Improvement of PIBT-based Solution Method for Lifelong MAPD Problems to Extend Applicable Graphs

Toshihiro Matsui[a]
Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya Aichi 466-8555, Japan

Keywords: Multiagent Pathfinding Problem, Lifelong Multiagent Pickup-and-Delivery Problem, PIBT, Swap Operation.

Abstract: We address an extension of priority inheritance with backtracking (PIBT) for lifelong multiagent pickup-and-delivery (MAPD) problems that performs a swap operation integrated into the original algorithm to adapt specific extended case problems. The multiagent pathfinding (MAPF) problem has been widely studied as a basis for various practical multiagent systems. PIBT is a scalable and on-demand solution method for continuous MAPF problems, where each agent determines its next move in each time step by locally solving agent-move collisions. Since it can be applied to limited cases such as biconnected graphs, several extensions using additional techniques have been suggested. However, there are opportunities to extend the PIBT process with several techniques that can be integrated into the solution process itself. As the first step, we extend a solution method based on PIBT for lifelong MAPD problems, fundamental continuous problems, by integrating a specific swap task. We address detailed techniques, including additional management of priorities, subgoals, and states of agents. We also experimentally evaluate the proposed approach with several problem settings.

1 INTRODUCTION

We address an extension of priority inheritance with backtracking (PIBT) (Okumura et al., 2022; Okumura et al., 2019) for lifelong multiagent pickup-and-delivery (MAPD) problems that performs a swap operation integrated into the original algorithm to adapt specific extended case problems. The multiagent pathfinding (MAPF) problem has been widely studied as a basis for various practical multiagent systems, including robot navigation, autonomous carriers in warehouses and construction sites, autonomous taxiing of airplanes and video games (Ma et al., 2017). This problem is a combinatorial optimization problem finding a set of agents' paths, where all the agents must move from their start locations to their goal locations without colliding with each other. The set of paths should be minimized by optimization criteria.

Several types of solution methods for MAPF problems, including optimal and quasi-optimal methods, have been developed. A major optimal approach is based on variants of Conflict Based Search (Sharon et al., 2015), which performs two layers of search. There are several optimal and quasi-optimal extended variations (Ma et al., 2019; Barer et al., 2014) that address the mitigation of the relatively high computational cost of the optimal search method.

A different greedy approach individually finds and reserves the single quasi-optimal path in a time-space graph for each agent according to an order on all the agents (Silver, 2005). There are also different approaches, including push, swap, and rotate operations among agents (De Wilde et al., 2014; Luna and Bekris, 2011), and general optimization methods.

The MAPF problem has been extended to the continuous MAPF problem, where each agent updates its sequence of subgoals, and a MAPF method is repeatedly performed for the sequences. The lifelong multiagent MAPD problem is an important class of continuous MAPF problems, where each agent repeatedly performs pick-up and delivery tasks (Ma et al., 2017). While a scalable quasi-optimal approach for this problem is based on a theorem regarding the endpoints of agents' paths (Ma et al., 2017), there are several challenges to improving the performance of solution methods (Li et al., 2021; Yamauchi et al., 2022).

We focus on PIBT that is a solution method for continuous MAPF problems, where each agent determines its next move in each time step by locally solving collisions of agents' moves. PIBT performs a management of priorities of agents and a dedicated back-tracking method. Although it can be applied to

[a] https://orcid.org/0000-0001-8557-8167

Matsui, T.
Improvement of PIBT-based Solution Method for Lifelong MAPD Problems to Extend Applicable Graphs.
DOI: 10.5220/0013243800003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 1*, pages 123-134
ISBN: 978-989-758-737-5; ISSN: 2184-433X

limited cases such as biconnected graphs, the method can work with narrow aisles and dense populations of agents. There are several extensions of PIBT using additional techniques (Okumura et al., 2019), including methods addressing more general cases of graphs (Okumura et al., 2022; Okumura, 2023). However, these methods basically employ external extensions where PIBT can be considered a module. There are opportunities to extend the PIBT with techniques that can be integrated into the solution process itself.

This consideration is important to understand the detailed properties of the original solution method and to uncover some informative insights to improve the solution method or some heuristics. As the first step, we extend a solution method based on PIBT for lifelong MAPD problems, fundamental continuous problems, by integrating a specific swap task.

We add a high-level layer of tasks to PIBT to manage individual cooperation tasks of groups of agents. Namely, we employ PIBT as a processing engine and introduce a context of a swap task of an individual group of agents. The tasks are independently constructed in a bottom-up manner, and their conflict situations are solved using their priority values. As the first study, we present the swap tasks of agents for a class of problems that can be naturally extended from that for the original PIBT. The approach to execute such bottom-up tasks of agents on a fundamental solution method as an engine is the major aim of this study.

We address detailed techniques, including additional management of priorities, subgoals and states of agents. We also experimentally evaluate the proposed approach with several problem settings.

In the next section, we present the background of our study, including multiagent pathfinding problems, lifelong pickup-and-delivery problems, and the solution method PIBT. The details of our proposed approaches are described in Section 3. We first consider some important graph structures of maps and then establish a set of operations regarding specific swap tasks to address dead-end aisles. We experimentally verify our approach in Section 4 and conclude in Section 5.

2 BACKGROUND

We note that several segments in the following subsections are based on the literature (Matsui, 2024b) with the same background in part, although the aim of this study is completely different from the previous work.

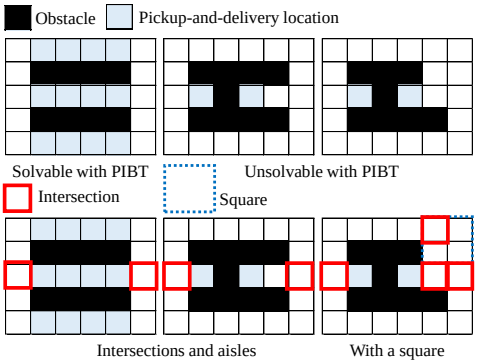


Figure 1: Warehouse map and decomposed structures.

2.1 MAPF and Lifelong MAPD

The multiagent pathfinding (MAPF) problem is an optimization problem for finding a set of paths of multiple agents where there are no collisions between the paths. A MAPF problem consists of a graph $G = (V, E)$ representing a two-dimensional map, a set of agents $\mathcal{A}$, and a set of pairs of vertices that represent the start and goal locations for individual agents. All agents must move from their start locations to their goal locations without colliding with each other, and the set of agents' paths, including stay/wait actions, should be minimized by optimization criteria. There are two types of collision paths to be avoided; two agents must not stay at the same location at the same time (a vertex collision) and must not move on the same edge at the same time from both ends of the edge (a swapping collision). In a fundamental setting, a graph representing a four-connected grid-like map containing obstacles is employed, and time steps are discrete. The continuous MAPF problem is an extended class of MAPF problems where each agent updates its sequence of subgoals, and a solution method for MAPF is repeatedly performed for the sequences.

