles the pruning and relaxation in pathfinding improved by the MLA* [18] algorithm. With this technique, the length of travel paths can be reduced, although such paths might block more EPs and decrease the number of allocatable tasks.

Figure 4 shows an example where the length of a_i's travel path can be reduced by moving through EPs. In this example, in both the left and right side cases, agent a_i's path ends at the third cell to the right from a_i's current (start) location. While the EPs between the start and end locations of a_i cannot be shortcuts in the original TP algorithm (left side case), the extended method allows a shortcut through EPs (right side case). Agent a_i temporarily reserves a continuous stay at its goal location in addition to its reserved path. The reservation at the current goal location is canceled when a new task is allocated to the agent.

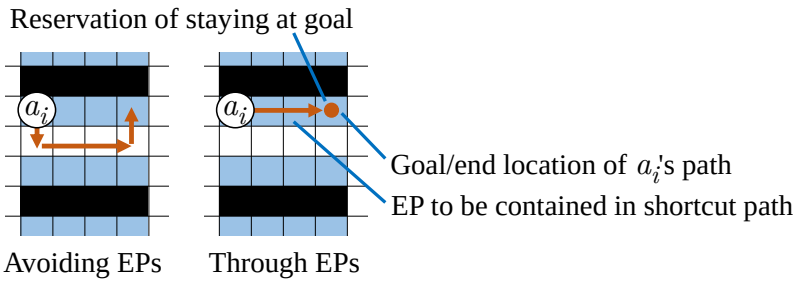

Figure 4. Path through EPs.

3.5. Adding Dummy Path for Retreating

The task allocation is affected by the locked task endpoints that are the last locations of the reserved paths. If a task's delivery location is locked, such a task cannot be assigned to avoid deadlock situations of paths. To reduce the number of locked tasks, additional dummy paths are introduced [16,42]. By adding a dummy path, an agent's path is expanded so that it retreats to a non-task endpoint with no waiting tasks.

A dummy path is created with the destination of a free non-task EP that is not locked by other agents' paths so that an agent reduces the situations where the delivery locations of waiting tasks are locked at the end of the agent's path. An agent selects the destination of its new dummy path from free non-task EPs by referring to the task list and the reserved paths of other agents. Note that the ends of a path in the TP algorithm must be EPs. Therefore, the destination of an additional dummy path is always a free EP (a non-task EP in the previous study).

In addition, such a dummy path might be canceled, if necessary. The additional staying path of an agent shown in Section 3.4 is a special case of a dummy path. Dummy paths can reduce the number of locked tasks, while the agents might reserve relatively long retreat paths if non-task endpoints are located in separate areas in a warehouse.

In the example in Figure 5, the goal locations of agent a_i and new task τ_k were originally identical, and τ_k cannot be allocated to any agents. By adding dummy paths from their common goal location to two different EPs, such a conflict is resolved, and task τ_k can be allocated to an agent.

In a basic case, each dummy path starts from the last location of a path and ends at a parking location [16].

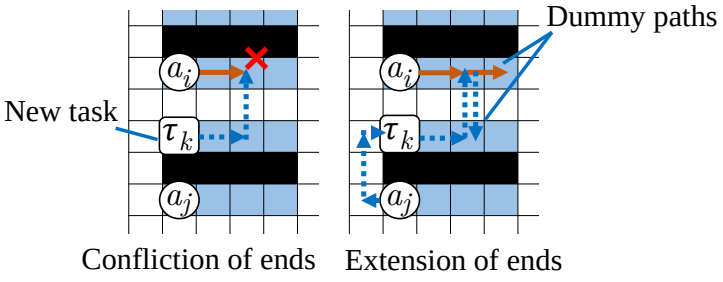

Figure 5. Adding dummy paths.

3.6. Partial Reservation of Tasks' Paths

In lifelong MAPD problems, tasks generated on demand are continuously allocated to multiple agents depending on the situation in each time step. Such a situation might change before an agent traverses a long path, and this can affect the allocation of new tasks and paths.

For this issue, several methods divide the paths of tasks to delay the planning of future paths [13,42]. While there are several relatively complex methods to employ such a partial path within a time window, we focus on a relatively simple method that divides the path of a single task into pickup and delivery paths.

4. Integration and Improvement of Efficient Techniques

To investigate the effects and trade-offs of integrated solution methods based on the additional techniques shown in the previous section, we improve some details of the techniques and incrementally combine them. We also address problems without non-task endpoints.

Our selected additional approaches that have different and complementary effects are summarized in Table 2. In the following, we incrementally combine these techniques. Possible full combinations of them in our result are shown in Table 3.

4.1. Generalization of Task and Non-Task EPs

To reduce the redundant space in warehouses, we allow cases where the initial and retreat locations of agents can be any EPs, including maps without non-task EPs (Figure 6). In this example, the left and right side areas with non-task EPs in the map in Figure 1 are eliminated. This case requires the relaxation of the second condition for well-formed MAPD problems in Section 3.1 that non-task endpoints must not be fewer than the number of agents. Instead, the space usage of warehouses is reduced. In the modified settings, each agent basically retreats to its nearest EP, excluding EPs on the reserved paths and those of the delivery locations of tasks waiting to be allocated. When no EP temporarily exists to retreat to in an environment with fewer non-task EPs than agents, such an agent stays at its current EP.

Even in this modified setting, all tasks can be completed without any deadlock situations for a finite number of on-demand tasks. In specially limited cases without non-task EPs, there are solutions as long as the number of agents $|A|$ is less than the number of task EPs $|V^{TSK}|$.

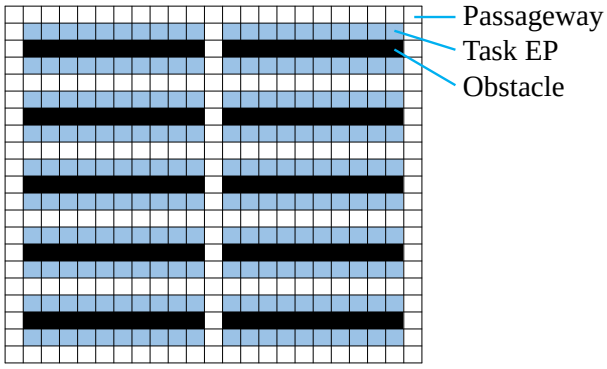

Figure 6. Env. 2: MAPD without non-task EPs (white: passageway, black: shelf (obstacle), light-blue: task EP). To improve the space usage of warehouse maps, several areas are eliminated, including non-task EPs. This requires the relaxation of a condition for well-formed problems regarding the numbers of non-task EPs and deployed agents.

Proposition 1. *The relaxed well-formed problem without non-task EPs can be solved with a modified TP algorithm, if the number of agents is less than the number of task EPs.*

Proof of Proposition 1. We briefly sketch this proof. In the TP algorithm, tasks are sequentially allocated to agents under previously allocated tasks and reserved paths. The reservation of the stay/retreat paths of agents is also performed in the same procedure. Therefore, the movements of agents are serialized in the worst case without deadlock situations as long as at least an agent's new path can be reserved.

If there is only a single free EP v^f, one agent can move to it. A new task whose delivery location is v^f can be allocated to an idle agent. Otherwise, v^f can be the goal location of another agent that retreats from a delivery location of new tasks. Such retreat moves of agents might be repeated until a new task is allocated.

Agent a_i at a pickup location of a waiting task can immediately allocate the task to a_i after an agent in the task's delivery location reserves a retreat path. In addition, in our modified method, agents do not retreat to EPs on reserved paths or to the delivery locations of new tasks waiting to be allocated.

Therefore, at least a single task can be sequentially performed if the number of agents is less than the number of EPs. □

The map in Figure 6 contains 200 (task) EPs, and at most 199 agents can be deployed. On the other hand, the map in Figure 1 contains 152 non-task EPs, and only the same number of agents can be deployed under the conditions for well-formed MAPDs.