The lifelong multiagent pickup-and-delivery (MAPD) problem (Ma et al., 2017) is a specific class of continuous MAPF problems, where multiple pickup-and-delivery tasks are repeatedly allocated to agents. Figure 1 shows examples of warehouse maps containing pickup-and-delivery locations. The tasks can be repeatedly generated in arbitrary time steps. A set of currently generated tasks is denoted by $\mathcal{T}$. Task $\tau_i \in \mathcal{T}$ has its pickup and delivery locations (s_i, g_i), where $s_i, g_i \in V$. An agent who is allocated to task τ_i first moves from its current location to pickup location s_i and then moves to delivery location g_i to complete the task. The problem consists of task allocation and continuous MAPF problems. At least partially greedy approaches are commonly employed to allocate tasks generated on demand, and MAPF

```
1  UNDECIDED← A(t) // agents list
2  OCCUPIED← ∅ // vertices list
3  update priorities p_i(t) for all agents a_i
4  while UNDECIDED≠ ∅ do
5    a ← the agent with the highest priority in UNDECIDED
6    PIBT(a,⊥) // ⊥ denotes empty
7  end while

9  function PIBT(a_i,a_j)
10   UNDECIDED←UNDECIDED\{a_i}
11   C_i ← ({v|(v_i(t),v) ∈ E} ∪ {v_i(t)})
12     \({v_j(t)}∪OCCUPIED)
13   while C_i ≠ ∅ do
14     v_i^* ← arg max_{v∈C_i} f_i(v) // most preferred move
15     OCCUPIED←OCCUPIED∪{v_i^*}
16     if a_k s.t. v_i^* = v_k(t) ∧ a_k ∈UNDECIDED exists then
17       if PIBT(a_k,a_i) is valid then
18         v_i(t+1) ← v_i^*
19         return valid // move with push
20       else
21         C_i ← C_i\OCCUPIED
22       end if
23     else
24       v_i(t+1) ← v_i^*
25       return valid // move/stay without push
26     end if
27   end while
28   v_i(t+1) ← v_i(t)
29   return invalid // stay by failing to move
30 end function
```

$v_i(t)$: location of agent a_i at time step t

Figure 2: PIBT at time step t (Okumura et al., 2022).

solvers are applied to the pathfinding.

A fundamental approach is based on the well-formed MAPD problems that take into account endpoint vertices, which can be pickup, delivery, or parking locations of agents (Čáp et al., 2015; Ma et al., 2017). However, this requires extra aisle space in maps and relatively large redundancy of parallelism on task execution including agents' movements.

We focus on a different type of solution method, PIBT (Okumura et al., 2022), that can be applied to narrow maps with dense populations of agents, although this method is also a greedy approach with several restrictions as mentioned below.

2.2 PIBT

PIBT is a scalable solution method for the (continuous) MAPF problem (Okumura et al., 2022). The method performs push operations among agents according to the priority of the agents. In each time step, each agent decides its next move/stay. When an agent cannot push other agents on all vertices neighboring its current location, a backtracking is performed to find other push chains.

In the pseudo code (Fig. 2), it is assumed that each agent a_i has its goal location, and the preference value of location v based on the goal is represented by $f_i(v)$ (line 14). The priority $p_i(t)$ of agent a_i consists of the elapsed time for the current goal and a small tie-break value based on a_i's identifier. Agent a having the locally highest priority initiates a recursive push process (line 6). Agent a_i selects its most preferred move from those remaining and *pushes* its neighboring agent to clear a vertex if necessary (lines 14-17). The pushed agent a_j tries to move to its neighboring vertex and also pushes a_j's neighboring agent if necessary. If all the agents pushed by agent a_i can move or there is no agent obstructing a_i, a chain of moves is determined (line 18). As a result, the locations of a set of agents in a cycle might rotate. If one of the pushed agents cannot move, backtracking is performed (line 29) so that its parent agent can try to move in a different direction. An agent that cannot move in this process stays in its current location (line 28).

PIBT can solve problems represented by several types of graphs, including biconnected ones, that always allow the rotation of agents' locations. The method can work with narrow aisles and dense populations of agents, even if all non-obstacle vertices are occupied by agents. However, it easily sticks in the case of maps with dead ends.

In the case of continuous problems, each agent has its list of subgoals and continues to move to the first subgoal with increasing its priority. After reaching the first subgoal, the subgoal is removed from the list and the priority of the agent is reset. For MAPD problems, we employ a baseline greedy task allocation method in which each agent having no tasks selects a task whose pickup location is nearest to its current location.

3 SPECIFIC SWAP TASK

We improve PIBT for an extended case where unbranched narrow aisles with single dead-ends (DE aisles) are added to a basic map represented by a biconnected graph that can be well handled by the original algorithm. This is the minimal extension to introduce a specific case of swap operation (De Wilde et al., 2014; Luna and Bekris, 2011) based on PIBT.

When an agent is blocking another agent in a DE aisle, both agents can retreat from the aisle to swap their locations. Specifically, in the case of PIBT, a reasonable action is to perform the swap operation of the agents on a biconnected component of a graph by simply employing PIBT itself (Fig. 5). Although this is intuitively simple, our aim is to clarify the details of several important extensions in this kind of algorithm for future study to address more general cases.

Swap task		MAPD task	Idle state
One-push sequence Move dir. const./pref.	Retreat task Subgorl: r_i	Pickup-delivery task Subgoals: (s_i, g_i)	Stay task Subgoal: cur. loc.
PIBT for continuous MAPF			

Figure 3: Tasks on PIBT.

3.1 Decomposition of Map Structure

We address the extended case where the undirected graphs of maps consist of a biconnected component and several parts of DE aisles. To concentrate on a set of important operations naturally extending PIBT to adapt to this case, we do not consider without cycles and with *isthmuses* (De Wilde et al., 2014) cases, and we will address such general cases that require several additional techniques in future study. While we employ graphs representing maps in a four-connected grid world as common settings, our approach can be extended for non-grid maps.

Except for obstacle vertices, we decompose the parts of a graph into the following structures: 1) Aisle including DE aisle, 2) Intersection vertex, including end vertices of a *square*, connecting to aisles, and 3) Other part of square (Fig. 1). Since PIBT works effectively for parts with sufficient space such as squares, we distinguish the narrow parts from others and improve the original algorithm by adding several operations that consider such narrow parts.