4.2. Employing Task Allocation Considering Estimated Pickup Times

We first integrate the generalization of EPs shown in the previous subsection and the task allocation considering the estimated pickup times (Section 3.3). While we borrowed the idea from a previous study [41], we employed the following method considering the balance of computational cost and missing allocations due to a heuristic search. Lines 6 and 7 in Figure 2 are replaced by the procedure shown in Figure 7. Here, $end(a)$ denotes the end of agent a's path.

```
1  d^⊥ ← ∞.
2  τ^⊥ ← empty. // task to be allocated to a_i
3  foreach a_k in A \ {a_i} do
4    d_k^⊥ ← ∞.
5    τ_k^⊥ ← empty. // task assumed to be allocated to a_k
6    done
7  foreach task τ_j in T' do
8      d_i ← h(loc(a_i), s_j) // estimated pickup time of a_i
9      d_k^* ← ∞.
10     k^* ← empty. // ID of an nearest agent to pickup location of τ_j
11     foreach a_k in A \ {a_i} do
12       d_k ← (reserved time at end(a_k)) + h(end(a_k), s_j) // estimated pickup time of a_k
13       if d_k < d_i ∧ d_k < d_k^* then do
14         d_k^* ← d_k.
15         k^* ← k.
16         done
17       done
18     undertaken ← false.
19     if k^* ≠ empty then do
20       if τ_{k^*}^⊥ = empty then do
21         d_{k^*}^⊥ ← d_k^*.
22         τ_{k^*}^⊥ ← τ_j.
23         undertaken ← true.
24         done
25       else if d_k^* < d_{k^*}^⊥ then do
26         if then d_{k^*}^⊥ < d^⊥ do // a_i takes a task back from a_{k^*}.
27           d^⊥ ← d_{k^*}^⊥.
28           τ^⊥ ← τ_{k^*}^⊥.
29         done
30         d_{k^*}^⊥ ← d_k^*.
31         τ_{k^*}^⊥ ← τ_j.
32         undertaken ← true.
33         done
34       done
35   done
36 if ¬undertaken ∧ d_i < d^⊥ then do
37   d^⊥ ← d_i.
38   τ^⊥ ← τ_j.
39   done
40 if τ ≠ empty then do // line 6 in the original pseudo code
41   τ ← τ^⊥. // line 7 in the original pseudo code
```

Figure 7. Task allocation considering estimated pickup time and backup of task (agent a_i).

As in the previous study in Section 3.3, if agent a_k can pick up task τ_j in fewer steps than a_i, then task τ_j should be allocated to a_k. Therefore, agent a_i does not immediately allocate τ_j to it. However, if a_k finds another task that can be picked up by a_k in fewer steps than τ_j, a_k ignores τ_j. We address this issue within a greedy method. In the procedure, agent a_i leaves task τ_j to agent a_k if (reserved time at $end(a_k)) + h(end(a_k), s_j)$ is less than $h(loc(a_i), s_j)$, where $end(a)$ denotes the end of agent a's path (lines 11–34 in Figure 7). In addition, agent a_i assumes that the task will be assigned to a_{k^*} with the minimum estimated pickup time (lines 13–17). If the task nearest a_{k^*} is updated in the search process, a_i takes the previous task back from a_{k^*} (lines 26–29). Then agent a_i keeps the nearest one to it in such tasks. If no agent is nearer task τ_j than a_i, agent a_i compares τ_j with its kept task to select one of them (line 36).

This is basically a greedy method, but it intends to reduce the missing allocation of tasks in a simple method. While the effect of such a heuristic depends on the problem settings, we prefer this approach.

We selected this heuristic as a relatively simple example of a greedy approach and slightly adjusted it to mitigate the situations of temporarily unallocated tasks due to greedy allocation. In cases of relatively sparse populations of agents in an environment containing obstacles, the accuracy of the estimated pickup time, which is partially based on heuristic distance, will be relatively low, and the task allocation might be incorrect. On the other hand, if the number of agents is sufficient, the time required to pickup each task is expected to be reduced, and the total time until the task is delivered can also be reduced.

4.3. Integration with Paths Containing Arbitrary EPs

Next we integrate a technique that allows paths containing arbitrary EPs (Section 3.4). Although this technique reduces the length of the paths, the paths containing EPs block the allocation of new tasks whose delivery locations are such EPs. It also needs to modify its selection of EPs to which to retreat so that the EPs locked by reserved paths are excluded.

We improve pathfinding so that it avoids the EPs of the delivery locations of new tasks. This modification allows the traversal paths of agents to only shortcut EPs which are not the delivery locations of new tasks. Since we found that a strict setting that completely avoids such EPs produces relatively long paths, we adopted a heuristic weight value W for the distances to the EPs to adjust the degree of avoidance.

Figure 8 shows an example of agent a_i's path that avoids the delivery location of new task τ_k. By inserting such a detour, the delivery location of τ_k is not locked by a_i's path, and τ_k can be allocated to an agent.

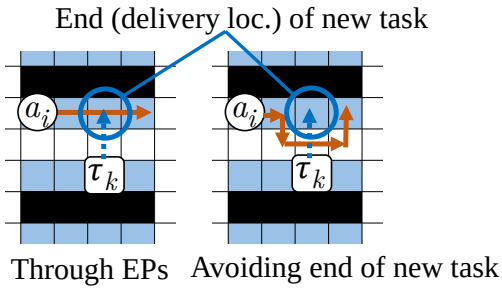

Figure 8. Avoiding ends of new task.

To manage the information of locked EPs, we employ an additional two-dimensional map corresponding to a warehouse's floor plan. Agents share this additional map as part of their token, and each vertex/cell v in the map consists of the following variables:

1. c_v^{LCK}: a counter of the number of paths locking this location;
2. t_v^{STY}: a variable recording the start time of an additional staying path of an agent in this location if one exists;
3. c_v^{TSK}: a counter of the number of waiting tasks whose delivery locations are this location.

The first and second variables, which are related to the reserved paths, are updated when a new path is reserved and when each agent moves on its reserved path at each time step. The last variable is updated when a new task is generated on demand and when a task is allocated to an agent. These are referenced in the additional methods shown above to check the locked EPs. c_v^{LCK} is employed in the task allocation.

In the A* algorithm on a time-space graph, the estimated total path length $g(v^t, v^{t+1})$ for a vertex v^t at time step t and an adjacent vertex v^{t+1} is described as follows. It is commonly represented as

$$g(v^t, v^{t+1}) = f(v^t) + u(v^t, v^{t+1}) + h(v^{t+1}), \tag{1}$$

where $f(v^t)$ is the distance (time) from a start vertex to v^t, $u(v^t, v^{t+1})$ is the distance from v^t to v^{t+1}, and $h(v^{t+1})$ is the heuristic distance from v^{t+1} to a goal vertex. We employ the following distance between neighboring vertices.

$$u(v^t, v^{t+1}) = \begin{cases} \infty & v^{t+1} \text{ is reserved by another agent,} \\ W & c_v^{TSK} > 0, \\ 1 \text{ (unit time)} & \text{otherwise.} \end{cases} \tag{2}$$

Here t_v^{STY} is employed to evaluate the reservation of a staying agent.

4.4. Integration with Management of Additional Dummy Paths

We next integrate the solution methods shown in previous Sections 4.1–4.3 and a technique of the additional dummy paths (Section 3.5). In the basic method, each additional dummy path starts form the last location of a path and ends at a parking location (i.e., mainly non-task EPs). We also address cases where any EP can be used as a parking location and where the number of non-task EPs is less than the number of agents. Therefore, dynamic management of dummy paths is necessary.

For each agent, we introduce a sequence of 'tasks' that consists of the following:

1. a pickup-and-delivery or retreat task, if one exists,
2. optional dummy retreat tasks, and
3. a stay task if there is nothing to do.