An aisle consists of vertices whose degrees are one (DE aisle) or two. In the case with squares, corner vertices whose degrees are two are excluded. An intersection vertex's degree is greater than two. The definition of square depends on the graphs. For a four-connected grid map, a square is a cluster of minimum cycles of the neighboring four vertices, although we distinguish intersection vertices from them.

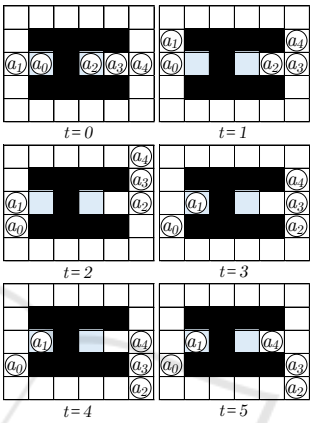
Here, we employ the following simple preprocessing to extract the parts: 1) Vertices whose degrees are one or two are marked as candidate vertices of aisles. 2) Vertices whose degrees are greater than two are marked as vertices of intersections. 3) Candidate vertices of aisles are excluded as a part of a square if they are contained in one of the minimum cycles. 4) Decomposed parts and corresponding vertices except that of squares are labeled with individual values to refer to each other in later steps. The map structure and the map data are shared by all the agents.

3.2 Integrating Specific Swap Operation

We introduce a set of operations for a specific swap task into a version of PIBT that solves lifelong MAPD problems. Since this baseline version has been integrated with a task assignment process for pickup-and-delivery tasks, we added another extension for a swap task assignment, as shown in Figs. 3-5 and Tbls. 1-3. The extended pseudo codes are shown in Fig. 9 in an

Initiator	Target	Swept	(Controller)	(Interruption)
Swap task				
(Ask) Ini-restraint (Complete)	Retreat Restraint	Swept Restraint	Initiator Target Initiator	Cancel, (one push) Cancel, (one push) Cancel
One push sequence for highest swap task only				
One push	(Ask) OP-Retreat Restraint (OP-Complete)	Swept Restraint	Target Initiator Target (To Initiator)	

Figure 4: Sub-modes and sequence in swap task.



Swap tasks

Init.	Tgt.	Swept
a_4	a_2	a_3
a_1	a_0	

$p_1(t) > p_0(t)$
$p_4(t) > p_2(t),\ p_3(t)$

Progress of a_4's

t	a_i	Sub-mode	$p_i(t)$
0	a_4	Init-Rstrnt.	$=p_2(t)$
	a_2	Retreat	$=p_4(t)$
	a_3	Swept	$p_3(t)$
1	a_4	Init-Rstrnt.	$=p_2(t)$
	a_2	Retreat	$=p_4(t)$
	a_3	Restraint	$p_3(t)$
2, 3, 4	a_4	Init-Rstrnt.	$=p_4(t)$
	a_2	Restraint	$=p_2(t)$
	a_3	Restraint	$p_3(t)$
5	*	Completed	*

$=p_i(t)$: originally a_i's priority at every time step
$p_i(t)$ uniformly increases at every time step.

Figure 5: Swap task.

Table 1: Constraint/preference of move direction $f_i(v)$.

Ds	P	Baseline: move on the shortest path to the first subgoal.
Dad	P	Avoid DE aisles without the first subgoal.
Dadrc	C	Avoid resolving DE aisle in a restraint mode.
Dadrp	P	Avoid resolving DE asl. in a rstrnt. mod. for higher swp. tsks.
Dado	C	Avoid other DE aisles if the initiator of the top most swap task.
Dl	C	Move on the limited path in a one-push sequence.
Dap	P	Option: Avoid the asl. on the first pusher's path (Matsui, 2024b).

P/C: Preference/Constraint
Priority: Dl > Dado > Dadrp > Dadrc > Dad > Dap > Ds

Table 2: Completion/cancel of swap task.

Ce	Cmpl.	The initiator entered the resolving DE aisle.
Co	Cancel	The task is to be overwritten.
Cp	Cancel	A restraint agent was pushed into the resolving DE aisle.
Ch	Cancel	The initiator in one of other DE aisles became the highest.

Table 3: Acceptable number of agents.

Co, Ce	Dadrc, Dad, Ds	N_b
Cp, Co, Ce	Dadrp, Dadrc, Dad, Ds	N_t
Ch, Cp, Co, Ce	Dl, Dado, Dadrp, Dadrc, Dad, Ds	N_s

N_b	The num. of vertices in the biconnected component.
N_t	(The num. of non-obstacle vertices) − (the num. of vertices in the longest pair of two DE aisles).
N_s	(The num. of non-obstacle vertices) − (the num. of vertices in the longest DE aisle).

appendix section. The life cycle of a swap task consists of several sub-modes that basically representing initiation, retreat, restraint and complete/cancel steps (Fig. 4). The agents related to a swap task are categorized into three types: initiator, target, and swept agents (Fig. 5). The control of a swap task basically consists of the state transition of sub-modes (Fig. 4), a *priority inversion* between the initiator and the target, additional preferences/constraints on the evaluation of agents' moves $f_i(v)$ (Tbl. 1), and several cancel rules to resolve conflicted tasks (Tbl. 2). Several possible combinations of the rules affect the acceptable number of agents (Tbl. 3).

In the following, we describe several details of our approach. We first address the member of cooperative swap tasks (Section 3.2.1). Then the basic flow of the task, including related contexts, is presented (Sections 3.2.2- 3.2.7). Finally, additional rules are introduced to extend applicable cases of the basic method (Sections 3.2.8 and 3.2.9).

3.2.1 Initiator, Target and Swept Agents

When agent a_i, whose first subgoal is in a DE aisle, detects a possible deadlock situation, a swap task is initiated by agent a_i. This situation is detected in the process of PIBT as agent a_i cannot push its next agent located in the DE aisle before arriving at a_i's first subgoal. Although such a situation might be inexact due to some perturbation of a system dependent on PIBT, we accept it as a margin for a bottom-up approach.

An agent can be an initiator primarily in the following two cases. 1) An agent who is at an intersection and entering a DE aisle containing its first subgoal. 2) An agent who is not being pushed and moving in a DE aisle containing its first subgoal. TA) It is possible to further restrict the former case with the condition that the agent is not being pushed.

Target agent a_j is in a push chain and asked to retreat from a DE aisle by initiator agent a_i when target agent a_j' is blocking the first subgoal of a_i. In addition, other agents between initiator agent a_i and target a_j in a push chain are also marked as swept agents that are dug by target agent a_j (Fig. 5).