Here we temporarily consider any type of travel paths to be corresponding types of tasks and assume a sequence of tasks has maximum length T. Each agent manages the information of its own task sequence.

First, each agent is assigned to just one from among pickup-and-delivery, retreat, and stay tasks and reserves its travel path as usual. Then dummy paths are added in the following two cases.

(1) If an agent has a pickup-and-delivery or a retreat task, and the end of its travel path blocks tasks that are waiting to be allocated, then a dummy retreat task is added, if possible. Here the dummy path's goal location is one of the other available (free) EPs. Each agent performs this operation at every time step.

(2) When an agent can select a new pickup-and-delivery task, the agent conditionally adds a dummy path. If the a new task's delivery location overlaps with a reserved path (excluding its end location), and if there is an available EP to which the agent can retreat, then a dummy path task is added after the allocated task. Next a path is reserved, including pickup, delivery, and retreat locations. To avoid deadlock situations in an environment with a small number of non-task EPs, a pickup-and-delivery task that requires no dummy path is prioritized in the task allocation.

After each agent completes its pickup-and-delivery task or its first retreat task, the agent cancels subsequent retreat tasks and tries to select a new task. However, such a new selected task might be infeasible, since the agent must move from its current location to avoid other agents' paths that are newly reserved after its own dummy path is reserved. In such cases, the canceled tasks and paths are restored to ensure the consistency of reservations.

We limit the maximum length of each task sequence T and the maximum length of each additional retreat path P to restrict the total path lengths. When there is no EP to which to retreat within a range of P from the current path end of an agent, its related preceding task is not allocated.

4.5. Employing Tasks' Paths Divided for Subgoals

Finally, we integrate a method that decomposes a single path of a task into multiple paths corresponding to the subgoals of the original task. In our investigation, we simply decompose a task's path into a sequence of two parts so that the former/latter ends/starts at a pickup location. Therefore, a MAPD problem is modified to a pair of MAPF problems, where each original task is decomposed into pickup and delivery tasks, with a constraint that the decomposed tasks must be assigned to the same agent.

Since this change might introduce different deadlock situations among decomposed tasks, we also modify the conditions in the task allocation.

1. The pickup-and-delivery tasks are associated to their original task and allocated to the same agent.
2. Each agent switches between its pickup and delivery modes and can only assign its consistent tasks. Namely, an agent can only allocate the originally same task in its pair of pickup and delivery modes.
3. The delivery tasks that have been allocated to agents should be prioritized to complete them.

For the last condition regarding the priority of tasks, we simply employ the original task-allocation rule of TP. Namely, the original task with no conflict of pickup and delivery locations is identically allocated to an idle agent as the original TP, although the allocated agent only reserves its pickup path at its first phase. After an agent arrives at the pickup location of its task, it allocates its task again to reserve a path to its delivery location.

The condition of conflict among the agents and the new tasks in line 5 (Figure 2) is modified as follows. Here each delivery location of a partially allocated task (in its pickup phase) is also considered its reserved location.

1. For new tasks: $\mathcal{T}' \leftarrow \{\tau_j \in \mathcal{T}$ s.t. no other agents' paths in the token and the partially allocated tasks end in s_j or $g_j\}$;
2. For reallocated tasks: $\mathcal{T}' \leftarrow \{\tau_j \in \mathcal{T}$ s.t. no other agents' paths in the token and the partially allocated tasks end in $g_j\}$.

In addition, conflicts between the pickup location of each new task and the reserved paths containing EPs are also avoided in the task allocation. Therefore, we also apply a weight parameter of the distance values to avoid the delivery locations of new tasks in the pathfinding (Section 4.3) to the pickup locations of new tasks.

Although this relatively safe rule might still limit the concurrency of task allocation, we focus on whether there is room to reduce the makespan by partially reserved paths. To avoid complicated situations with this approach, we do not employ techniques of dummy paths in this case.

4.6. Correctness of Integrated Method

The selected additional techniques to be integrated in our approach are correct in the sense of no deadlock situation, and we can combine them without a loss of the correctness of the original solution method TP. Therefore, the resulting methods inherit the correctness of the original TP. Indeed, the integrated techniques substantially complement each other and just need a few adjustments regarding the rules of the endpoints and the reserved paths, as mentioned in Sections 4.1 and 4.3–4.5. We do not employ a combination of dummy paths and subgoal-division techniques for simplicity in this paper, as shown in Section 4.5, although we believe that they can be integrated with a slightly complicated adjustment. An understanding of the fundamental property of endpoints also helps develop such integrated solution methods. However, as a baseline case, solution methods still depend on

the conditions of well-formed problems excluding those of non-task endpoints. The base line case is also a safeguard for the correctness of the extended solution methods.

5. Evaluation

We experimentally investigated the effects and influence of the integrated techniques. We first present the settings of our experiment, including the benchmark problems and the compared solution methods. Then we report the experimental results that reveal the effect of our approach.

5.1. Settings

We employed the following types of benchmark problems. We evaluated both well-formed problems with non-task EPs and variants without them.

- Basic (Envs. 1 and 2 in Figures 1 and 6): Env. 1 is based on a well-formed problem shown in a previous study [17], and Env. 2 is its variant without non-task EPs.
- Incoming–storing–outgoing (Envs. 3 and 4 in Figures 9 and 10): We modified the basic problems to represent the task flows. Here task endpoints are categorized as incoming, storing, and outgoing EPs. Half of the tasks are incoming tasks, and the other half are outgoing ones. The pickup/delivery location of each incoming task is an incoming/storing EP, and that of each outgoing task is a storing/outgoing EP.
- Large-scale (Envs. 5 and 6 in Figures 11 and 12): Env. 5 is based on another well-formed problem shown in a previous study [17], and Env. 6 is its variant without non-task EPs.

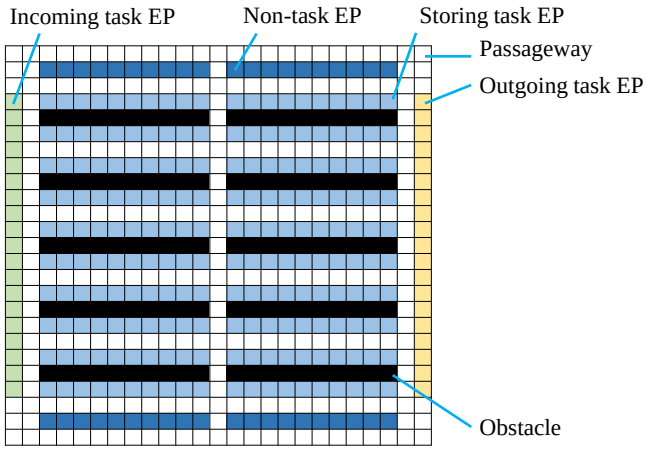

Figure 9. Env. 3: Well-formed MAPD (incoming–storing–outgoing) (light green (left side): incoming EP, light yellow (right side): outgoing EP, other task EPs: storing EPs).

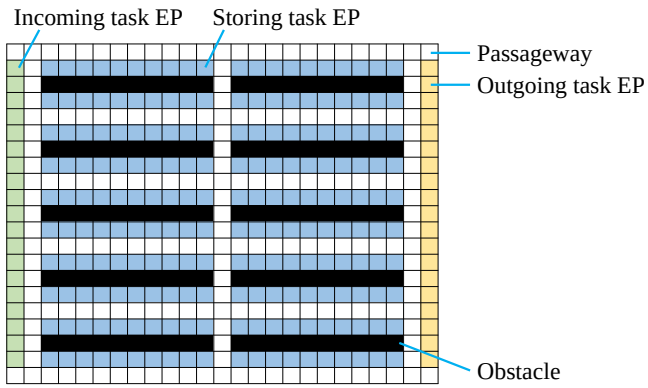

Figure 10. Env. 4: MAPD without non-task EPs (incoming–storing–outgoing) (light green (left side): incoming EP, light yellow (right side): outgoing EP, other task EPs: storing EPs).

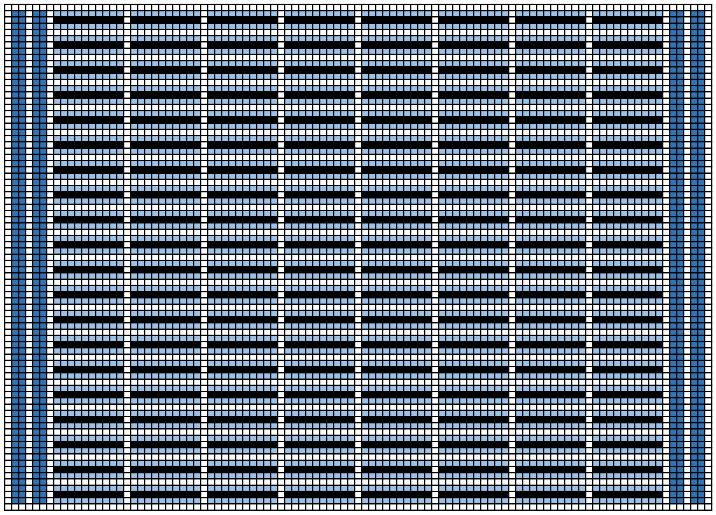

Figure 11. Env. 5: Well-formed MAPD (large-scale) [17].

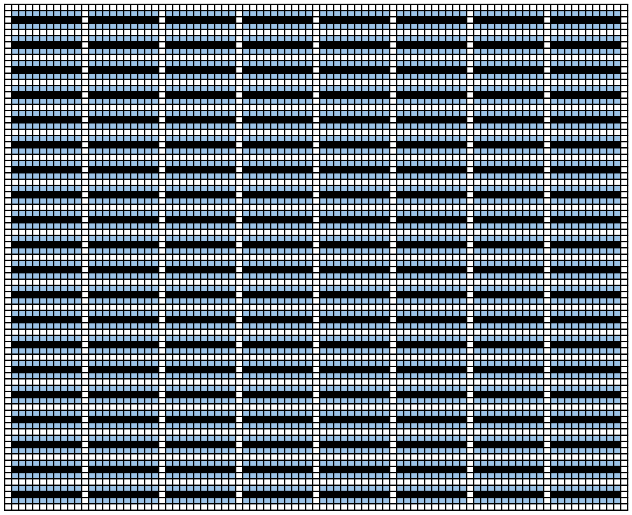

Figure 12. Env. 6: MAPD without non-task EPs (large-scale).

The parameters of the environments are summarized in Table 4.

Table 4. Parameters of environments.

Env.	Width	Height	#EP			
			Non-Task	Task	Incoming/Outgoing	Storing
1	35	21	152	200	-	-
2	23	21	0	200	-	-
3	25	25	40	238	19	200
4	25	21	0	238	19	200
5	101	81	632	3200	-	-
6	89	81	0	3200	-	-

We varied the number of agents and the task settings. We set the maximum number of agents depending on the number of non-task EPs for well-formed problems and the number of EPs for non-well-formed problems. For the task settings, we selected the total numbers of generated tasks, the number of tasks generated for each time step, and the candidates of the pickup-and-delivery locations depending on the types of maps. At every time step, NpT tasks were generated, up to the defined total maximum number of tasks.

Parameter NpT affects the possibility of the concurrent execution of the tasks. The initial locations of agents were selected from the non-task EPs for the well-formed problems and all the task EPs for the non-well-formed problems. The initial locations of the agents and the pickup-and-delivery locations of the tasks were randomly selected from their corresponding cells/vertices with uniform distribution. The parameters of the tasks and the agents are summarized in Table 5.

Table 5. Parameters of tasks and agents.

Env.	#Task	Incoming/Outgoing	NpT		Max. #Agent
1	500	-	1, 10	152	(#non-task EP)
2	500	-	1, 10	199	(#EP-1)
3	500	250	1, 10	40	(#non-task EP)
4	500	250	1, 10	199	(#EP-1)
5	1000	-	50	500	
6	1000	-	50	500	

We evaluated our selected additional techniques below by incrementally combining them. The possible full combinations of them in our result are shown in Table 3. We selected a subset from the possible full combinations, as shown in the following tables of results.

- TP: our baseline implementation of TP based on the literature [17]. For our study we adjusted a few details, which introduced a relatively small bias in our results.
- Pt: for the waiting tasks in the task allocation, each agent considers the estimated pickup times for all the agents (Section 4.2).
- TeW: allows agents to travel on paths through EPs (Section 4.3). Each move from an agent's location to a neighboring EP, which is a delivery location of tasks waiting to be allocated, is weighted by penalty coefficient value W to avoid such an EP.
- Ge: generalizes non-task and task EPs and allows the elimination of non-task EPs from a map to reduce redundant spaces (Section 4.1).
- DpT-P: employs dummy retreat paths to improve the concurrency of the executing tasks (Section 4.4). Here at most T 'tasks', including a pickup-and-delivery task and subsequent retreat tasks, are allocated, and each retreat task has a path whose length is at most P.
- Dpc$T$-P: an extension of DpT-P that cancels the reserved dummy paths, if possible.
- Sg: decomposes a pickup-and-delivery task into corresponding pickup and delivery paths (Section 4.5).

We evaluated the makespan (MS) to complete all tasks, the service time (ST) to complete each task, and the computation time (CT) per time step. The units for MS and ST are logical time steps, and the units for CT are milliseconds. Note that the computation times include some disturbances in a computation environment.

The results over ten runs with the random initial positions of the agents were averaged for each problem instance. We employed a computer with g++ (GCC) 8.5.0 -O3, Linux 4.18, Intel(R) Core(TM) i7-9700K CPU @ 3.60 GHz, and 64 GB memory for the experiment. The maximum memory usage was within 220 MB for large-scale problems.

Our experimental implementation of algorithms focuses on an investigation of the simulation results and employs several statistical processing and cache data of maps. There are opportunities for improving them for practical implementations.

5.2. Results

Tables 6 and 7 show the results for Env. 1 (basic).

The best value in each setting is emphasized in bold. We discuss the effect of integrated techniques with the results. Pt based on the estimated pickup times reduced the actual pickup times when a sufficient number of agents cover a whole map without congestion, and it also reduced the makespan and the service time ($NpT = 1$, 60 agents).

Te reduced the makespan and the service time by allowing shorter paths through EPs, although it increased the number of temporarily locked EPs. In these settings, the results were identical for the cases with/without Ge because there were enough EPs for the agent to retreat to.

Dp and Dpc improved the results by increasing the concurrency of tasks in the situations where the tasks were continuously assigned to a relatively large number of agents ($NpT = 10$, 60 and 152 agents).

Table 6. Env. 1 (basic, with non-task EP, $NpT = 1$).

#Agent	10			30			60			152		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
TP	1188.4	322.9	**0.026**	613.3	62.1	**0.058**	631.3	75.0	**0.101**	633.5	75.2	**0.179**
Pt	1204.4	328.0	0.044	611.2	65.7	0.077	565.1	43.5	0.133	554	37.0	0.653
PtTe3	1082.7	273.5	0.055	567.3	42.4	0.090	556.1	35.6	0.150	550.4	30.9	0.593
PtGe	1204.4	328.0	0.047	611.2	65.7	0.080	565.1	43.5	0.136	554	37.0	0.651
PtTe3Ge	1082.7	273.5	0.054	567.3	42.4	0.087	556.1	35.6	0.148	550.4	30.9	0.590
+Dp2-100	1136.3	303.1	0.067	558.1	39.7	0.104	542.5	28.7	0.204	540.2	25.9	0.959
+Dpc2-100	**1070.7**	**268.5**	0.082	**551.6**	**33.6**	0.129	**539.6**	**28.0**	0.211	**537.3**	**25.8**	0.783
PtTe3GeSg	1129.1	293.1	0.074	571.3	43.9	0.112	557.6	36.0	0.177	549.6	32.1	0.600

Table 7. Env. 1 (basic, with non-task EP, $NpT = 10$).