3.2.2 Initiation of Swap Task

To maintain the consistency of priority values among agents, we allow each agent a_i to initiate a swap task only if agent a_i has a priority value higher than the target, all swept agents, and all their initiator (controller in general cases) agents if any. For a_i itself, it must not have a swap task initiated by an agent with a higher priority value, while a_i can overwrite its own swap task. In addition, a_i cannot initiate a swap task during a specific critical section in a one push sequence discussed in Section 3.2.9. If an agent cannot find the target and swept agents satisfying the conditions above, the agent cannot initiate the swap task until its possible turn.

The initiation operation differs partially for target and swept agents. For a target agent a_j, a new retreat task with a new subgoal r_i is inserted. Basically, the retreat task must be done before a_j's pickup-and-delivery task if it has the task [1]. The new subgoal is the intersection adjacent to the DE aisle. Moreover, the priority values of the initiator and the target are exchanged so that the priority of the target is higher than the initiator. We allocate a retreat task only to a target agent to clarify the role of agents (Figs. 4 and 5).

3.2.3 Context of Swap Task

All member agents, including initiator a_i, target and swept ones, of a swap task that is initiated by agent a_i record 1) the identifiers of the initiator and target agents and 2) a vertex of retreat intersection identical to the subgoal r_i of the target agent's retreat task. One of initiator and target agents with higher priority is distinguished as 3) a controller of a swap task that can push its other members. An initiator also has 4) a set of identifiers for all members of its swap task. With this information, the initiator can notify its members of the completion/cancel of its swap task. Other members can ask their initiator to cancel their task, if necessary (Fig. 4). Each agent can be a member of at most one swap task. Therefore, the initiation and completion/cancel of each swap tasks must be atomic. Note that an initiated task can be overwritten by another initiator with a higher priority or the original initiator itself. In this case, the former task must be canceled to remove its shared information before the initiation of the new task, and that must also be atomic. Although this is slightly complicated, such procedures can be composed without contradiction. We implemented this process with procedures of individual agents that are called by related agents to update their status at appropriate timings in the main process of PIBT (in part of task initiation/termination lines in Fig. 9).

3.2.4 Retreat, Restraint and Completion Phases

After the initiation, target and swept agents immediately change their sub-modes to retreat and swept modes, while the initiator changes to a specific restraint mode (Fig. 4, and $t = 0$ in Fig. 5). These

[1] We slightly optimized this so that a_j's subgoal is processed at first if a_j just locates at its subgoal (Lines 48, and 49 in Fig. 9).

additional operations are performed in the process of PIBT (Fig. 9). The role exchange among initiator and target agents depends on a priority management but we separately describe its details in the next section.

As mentioned above, we allow the agents to initiate their possible swap tasks in time steps arbitrarily, and an existing swap task might be overwritten. Therefore, a swap task might be discarded without completion, although such situations will converge due to the consistent priority values among agents.

When a target agent arrives at the subgoal r_i of its retreat task, it completes the task and the priority values of the target and the initiator are exchanged. We note that the priority value of the retreating target increases at each time step in the manner of PIBT, while the priority is not reset after the retreat task. Then, the corresponding initiator agent recovers its dominance at least over its member agents ($t = 2$ in Fig. 5).

In addition, when each of the target and swept agents arrive at the retreat intersection identical to the subgoal r_i of the target agent's retreat task, each agent changes to the restrained mode. The agents are then inhibited from reentering the DE aisle from which they have retreated. During this period, the corresponding initiator agent can push the target and swept agents except for into the DE aisle of its first subgoal in the manner of PIBT.

When an initiator agent enters the DE aisle with its first subgoal, the initiator notifies its members of the completion of the swap task ($t = 5$, a_4 in Fig. 5). Then, each corresponding target and swept agent exits from the restrained mode and discards the swap task asked by the initiator. If other non-member agents enter a DE aisle during a swap task due to parallel moves of agents, a new swap task will drive the agents away.

3.2.5 Priority Management for Swap

As mentioned above, we allow each agent a_i to initiate a swap task only if agent a_i has a priority value higher than the target, all swept agents, and all their initiator (controller) agents if any. For a_i itself, it must not have a swap task initiated by an agent with a higher priority value, while a_i can overwrite its own swap task. In addition, a_i cannot initiate a swap task during a specific critical section in a one push sequence shown in Section 3.2.9. We permit agents to repeatedly ask to swap in arbitrary time steps if necessary. Before an agent initiates a new swap task, its old swap task is canceled if one exists.

As a result of the initiation of a swap task, the target agent to retreat must have a priority higher than its initiator agent. The operation must also not affect other agents. For this priority management, we employ a priority inversion technique between the initiator and the target of a swap task in this study. Although this is an intuitive idea, we found that the priority inversion raises several complicated issues in handling agents' information [2]. Since the controller agent with the highest priority in a swap task switches, we must always carefully identify the controller agent to evaluate the exact priority value of a swap task. The inversion must also be applied in all cases of canceling swap tasks. The inversion is immediately shared by all members to be *decided* in the same push chain in initiation cases, but can affect *decided/undecided* agents in other cases. This requires an additional mutex in the one push sequence shown in Section 3.2.9. We also note again that the priority value of a target agent increases during its retreat task, while the priority is not reset at the end of the task so that the priority value is returned to its original owner (an initiator). The same applies to another priority.

3.2.6 Subgoal to Retreat

For target agent a_j of a swap task, a retreat task with a subgoal vertex r_i is inserted as a_j's first task. The subgoal vertex r_i to retreat is the intersection adjacent to a DE aisle from which a_j is retreating. We prohibit retreating agents, including swept agents, from allocating their new pickup-and-delivery tasks if they do not have pickup-and-deliver tasks. Therefore, the retreat task is always prior to the pickup-and-delivery tasks of the target and swept agents.

Even though the target agent's first subgoal of its pickup-and-delivery task is outside of its current aisle, we always insert a new retreat task because it relates several other controls of the swap/retreat task. However, if a target agent is newly asked by another agent with a higher priority value to swap, the current swap/retreat task is canceled by asking its initiator before it is overwritten by that of the new swap task.

3.2.7 Limitation of Reentering DE Aisle

After the target and swept agents of a swap task move to the intersection adjacent to the corresponding DE aisle, they must not reenter the DE aisle. The agents change their sub-modes to the restraint mode to inhibit such reentering moves ((Figs. 4 and 5). Although it is possible to simply confine all restrained agents, inside of the biconnected component of a graph, this only well works with the number of agents up to the number of vertices within the biconnected component N_b (Tbl. 3). Instead, it is reasonable to inhibit only reentering a DE aisle related to the current

[2] In an appendix section, we mention another solution depending on monotonically increasing priority values.