#Agent	10			30			60			152		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
TP	1164.3	521.4	**0.030**	530.4	219.3	**0.089**	429.8	159.0	**0.147**	495.0	194.2	**0.401**
Pt	1177.5	525.7	0.061	529.3	225.7	0.243	399.3	159.4	0.366	388.5	156.0	0.862
PtTe3	1061.6	475.7	0.075	443.7	180.0	0.298	303.3	112.1	0.453	283.7	100.7	1.185
PtGe	1177.5	525.7	0.066	529.3	225.7	0.292	399.3	159.4	0.402	388.5	156.0	0.861
PtTe3Ge	1061.6	475.7	0.074	443.7	180.0	0.295	303.3	112.1	0.451	283.7	100.7	1.169
+Dp2-100	1222.0	586.6	0.086	508.3	231.3	0.322	341.3	149.4	0.542	287.9	109.4	15.260
+Dpc2-100	**1053.9**	**475.1**	0.117	**429.4**	**176.3**	0.492	**267.3**	**101.0**	1.080	**245.3**	**85.6**	18.858
PtTe3GeSg	1106.7	498.6	0.138	466.4	190.7	0.424	327.4	119.1	0.515	289.1	109.0	1.649

For such problems, Dp and Dpc were relatively competitive with PIBT (Table 2 in a previous work [12]).

In these cases, SG completed all the tasks without deadlock situations, although the makespan and the service time slightly increased in most results. This overhead was caused by the waiting time for the task allocations divided into subgoals as well as some increments of the length of the divided paths.

Tables 8 and 9 show the results with different parameter settings. For weight parameter W imposed on the distance to the EPs with waiting tasks, a relatively small value achieved a good trade-off between the avoidance of such EPs and the total length of the detour route.

To avoid excessive locking of EPs by dummy paths, appropriate settings appeared to emerge for the maximum number of tasks T, including dummy retreat tasks, and the maximum length of dummy paths P. In these results, a smaller value of T and a sufficiently larger value of P were relatively better.

Table 8. Env. 1 (basic, with non-task EP).

NpT	1						10					
#Agent	**10**		**30**		**60**		**10**		**30**		**60**	
Alg.	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**
PtTe1	**1046.6**	**254.8**	595.1	53.5	568.3	40.5	**1009.2**	**445.0**	489.7	182.1	406.4	141.2
PtTe3	1082.7	273.5	567.3	42.4	556.1	35.6	1061.6	475.7	443.7	**180.0**	303.3	**112.1**
PtTe5	1111.5	286.6	562	**41.6**	**547**	**34.6**	1092.9	491.3	**443.4**	184.5	**298.6**	115.7
PtTe10	1117.6	291.1	**561.4**	42.1	552.4	34.7	1102.1	498.6	462.5	192.7	317.6	121.3

Table 9. Env. 1 (basic, with non-task EP).

NpT	1						10					
#Agent	**10**		**30**		**60**		**10**		**30**		**60**	
Alg. PtTe3Ge	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**	**MS**	**ST**
+Dpc2-100	**1070.7**	268.5	**551.6**	**33.6**	**539.6**	**27.99**	1053.9	**475.1**	429.4	**176.3**	**267.3**	**101.0**
+Dpc2-1	1078.9	271.1	562.1	40.6	548.8	33.81	**1051.3**	475.5	434	178.4	289	113.2
+Dpc5-3	1071.4	**268.1**	552.1	33.7	540.9	28.03	1053.2	476.7	**420.1**	177.9	279.4	108.4

Tables 10 and 11 show the result for Env. 2. The absence of non-task EPs in this setting increased the trade-offs between the effect and overhead in the additional methods when the population density of the agents is relatively high. When the number of agents was not excessive, a solution method based on Ge competed with those in similar problems with non-task EPs.

Table 10. Env. 2 (basic, without non-task EP, $NpT = 1$).

#Agent	**10**			**30**			**60**			**199**		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
PtGe	1201.1	319.7	**0.036**	606.2	60.6	**0.058**	555.8	34.7	**0.092**	1604.6	523.4	**4.228**
PtTe3Ge	1082.1	270.7	0.045	570.9	41.0	0.075	550.4	29.2	0.112	**1597.7**	**520.9**	4.534
+Dp2-100	1129.1	297.0	0.054	560.4	39.0	0.085	536.8	25.6	0.148	1646.3	535.5	23.805
+Dpc2-100	**1064.5**	**263.0**	0.063	**550.2**	**32.9**	0.098	**536.3**	**25.5**	0.150	1646.3	535.5	23.705
PtTe3GeSg	1124.9	288.2	0.064	575.1	42.3	0.091	548.6	30.4	0.127	1940.7	678.6	4.796

Table 11. Env. 2 (basic, without non-task EP, $NpT = 10$).

#Agent	**10**			**30**			**60**			**199**		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
PtGe	1166.9	516.4	**0.053**	516.5	213.0	**0.239**	395.7	146.0	**0.318**	1520.4	664.7	**6.215**
PtTe3Ge	1058.5	472.8	0.064	429.3	**172.3**	0.263	308.9	108.2	0.387	**1503.4**	**658.7**	6.642
+Dp2-100	1230.9	584.7	0.071	525.6	237.3	0.282	331.8	136.5	0.547	1533.4	667.8	32.361
+Dpc2-100	**1050.9**	**471.7**	0.093	**424**	173.3	0.431	**284.2**	**105.4**	0.858	1533.4	667.8	32.061
PtTe3GeSg	1105.6	496.3	0.129	457.9	183.2	0.386	313.8	114.5	0.445	1812.7	819.7	6.670

Although Dp and Dpc reduced the makespan and the service time in appropriate settings, their overhead also increased with the number of agents. Particularly in the excessively dense case of 199 agents, there was almost no room for the effect to reduce makespan and service times by the methods.

Tables 12 and 13 show the result for Env. 3 (incoming–storing–outgoing). Although this environment resembles Env. 2, there are incoming and outgoing EPs on both the left

and right sides. Non-task EPs for well-formed problems are placed in the top and bottom areas. One endpoint of each task is an incoming or outgoing EP, and this situation relatively lengthens the travel paths of the agents.

In this case, the combinations of Pt, Te, Ge, and Dp/Dpc well reduced the makespan and the service time. PtTeGeSg slightly reduced the makespan more than its baseline method PtTeGe in the case of $NpT = 1$ and 40 agents, while there is still some overhead in service times. Although this number of agents is maximum (i.e., the number of non-task EPs), below we confirm a similar result in an environment without non-task EPs.

Table 12. Env. 3 (incoming–storing–outgoing, with non-task EP, $NpT = 1$).

#Agent	10			20			30			40		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
TP	2008.6	847.1	**0.016**	1171.4	445.0	**0.032**	1028.4	282.9	**0.042**	1026.8	216.4	**0.047**
Pt	2040.9	858.6	0.023	1166.6	435.8	0.055	903.2	274.3	0.084	922.2	213.7	0.089
PtTe3	1838.9	757.2	0.029	992.2	351.5	0.067	768.5	209.0	0.097	781.9	146.8	0.089
PtGe	2040.9	858.6	0.025	1166.6	435.8	0.057	903.2	274.3	0.088	922.2	213.7	0.092
PtTe3Ge	1838.9	757.2	0.028	992.2	351.5	0.066	768.5	209.0	0.095	781.9	146.8	0.089
+Dp2-100	1904	801.4	0.041	1015.6	373.7	0.090	699.4	217.4	0.136	538.8	146.6	0.164
+Dpc2-100	**1821.4**	**749.5**	0.050	**966.3**	**344.6**	0.116	**676.8**	**208.9**	0.192	**531.0**	**137.8**	0.261
PtTe3GeSg	1906.5	769.5	0.053	1072.7	359.4	0.091	812.4	221.6	0.111	760.5	164.9	0.116

Table 13. Env. 3 (incoming–storing–outgoing, with non-task EP, $NpT = 10$).

#Agent	10			20			30			40		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
TP	1999.8	946.4	**0.016**	1153.8	543.5	**0.033**	983.3	372.0	**0.045**	1020.6	308.5	**0.050**
Pt	2039.1	957.5	0.025	1172.9	545.0	0.061	909.9	380.7	0.098	910.9	311.9	0.116
PtTe3	1831.5	858.8	0.031	999.1	459.7	0.076	754.7	**310.0**	0.120	745.2	**242.9**	0.133
PtGe	2039.1	957.5	0.027	1172.9	545.0	0.064	909.9	380.7	0.103	910.9	311.9	0.120
PtTe3Ge	1831.5	858.8	0.031	999.1	459.7	0.074	754.7	**310.0**	0.120	745.2	**242.9**	0.133
+Dp2-100	1914.4	915.8	0.044	1024.7	489.7	0.101	709.1	336.1	0.168	545	258.9	0.226
+Dpc2-100	**1820.9**	**853.6**	0.055	**963.4**	**450.6**	0.146	**676.6**	313.6	0.276	**528.6**	244.3	0.426
PtTe3GeSg	1894.2	873.3	0.067	1056.5	468.6	0.122	774.3	325.7	0.152	748.4	264.2	0.147

Tables 14 and 15 show the result for Env. 4. Although the non-task EPs were eliminated from the previous setting in this environment, the result resembles that of Env. 3.

In this result, PtTeGeSg slightly reduced the makespan more than PtTeGe in the case of sixty agents. We saw such a trend in the cases of 40 and 50 agents. Since these problem settings often generate long travel paths of agents in opposite directions, the effect of Sg appeared to be revealed in the cases of relatively dense populations of agents. However, no such result surfaced in the extremely dense case (199 agents).

Table 14. Env. 4 (incoming–storing–outgoing, without non-task EP, $NpT = 1$).

#Agent	10			30			60			199		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
PtGe	2052.6	860.1	**0.021**	915.2	277.4	**0.074**	944.2	183.49	**0.089**	728.5	159.4	**0.842**
PtTe3Ge	1833.6	753.4	0.026	764.3	**201.4**	0.087	778.4	109.92	0.082	626.6	**105.2**	0.827
+Dp2-100	1909.2	798.5	0.037	692.2	215.8	0.125	406.3	79.12	0.185	587.7	159.3	15.970
+Dpc2-100	**1824.2**	**745.6**	0.046	**671.6**	203.6	0.182	**395**	**69.88**	0.294	**568.5**	147.8	16.770
PtTe3GeSg	1896	763.8	0.051	798.2	217.9	0.102	722.5	143.05	0.192	747.3	168.4	1.452

Table 15. Env. 4 (incoming–storing–outgoing, without non-task EP, $NpT = 10$).

#Agent	10			30			60			199		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
PtGe	2039.2	955.8	**0.023**	915.2	379.9	**0.091**	928.2	278.0	**0.126**	783.5	285.1	**1.442**
PtTe3Ge	1824.5	853.2	0.029	754.2	**306.8**	0.112	757.7	190.5	0.135	656.2	**205.4**	1.614
+Dp2-100	1912.3	909.1	0.040	704.2	333.7	0.158	407.5	189.5	0.322	544.2	240.1	27.406
+Dpc2-100	**1813.8**	**849.6**	0.051	**677.3**	312.0	0.262	**383.5**	**167.6**	0.682	**544.7**	238.1	29.549
PtTe3GeSg	1896.8	867.7	0.062	776.6	320.9	0.137	728.8	235.2	0.250	764	295.9	3.421

Tables 16 and 17 show the result for Envs. 5 and 6 (large-scale). The populations of the agents were relatively sparse, and the effect of the additional methods was also relatively small. However, a combination of additional methods reduced the makespan and the service time, particularly for 300 and 500 agents. In the latter case, PtTeGeSg slightly reduced the makespan more than PtTeGe in average.

Table 16. Env. 5 (large-scale, with non-task EP, $NpT = 50$).

#Agent	100			300			500		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**
TP	923.5	405.2	**0.884**	584.8	214.2	**2.142**	596.5	194.0	**3.164**
Pt	981.8	449.2	4.956	535.8	231.2	17.998	493	196.8	18.505
PtTe3	920.9	407.4	5.237	488.3	199.5	15.873	468.6	175.6	36.748
PtGe	981.8	449.2	6.000	535.8	231.2	22.064	493	196.8	20.991
PtTe3Ge	920.9	407.4	5.185	488.3	199.5	15.795	468.6	175.6	35.248
+Dp2-100	910.7	407.2	5.703	469	198.3	14.286	404.6	157.8	30.813
+Dpc2-100	**896.6**	**399.5**	7.019	**426.9**	**179.3**	31.415	**367.4**	**140.2**	44.893
PtTe3GeSg	933	412.6	7.188	492	201.2	19.832	445.2	169.3	26.352

Table 17. Env. 6 (large-scale, without non-task EP, $NpT = 50$).

#Agent	100			300			500		
Alg.	**MS**	**ST**	**CT**	**MS**	**ST**	**CT**	MS	**ST**	**CT**
PtGe	913.8	397.9	**4.622**	489.8	176.5	18.496	421.9	137.0	22.550
PtTe3Ge	877.9	379.0	4.777	454.9	159.9	**14.704**	412.2	119.0	20.364
+Dp2-100	883.6	379.8	5.240	428.5	160.5	16.794	346.3	115.7	35.655
+Dpc2-100	**856.6**	**371.4**	6.279	**403.7**	**148.9**	26.292	**327.6**	**108.7**	36.124
PtTe3GeSg	890.6	384.4	6.862	444.4	162.3	18.403	399.6	122.1	**19.740**

A major issue here is the overhead of the additional methods in the computation times, although there are chances to improve our experimental implementation, as discussed in Section 5.1. One such overhead comes from our implementation of Pt that considers the estimated pickup times of the other agents in the task allocation. Since our current version evaluates all the agents and all the allocatable tasks in the task allocation process, it was affected by the numbers of both agents and allocatable tasks.

Another overhead was due to the canceling and restoring of the dummy paths in Dpc. Our current version searches for the possibility of task allocation with the pathfinding of each task's path after a dummy path of an agent is temporarily canceled. The dummy path is restored if the task allocation fails. Our current version was also affected by the number of agents and the size of the environments. In practical cases, a large warehouse's map might be divided into sub-areas to mitigate the computational cost of the solution methods.

In summary, we found that the redundancy in well-formed problems and solution methods based on EPs can be reduced by appropriately combining and setting our selected additional techniques.

1. Pt was effective in situations where a certain number of tasks to be assigned were waiting in an environment with a sufficient number of agents.
2. Te*W* was effective in most situations, although there were some trade-offs regarding the number of locked/unlocked EPs.
3. Ge was a useful option that reduced the number of non-task EPs.
4. Dp*T*-*P*/Dpc*T*-*P* was effective in situations with a sufficient number of free EPs to which agents can additionally retreat.
5. Although opportunities remain for improving the current SG to reduce redundant waits for the decomposed task assignments, some results revealed the effectiveness of this technique.
6. The incremental combinations of Pt, Te, and Dp/DPc were basically effective. However, there are inherent trade-offs among the methods, especially in situations where the population density of the agents was high and the number of free EPs was relatively small.

Although Dp and Dpc required relatively large computational cost to manage the dummy tasks/paths, the experimental implementation of the methods can be improved. Additionally, some parts of the computation can be performed while the agents are on their journey, and their moves typically take a longer time than the computation.

Figures 13–15 show the makespans in a major part of our result. We extracted them from the cases with non-task EPs, to include the base TP algorithm. Although the result revealed several trade-offs, a full combination '+Dpc2-100' was effective with sufficient numbers of agents.