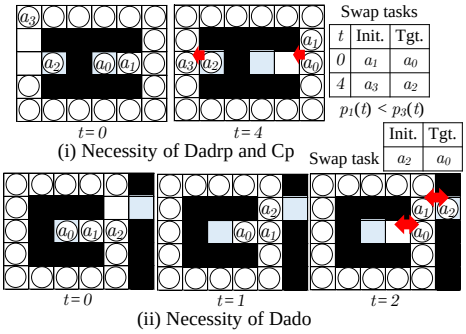


Figure 6: Necessity of extended rules.

swap task. Here, we control the movement direction of agents. Such restriction can be represented by a preference value (Dad) for each movement direction or a hard constraint to exclude such a move (Dadrc) shown in Tbl. 1.

In general, an agent at an intersection should not enter any DE aisle that does not contain its first subgoal regardless of its mode. This is represented by the preference value of the movement direction as a basic extension (Dad), and this preference value must be evaluated prior to the original values of $f_i(v)$ and must not be evaluated prior to other extended constraints and preference values. Most importantly, in the case of restrained agents, the choice of inhibited reentering is eliminated from their movement direction (Dadrc). The restrained mode of a target/swept agent is held until the completion/cancel of the swap task.

3.2.8 More Extended Rules

However, the set of rules above well works with up to N_b agents (Tbl. 3). With the number of agents over N_b, the parallel moves of agents due to PIBT can cause a dead-lock situation. In the case of $t = 4$ shown in Fig. 6 (i), target agent a_2 tries to retreat, and another target a_0 having a lower priority value blocks a_2 by avoiding a_0's own inhibited DE aisle. To resolve this situation, we modify the rule as follows: If restrained agent a_i is pushed at an intersection and the first agent in the push chain has a priority value higher than a_i's controller agent, the limitation of reentering a_i's inhibited DE aisle is considered by a preference value (Dadrp) rather than a hard constraint. As the result, a_i is pushed into its inhibited DE aisle and asks to is initiator to discard the swap task (Cp). Since we allow parallel execution of swap tasks, this rule is necessary to discard a lower priority task in a race condition. Above rules are summarized in Tbls. 1-3.

By adding rules of Dadrp and Cp, the solution process works with up to N_t agents (Tbl. 3). This limitation assures that a pair of an initiator and a target of the topmost swap task can stay in the biconnected

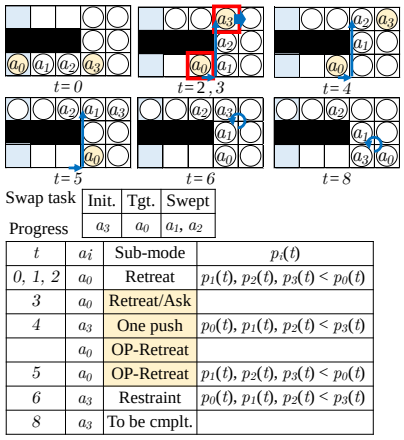


Figure 7: One push sequence.

component. With the number of agents over N_t, the initiator of the topmost swap task can be pushed into a DE aisle. Under this situation, if one of member agents blocks its inhibited DE aisle and there is no room for the initiator to back to the biconnected component, the system sticks (Fig. 6 (ii)). We can add a rule so that the initiator of the topmost swap task always avoids DE aisles except for that of its subgoal (Dado). However, two issues still remain. First, an initiator in a DE aisle might be promoted to that of the topmost swap task. For this case, we force the initiator to cancel its swap task and to retry from the current situation (Ch), and the topmost agent eventually completes its task. The second issue shown below is a self-lock situation in the topmost swap task.

3.2.9 One Push Sequence

We introduce the final extension that is a special mode in a swap task with the highest priority. The rule of Dado prevents the initiator of the topmost swap task from entering other DE aisles during the retreat task phase in its swap task. Instead of that, there can be a type of deadlock situations in the case of the number of agents greater than N_t. This situation is always identical where the initiator of the topmost swap task blocks up to two DE aisles containing at least one unoccupied vertex by staying an intersection connecting to the DE aisles, and other DE aisles except for the resolving one have been already occupied. As the result, the corresponding target agent of the topmost swap task cannot push and sticks in a DE aisle ($T = 3$ in Fig. 7).

To solve this problem, we introduce a special one push sequence where the sticking target agent asks to its initiator to one push to retreat from the blocked intersection (Fig. 4 and the case of $T =$ 3-5 in Fig. 7). Since the initiator is in the biconnected component,

it can always move. After that, the target can push a set of agents into an unoccupied DE aisle. Here, the priority inversion between the initiator and the target agent is applied twice to exchange the control privilege of their swap task. We note that this priority inversion can be performed between an initiator deciding its one push action and a target that does not immediately decide its next action [3]. Therefore, the chances where a non-member agent inverts the one push must be inhibited with a mutex to protect this critical section. We simply force remaining agents to stay in their current location at this time step so that the target cannot be interrupted. In the next time step, the target has the highest priority and can correctly push before others' actions.

In addition, we introduce a special retreat mode following the one push. In the push chain of the PIBT process, each agent basically moves according to its preference value $f_i(v)$, and it might invert the one push action [4]. To avoid this situation, we force the agents pushed by the target to move on the shortest path from the current target's location to the intersection that has been released by the initiator (Dl) ($T = 3$-5 in Fig. 7). Note that this restriction also affects the initiator's one push action by inhibiting its move to an inverted direction. More importantly, the target agent waiting for one push does not move. With this rule, the target successfully completes its retreat mode, and the solution process well works with the number of agents up to N_s (Tbl. 3) that is the theoretical limit.

3.3 Correctness

We briefly sketch the correctness of our method for appropriate settings.

Proposition 1. *A swap task that is initiated by an agent with the highest priority always completes.*

Proof. All the tasks except for one with the highest priority can be canceled when they conflict with another task with a higher priority value. The mutex for the special critical section at the end of one push sequence protects the role exchange between an initiator and a target from an interruption by non-members' pushes. Therefore, the swap task with the highest priority from its initiation is always completed. □

Proposition 2. *All agents have chances to be the one with the highest priority.*

Proof. A swap task with a temporal priority inversion is performed under the priority of its original initiator agent, and the initiator's priority value increases according to the manner of PIBT. Therefore, a priority value of each agent still monotonically increases until the agent reaches its first subgoal of a pickup-and-delivery or stay task. All swap tasks are eventually completed/canceled without resetting priority values, and the highest one is always completed. Therefore, all tasks, including swap tasks, eventually complete. An agent that completes one of other tasks resets its priority. Therefore, all agents have chances to act, and all allocated tasks eventually complete. □

We also note that the presented rules are composed step by step in a lazy manner to find issues to be addressed, and there are other solutions and opportunities to reduce some redundancy. Regarding the completeness, at least there can be dead-lock situations if a system is incorrectly configured with an inappropriate number of agents. The time complexity of the additional part in the PIBT process relates the interaction/maintenance among agents' states and that is almost linear for the number of agents.

For the acceptable number of agents shown in Table 3, the following intuitive proposition exists.

Proposition 3. *In a map where DE aisles are added to a basic map represented by a biconnected graph, any swap tasks can be done in an appropriate sequence if the number of agents is not greater than N_s, where $N_s =$ (the number of non-obstacle vertices) $-$ (the number of vertices in the longest DE aisle).*

Proof. If a group of agents performing a single swap task can empty its resolving DE aisle, and if its initiator agent remains in a biconnected component of the graph of map, the initiator agent can rotate the agents in the biconnected component to move to the DE aisle. For this reason, the number of unoccupied vertices must not be less than that of the longest DE aisle. □

Therefore, the verification with N_s agents is a goal in this study.

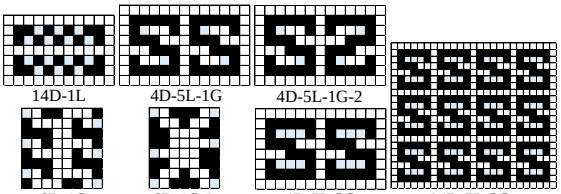
4 EVALUATION

4.1 Settings

We experimentally verified several details of our extended techniques, since we currently concentrate on an extension of PIBT for a specific case of lifelong MAPD problems. While there are several related scalable complete solution methods, including

[3] Here, we do not prefer to reorder the agents in a queue to be processed by the PIBT procedure in a single time step.

[4] This only causes redundant moves of agents (but not preferred), since at least one agent is pushed into an unoccupied aisle.



* Results for 4D-5L-1G-2 and 6D-1G(-2) are eliminated due to space limitation.

Figure 8: Maps for benchmark problems.

PIBT+ (Okumura et al., 2022) and LaCAM (Okumura, 2023), there appears to be additional opportunities to adjust/tune those methods for lifelong MAPD problems. The Token-Passing algorithm for lifelong MAPD problems is only available for well-formed problems with the number of agents at most ((the number of end points)−1) (Ma et al., 2017; Matsui, 2024a) [5]. To mitigate this limitation, we focused on PIBT that is available with large number of agents in narrow maps regardless endpoints and extended it to address specific maps with dead ends. Here we concentrated on the extension of PIBT with additional cooperative swap tasks among agents. TP-based solution methods cannot be applied or can be applied with few agents in most problem settings in our study, and such settings are the beneficial cases of PIBT variants. Dense settings of agents in our experiment appear to be impractical for general optimal search methods. Within the context of PIBT, there are opportunities to employ efficient techniques (Okumura et al., 2019; Yamauchi et al., 2022). Since our current major interest is the experimental verification of the correctness of our extended rules, the performance comparison with those methods will be separately addressed in our future study.

For benchmark problems, we employed the maps shown in Fig. 8 that might not be handled by the original PIBT but can be with our approach. We varied the number of agents up to the theoretical limit. For MAPD problems, NpT tasks were randomly generated with a uniform distribution at every time step with up to 500 tasks in total.

We compared the following solution methods. BASE: Our baseline implementation of the extended PIBT. When an agent is in an intersection connected to a DE aisle, the agent can initiate a swap task if necessary regardless of whether it is pushed or not. TA: When an agent is in an intersection connected to a DE aisle and the agent is not pushed, the agent can initiate a swap task if necessary (Section 3.2.1). RP: An optional strategy where each agent avoids the shortest path preferred by the first agent in its push chain at each intersection (Matsui, 2024b) (Dap in Tbl. 1) [6].

[5] Theoretically safe bounds are (13, 3, 3, 5, 5, 1, 1) for (14D-1L, 4D-5L-1G, *-2, 6D-1G, *-2, 4D-5L-3G, 24D-5L-3G) in Fig. 8, where '1' denotes non-well-formed settings.

Table 4: Makespan and service time.

NpT	#Agt.	10		20		53 (N_t)		54 (N_s)	
(Map)	Alg.	MS	ST	MS	ST	MS	ST	MS	ST
1 (14D-1L)	BASE	1063	254	990	225	**3828**	**1677**	4831	2197
	TA	1078	260	1005	224	4248	1891	5030	2305
	RP	1043	244	**967**	**207**	3857	1692	**4771**	**2177**
	TA+RP	**1039**	**241**	996	216	4251	1907	5052	2313
10	BASE	1078	470	972	419	**3861**	**1869**	**4644**	**2316**
	TA	1048	460	965	417	4136	2017	4994	2484
	RP	**1023**	**447**	960	414	3886	1898	4714	2331
	TA+RP	1029	449	**952**	**409**	4144	2030	4992	2486
	#Agt.	10		20		54 (N_t)		59 (N_s)	
1 (4D-5L-1G)	BASE	2540	948	2473	935	5816	2819	**8578**	**4311**
	TA	2540	948	2484	934	5865	2831	8718	4386
	RP	**2529**	**947**	2454	918	**5741**	**2780**	8625	4324
	TA+RP	**2529**	**947**	**2444**	**915**	5782	2797	8764	4421
10	BASE	2540	1171	2471	1150	5799	2996	8605	**4499**
	TA	2540	1171	2477	1152	**5761**	**2967**	8711	4576
	RP	**2519**	**1161**	**2464**	**1138**	5793	2981	**8590**	4528
	TA+RP	**2519**	**1161**	2470	1143	5774	2992	8789	4621
	#Agt.	10		20		54 (N_t)		59 (N_s)	
1 (4D-5L-3G)	BASE	1963	666	1822	590	**4010**	**1750**	6054	2790
	TA	1963	666	1821	583	4092	1796	6039	2812
	RP	**1850**	**602**	**1786**	**581**	4016	1764	6005	2791
	TA+RP	**1850**	**602**	1787	582	4037	1771	**5963**	**2759**
10	BASE	1936	875	1833	801	3975	1927	5986	2966
	TA	1936	875	1844	804	4048	2001	6012	2962
	RP	**1847**	**828**	1789	775	**3925**	**1922**	5914	2942
	TA+RP	**1847**	**828**	**1775**	**771**	3987	1940	**5890**	**2926**
	#Agt.	100		200		300 (N_t)		305 (N_s)	
1 (24D-5L-3G)	BASE	840	198	1695	693	7904	4515	11287	6610
	TA	842	198	1704	696	**7785**	4437	11271	6627
	RP	**795**	**177**	1431	580	7875	**4393**	11278	6523
	TA+RP	811	184	**1406**	**573**	7899	4426	**11160**	**6447**
10	BASE	731	320	1453	724	7499	4398	11087	6644
	TA	743	324	1453	731	**7485**	4381	10867	6479
	RP	**692**	**307**	1229	**633**	7548	**4335**	**10856**	**6336**
	TA+RP	694	308	**1228**	638	7648	4379	10908	6364