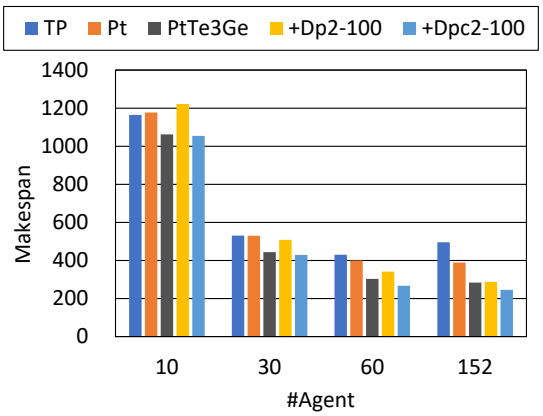

Figure 13. Env. 1 (basic, with non-task EP, NpT = 10).

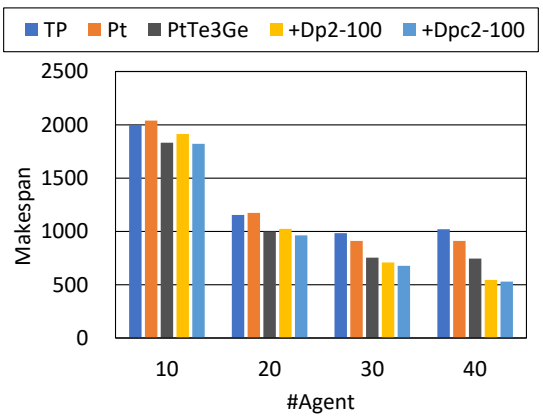

Figure 14. Env. 3 (incoming–storing–outgoing, with non-task EP, NpT = 10).

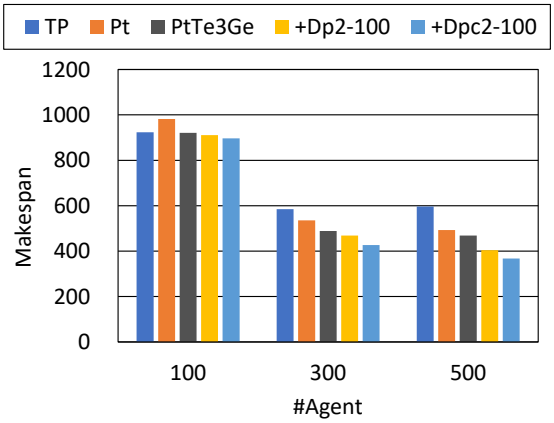

Figure 15. Env. 5 (large-scale, with non-task EP, NpT = 50).

6. Discussion

The effects of the additional integrated methods are basically complementary. The Pt method that considers the estimated pickup times affects the task allocation process. Te, which allows travel paths of agents to contain arbitrary EPs, reduces the path length. Dp employs additional dummy paths and resolves the conflict among an end of an agent's path and a delivery location of a task waiting to be allocated. Ge, which accepts map setting without non-task EPs, reduces the space of maps, and Dpc, which conditionally cancels dummy paths, is particularly important in such narrow maps. Regarding Te and Dp, an agent avoids the delivery locations of new tasks in its pathfinding process with our improved Te, and Dp resolves such conflicts after the agent's path is reserved.

We describe some directions to enhance the employed techniques. Our Pt method that considers the estimated pickup times in the task allocation process currently evaluates all the agents and all the allocatable tasks; it requires relatively large computational cost for large-scale problems. Some elements of the agents and tasks that are evaluated in the process can be eliminated considering the areas in a large map.

For the Te method that allows the travel paths of agents to contain arbitrary EPs, we introduced weight values that are added to the distance values in the pathfinding to avoid the delivery locations of new tasks, and the experimental result revealed the room to adjust the weight parameter for situations (Table 8). The insertion of such detour paths might be controlled by evaluating the density of idle agents in local areas in a certain period of time.

Our current Sg method, which divides a task's path was simply applied to all the tasks as an investigation to integrate it with several heuristic add-on methods. To bring out its effect, the division could be conditionally applied to tasks whose pickup-and-delivery locations are separated areas in a map, while the integration of different task allocation rules is necessary for decomposed and non-decomposed tasks.

The solution methods based on TP complete all the tasks generated on demand if a finite number of tasks are generated. However, there is an issue of fairness among tasks waiting to be allocated, and a starvation situation can be a problem in the task allocation. This drawback can be addressed with an approach using the threshold or the priority values to consider the waiting times of tasks. Generally, there is a trade-off between efficiency and fairness. We mainly focused on efficiency in the makespan to complete all the tasks.

Although we employed several techniques in the solution methods to partially relax the conditions of the well-formed MAPD problems, the maps of the warehouses themselves remain restricted by some fundamental conditions regarding endpoints. The solution methods still depend on the conditions as a safeguard. The challenges of mitigating the limitations of maps will require greater consideration of the conditions for avoiding deadlock situations within time windows.

In summary, excluding an experimental version of subgoal division of each task, the full combination of our selected techniques with several improvements resulted in the best performance in most cases in the sense of makespan. We recommend this solution

method in our evaluated methods except for excessively dense agent populations. When computational overhead is an issue due to dense agent populations, a technique with dummy paths can be omitted in turn for trade-offs with other performances.

7. Conclusions

We investigated the integration of several additional efficient techniques that improve the solution methods for the lifelong multiagent pickup-and-delivery (MAPD) problem by considering endpoints. Although these additional techniques have been proposed to reduce some redundancies in the problem settings themselves and the concurrency of allocated tasks, there were opportunities to combine the techniques with improving them for further practical solution methods. For an analysis and better understanding of the additional solution techniques based on endpoints, we incrementally integrated the techniques and experimentally investigated their contributions to the quality of task allocation and the paths of the agents.

Our result revealed significant complementary effects of the integrated additional techniques and the trade-offs among them in several cases of problem settings. A full combination of our selected techniques with several improvements, excluding an experimental version, resulted in the best performance in most cases for the sense of makespan. We recommend this solution method in our evaluated methods except in the case of excessively dense agent populations. We believe that our experimental analysis will contribute to further development of efficient solution methods.

Future work will include detailed analysis of the trade-offs among the integrated additional techniques, improvement of different solution methods by employing our result, and further relaxation of the settings in well-formed problems. Although we concentrated on a scalable heuristic solution method based on EPs, opportunities will exist to investigate variants of exact solution methods with our result.

Funding: This study was supported in part by The Public Foundation of Chubu Science and Technology Center (thirty-third grant for artificial intelligence research) and JSPS KAKENHI (grant number 22H03647).

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Veloso, M.; Biswas, J.; Coltin, B.; Rosenthal, S. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015; pp. 4423–4429.
2. Wurman, P.R.; D'Andrea, R.; Mountz, M. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Mag.* **2008**, *29*, 9–19.
3. Hönig, W.; Preiss, J.A.; Kumar, T.K.S.; Sukhatme, G.S.; Ayanian, N. Trajectory Planning for Quadrotor Swarms. *IEEE Trans. Robot.* **2018**, *34*, 856–869. [CrossRef]
4. Morris, R.; Pasareanu, C.S.; Luckow, K.S.; Malik, W.; Ma, H.; Kumar, T.S.; Koenig, S. Planning, Scheduling and Monitoring for Airport Surface Operations. In Proceedings of the Workshops of the Thirtieth AAAI Conference on Artificial Intelligence Planning for Hybrid Systems: Technical Report WS-16-12, Phoenix, AZ, USA, 12–13 February 2016; pp. 608–614.
5. Wen, L.; Liu, Y.; Li, H. CL-MAPF: Multi-Agent Path Finding for Car-Like robots with kinematic and spatiotemporal constraints. *Robot. Auton. Syst.* **2022**, *150*, 103997. [CrossRef]
6. Silver, D. Cooperative Pathfinding. In Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina del Rey, CA, USA, 1–3 June 2005; pp. 117–122.
7. Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N.R. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artif. Intell.* **2015**, *219*, 40–66. [CrossRef]
8. Ma, H.; Harabor, D.; Stuckey, P.J.; Li, J.; Koenig, S. Searching with Consistent Prioritization for Multi-Agent Path Finding. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 7643–7650.
9. Barer, M.; Sharon, G.; Stern, R.; Felner, A. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In Proceedings of the Annual Symposium on Combinatorial Search (SoCS), Prague, Czech Republic, 15–17 August 2014; pp. 19–27.