As common metrics, we evaluated the makespan (MS) and service time (ST) that are the number of time steps to complete all tasks and that to complete each task. We also evaluated the number of initiated swap tasks and related metrics. The results over ten executions with random initial locations of agents were averaged for each problem instance. The experiments were performed on a computer with g++ (GCC) 8.5.0 -O3, Linux 4.18, Intel (R) Core (TM) i9-9900 CPU @ 3.10 GHz, and 64 GB memory.

4.2 Results

The solution methods correctly completed for all problem settings. The result revealed that swap tasks well worked with intersections connected to multiple DE aisles (14D-1L and 4D-5L-1G-2), with DE aisles containing multiple pickup-and-delivery locations (4/24D-5L-3G) and with a square (6D-1G(-2)). Table 4 shows the makespan and service time. Here,

[6] We note that the aim of our study is completely different from the previous work that addressed the strategies to reduce redundant moves of agents by considering some knowledge of map structures. We just borrowed one of such strategies to vary the movements of agents for verification.

Table 5: Initiated swap tasks

NpT	#Agt.	10	20	53 (N_t)	54 (N_s)	
(Map)	Alg.	IN	IN	IN	IN	OP
1 (14D-1L)	BASE	117.3	255.5	1191.9	1431.4	3.2
	TA	114.5	211.1	**612.5**	683.9	**1.5**
	RP	122.5	257.5	1239.6	1422.2	3.5
	TA+RP	**113.5**	**208.7**	629.5	**681.7**	1.9
10	BASE	126	262.7	1209.9	1395.6	2.7
	TA	**107.9**	**205.2**	**613.1**	**668.8**	1.9
	RP	117.1	254.9	1242.6	1434.8	2.7
	TA+RP	110.9	210.4	627.3	683.2	**1.8**
	#Agt.	100	200	300 (N_t)	305 (N_s)	
1 (24D-5L-3G)	BASE	408.3	979.7	2669.4	2524.7	9.4
	TA	398.8	916.7	**2075.3**	1829.6	7.1
	RP	**395.9**	938	2686.6	2479.3	9
	TA+RP	401.2	**898.3**	2167.2	**1778.2**	**6.2**
10	BASE	374.6	963.3	2693.5	2518	9.3
	TA	**370**	920.6	**2064.6**	**1807.1**	**6**
	RP	374	944.7	2705.7	2453.8	8.6
	TA+RP	374.5	**880.4**	2211.7	1859.4	6.4

IN: initiation, OP: one push (#agt.$> N_t$)

Table 6: Ratio of completed/re-initiated swap tasks.

NpT	#Agt.	10		20		53 (N_t)		54 (N_s)	
	Alg.	CM	RI	CM	RI	CM	RI	CM	RI
1 (14D-1L)	BASE	0.92	0.70	0.79	0.56	0.40	0.47	0.37	0.48
	TA	**0.971**	**0.16**	0.940	0.311	**0.79**	0.18	**0.783**	**0.17**
	RP	0.92	0.66	0.81	0.57	0.38	0.44	0.37	0.48
	TA+RP	0.968	0.49	**0.944**	**0.19**	0.77	**0.15**	0.780	0.18
10	BASE	0.92	0.72	0.81	0.60	0.39	0.47	0.37	0.48
	TA	0.970	0.40	**0.95**	0.28	**0.79**	**0.15**	**0.79**	**0.16**
	RP	0.93	0.61	0.81	0.58	0.38	0.44	0.36	0.48
	TA+RP	**0.972**	**0.29**	0.92	**0.26**	0.76	0.16	0.77	0.17
	#Agt.	100		200		300 (N_t)		305 (N_s)	
1 (24D-5L-3G)	BASE	0.86	0.45	0.79	0.35	0.63	0.31	0.62	0.33
	TA	0.86	0.42	0.816	**0.34**	**0.72**	**0.27**	0.735	**0.27**
	RP	0.86	0.38	0.80	0.36	0.63	0.32	0.63	0.36
	TA+RP	**0.87**	**0.37**	**0.819**	0.35	0.71	0.28	**0.737**	0.30
10	BASE	0.872	**0.37**	0.79	0.36	0.63	0.30	0.62	0.34
	TA	**0.873**	0.40	0.80	**0.33**	**0.72**	**0.27**	**0.74**	**0.28**
	RP	0.870	0.40	0.79	0.36	0.63	0.32	0.62	0.35
	TA+RP	0.872	0.45	**0.81**	0.35	0.71	0.30	0.72	0.31

CM: completed, RI: rei-init. for the same tgt. and DE aisle

our major interest is not the performance comparison among the solution methods with different minor strategies but the confirmation of their completion. For different settings of problems, the methods were differently affected by the perturbation in their greedy solution process containing swap tasks. From the results of 4D-5L-1G and 4D-5L-3G, the larger number of pickup-and-delivery locations appeared to simply increase the parallelism of the tasks in these settings. Although the methods well worked with the theoretically densest populations of agents, there exits an appropriate number of agents as the common issue.

Table 5 shows the number of initiated swap tasks and that of one push sequences. In these problem settings, TA relatively reduced the number of swap tasks by excluding the agents being pushed from candidates of initiator agents. However, in several different settings we could not find such a significant difference. A few numbers of one push sequences were performed with the number of agents greater than N_t.

Table 6 shows the ratio of completed swap tasks, and the ratio of swap tasks, which are canceled and re-initiated for the same DE aisle, to all canceled tasks. The ratio of completed swap tasks tends to decrease with the density of populations. In these problem settings, TA relatively increased the completion ratio and relatively decreased the overwritten ratio, while those were not so significant in other settings.

With our experimental implementation, the averaged execution time of the solution process was within 9 seconds in the case of 24D-5L-3G, NpT=1, 305 agents and averaged makespan of 11278 time steps. As the first result, we successfully confirmed the completion of solution methods in several fundamental settings and revealed several characteristics regarding the swap tasks, while there are opportunities to improve the solution method.