10. Hart, P., N.N.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
11. Hart, P., N.N.; Raphael, B. Correction to 'A Formal Basis for the Heuristic Determination of Minimum-Cost Paths'. *SIGART Newsletter* **1972**, *37*, 28–29.
12. Okumura, K.; Machida, M.; Défago, X.; Tamura, Y. Priority Inheritance with Backtracking for Iterative Multi-Agent Path Finding. *Artif. Intell.* **2022**, *310*, 103752. [CrossRef]
13. Okumura, K.; Tamura, Y.; Défago, X. winPIBT: Expanded Prioritized Algorithm for Iterative Multi-agent Path Finding. *arXiv* **2019**, arXiv:1905.10149.
14. Dantzig, G.B.; Ramser, J.H. The Truck Dispatching Problem. *Manag. Sci.* **1959**, *6*, 80–91. [CrossRef]
15. Bi, H.; Zhu, X.; Lu, F.; Huang, M. The Meal Delivery Routing Problem in E-commerce Platforms under the Shared Logistics Mode. *J. Theor. Appl. Electron. Commer. Res.* **2023**, *18*, 1799–1819. [CrossRef]
16. Liu, M.; Ma, H.; Li, J.; Koenig, S. Task and Path Planning for Multi-Agent Pickup and Delivery. In Proceedings of the Eighteenth International Conference on Autonomous Agents and Multiagent Systems, Montreal, QC, Canada, 13–17 May 2019; pp. 1152–1160.
17. Ma, H.; Li, J.; Kumar, T.S.; Koenig, S. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In Proceedings of the Sixteenth Conference on Autonomous Agents and Multiagent Systems, Sao Paulo, Brazil, 8–12 May 2017; pp. 837–845.
18. Grenouilleau, F.; van Hoeve, W.J.; Hooker, J. A Multi-Label A* Algorithm for Multi-Agent Pathfinding. In Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, Berkeley, CA, USA, 11–15 July 2019; pp. 181–185.
19. Yamauchi, T.; Miyashita, Y.; Sugawara, T. Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks. In Proceedings of the Twenty-First International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, 9–13 May 2022; pp. 1427–1435.
20. Matsui, T. Investigation of Integrating Solution Techniques for Lifelong MAPD Problem Considering Endpoints. In Proceedings of the Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection: 21st International Conference, Guimarães, Portugal, 12–14 July 2023; pp. 175–187.
21. Surynek, P.; Felner, A.; Stern, R.; Boyarski, E. Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective. In Proceedings of the Twenty-Second European Conference on Artificial Intelligence, The Hague, The Netherlands, 29 August–2 September 2016; pp. 810–818.
22. Čapek, M.; Surynek, P. DPLL(MAPF): An Integration of Multi-Agent Path Finding and SAT Solving Technologies. In Proceedings of the Fourteenth International Symposium on Combinatorial Search, Guangzhou, China, 26–30 July 2021; Volume 12, pp. 153–155.
23. Gómez, R.N.; Hernández, C.; Baier, J.A. A Compact Answer Set Programming Encoding of Multi-Agent Pathfinding. *IEEE Access* **2021**, *9*, 26886–26901. [CrossRef]
24. De Wilde, B.; Ter Mors, A.W.; Witteveen, C. Push and Rotate: A Complete Multi-Agent Pathfinding Algorithm. *J. Artif. Int. Res.* **2014**, *51*, 443–492. [CrossRef]
25. Luna, R.; Bekris, K.E. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain, 16–22 July 2011; Volume 1, pp. 294–300.
26. Ma, H.; Kumar, T.K.S.; Koenig, S. Multi-Agent Path Finding with Delay Probabilities. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 3605–3612.
27. Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Bartak, R.; Zhou, N.F. Robust Multi-Agent Path Finding. In Proceedings of the Eleventh Annual Symposium on Combinatorial Search, Stockholm, Sweden, 14–15 July 2018; Volume 9, pp. 2–9.
28. Andreychuk, A.; Yakovlev, K.; Surynek, P.; Atzmon, D.; Stern, R. Multi-agent pathfinding with continuous time. *Artif. Intell.* **2022**, *305*, 103662. [CrossRef]
29. Andreychuk, A.; Yakovlev, K.; Boyarski, E.; Stern, R. Improving Continuous-time Conflict Based Search. In Proceedings of the Proceedings of The Thirty-Fifth AAAI Conference on Artificial Intelligence, Held Virtually, 2–9 February 2021; Volume 35, pp. 11220–11227.
30. Miyashita, Y.; Yamauchi, T.; Sugawara, T. Distributed Planning with Asynchronous Execution with Local Navigation for Multi-Agent Pickup and Delivery Problem. In Proceedings of the Twenty-Second International Conference on Autonomous Agents and Multiagent Systems, London, UK, 29 May–2 June 2023; pp. 914–922.
31. Yakovlev, K.S.; Andreychuk, A. Any-Angle Pathfinding for Multiple Agents Based on SIPP Algorithm. In Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, Pittsburgh, PA, USA, 18–23 June 2017; pp. 586–593.
32. Atzmon, D.; Diei, A.; Rave, D. Multi-Train Path Finding. In Proceedings of the Twelfth Annual Symposium on Combinatorial Search, Napa, CA, USA, 16–17 July 2019; Volume 10, pp. 125–129.
33. Gregory Gutin, A.P.P. (Ed.) *The Traveling Salesman Problem and Its Variations*, 1st ed.; Combinatorial Optimization; Springer-Verlag: New York, NY, USA, 2007.
34. Pop, P.C.; Cosma, O.; Sabo, C.; Sitar, C.P. A comprehensive survey on the generalized traveling salesman problem. *Eur. J. Oper. Res.* **2024**, *314*, 819–835. [CrossRef]
35. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer Networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; Volume 28.

36. Mele, U.J.; Gambardella, L.M.; Montemanni, R. A New Constructive Heuristic Driven by Machine Learning for the Traveling Salesman Problem. *Algorithms* **2021**, *14*, 267. [CrossRef]
37. Hu, Y.; Yao, Y.; Lee, W.S. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowl.-Based Syst.* **2020**, *204*, 106244. [CrossRef]
38. Cheikhrouhou, O.; Khoufi, I. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. *Comput. Sci. Rev.* **2021**, *40*, 100369. [CrossRef]
39. Martí, P.; Jordán, J.; Arrieta, M.A.G.; Julián, V. A Survey on Demand-Responsive Transportation for Rural and Interurban Mobility. *Int. J. Interact. Multim. Artif. Intell.* **2023**, *8*, 43. [CrossRef]
40. Čáp, M.; Vokřínek, J.; Kleiner, A. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-Formed Infrastructures. In Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, Jerusalem, Israel, 7–11 June 2015; pp. 324–332.
41. Shimokawa, M.; Matsui, T. Improvement of Task Allocation in TP Algorithm for MAPD Problem by Relaxation of Movement Limitation and Estimation of Pickup Time. *Trans. Jpn. Soc. Artif. Intell.* **2022**, *37*, A-L84. [CrossRef]
42. Li, J.; Tinka, A.; Kiesel, S.; Durham, J.W.; Kumar, T.K.S.; Koenig, S. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, Held Virtually, 2–9 February 2021; pp. 11272–11281.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.