5 CONCLUSIONS

We improved a solution method based on PIBT for lifelong MAPD problems by integrating a specific swap task. We presented detailed techniques for such an extension, including additional management of priorities, subgoals and states of agents. We also experimentally verified the proposed approach with several problem settings. While we concentrated on the extension of a specific swap task that can be naturally integrated with the original PIBT algorithm as our first study, we also investigated several important detailed properties of the original solution method that are necessary to extend this solution method.

In our future study, we will address more general cases with further extensions and evaluate with related solution methods, including investigation for graphs without cycles and with isthmuses, comparison with scalable/incomplete methods based on top-down approaches, and application of the solution methods with real-time and bottom-up properties to practical domains.

ACKNOWLEDGEMENTS

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.

REFERENCES

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal Variants of the Conflict-Based Search Al-

gorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Annual Symposium on Combinatorial Search*, pages 19–27.

De Wilde, B., Ter Mors, A. W., and Witteveen, C. (2014). Push and rotate: A complete multi-agent pathfinding algorithm. *J. Artif. Int. Res.*, 51(1):443–492.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S., and Koenig, S. (2021). Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 11272–11281.

Luna, R. and Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 1, pages 294–300.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., and Koenig, S. (2019). 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*, pages 7643–7650.

Ma, H., Li, J., Kumar, T. S., and Koenig, S. (2017). Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the Sixteenth Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845.

Matsui, T. (2024a). Integration of Efficient Techniques Based on Endpoints in Solution Method for Lifelong Multiagent Pickup and Delivery Problem. *Systems*, 12(4-112).

Matsui, T. (2024b). Investigation of Heuristics for PIBT Solving Continuous MAPF Problem in Narrow Warehouse. In *Proceedings of the Sixteenth International Conference on Agents and Artificial Intelligence*, volume 1, pages 341–350.

Okumura, K. (2023). LaCAM: search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, pages 11655–11662.

Okumura, K., Machida, M., Défago, X., and Tamura, Y. (2022). Priority Inheritance with Backtracking for Iterative Multi-Agent Path Finding. *Artificial Intelligence*, 310.

Okumura, K., Tamura, Y., and Défago, X. (2019). winPIBT: Expanded Prioritized Algorithm for Iterative Multi-agent Path Finding. *CoRR*, abs/1905.10149.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219:40–66.

Silver, D. (2005). Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122.

Čáp, M., Vokřínek, J., and Kleiner, A. (2015). 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*, pages 324–332.

Yamauchi, T., Miyashita, Y., and Sugawara, T. (2022). 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*, pages 1427–1435.

APPENDIX

Pseudo Code of Extended PIBT Algorithm

The pseudo codes of our extended version of the PIBT algorithm are shown in Fig. 9. Since the original version of the pseudo codes are described in a compact form, we first expanded an if-block (lines 16-23 in Fig. 2) with two internal blocks (lines 24-42 in Fig. 9). Additional parameters a_f, a_s, and p_d, and return value a_t in function PIBT propagate additional information in its recursion process (lines 6, 13, 28, 34, 40, 44, and 56).

a_f represents the first pusher in a push chain and that is implicitly referred in several extended rules for $f_i(v)$ (lines 6, 13, 14, 19-20, 22, and 28).

To initiate each swap task, we utilized the recursion process of PIBT in a slightly technical manner. In a top-down path of the recursion, the information of a candidate a_s for an initiator agent and an associating priority value p_d is propagated (lines 6, 13, 15, 16, and 26-29). When agent a_i having a candidate initiator a_s cannot move, a_i enables a swap task initiated by a_s, by setting target $a_t = a_i$ (lines 47-50). Then a_s partially initiates its swap task for a_i (line 51). Namely, the initiation process by a_s is performed in a return path of recursion. Similarly, the relating swept agents are also initiated in the same return path (lines 52, and 53). Here, we decomposed the communication among the member agents of each swap task, including the cancellation of existing tasks by considering correct timings. Finally, the initiation is completed in the level of a_s (line 29).

The completion/cancellation of swap tasks is checked in several appropriate timings (lines 17, 31, 37, 43, and the implicit cancel communication among agents). Subgoals and sub-modes of agents are updated in the timing of their moves if necessary (lines 10, and 11). In addition, the special rules for the one-push sequences are also embedded (lines 7-8, 32-33, and 38-39), including mutex of the sequence.

```
UNDECIDED← A(t) // agents list
OCCUPIED← ∅ // vertices list
update priorities p_i(t) for all agents a_i
while UNDECIDED≠ ∅ do
  a ← the agent with the highest priority in UNDECIDED
  PIBT(a,⊥,⊥,⊥,⊥) // ⊥ denotes empty
  if inOPretreat(target(a)) then make the rest of agents
    in UNDECIDED stay. end if // mutex to protect role xchg.
end while
Manage sub-modes that can be done in an update phase of
  agents' locations and subgoals.

function PIBT(a_i,a_j,a_f,a_s,p_d)
  if a_f = ⊥ then a'_f ← a_i else a'_f ← a_f end if, a_t ← ⊥
  if a_s ≠ ⊥ then p'_d ← max(p_d, p_i(t), p_controller(a_i)(t))
  else p'_d ← p_d end if
  Apply Ch.
  UNDECIDED←UNDECIDED\{a_i}
  C_i ← ({v|((v_i(t),v) ∈ E} ∪ {v_i(t)) ∧ f_i(v) ≠ ⊥})
    \({v_j(t)}∪OCCUPIED) // with new constraints for f_i(v)
  while C_i ≠ ∅ do
    v*_i ← arg max_{v∈C_i} f_i(v) // with new preferences for f_i(v)
    OCCUPIED←OCCUPIED∪{v*_i}
    if a_k s.t. v*_i = v_k(t) exists then
      if a_k ∈UNDECIDED then
        if a_i can be an initiator then a'_s ← a_i, p'_d ← p_i(t)
        else a'_s ← a_s end if
        (r,a_t) ←PIBT(a_k,a_i,a'_f,a'_s,p'_d)
        if a'_s = a_i then a_i ← ⊥ end if // complete initiation
        if r is valid then
          v_i(t+1) ← v*_i, apply Ce and Cp.
          if inOnePush(a_i) then ask target(a_i) to
            one-push retreat. end if
          return (valid,⊥) // move with push
        else C_i ← C_i\OCCUPIED end if
      else
        v_i(t+1) ← v*_i, apply Ce and Cp if v_i(t+1) ≠ v_i(t).
        if v_i(t+1) = v_i(t)∧inTopMostSwapTask(a_i)∧
        target(a_i)=a_i then ask initiator(a_i) to one push. end if
        return (valid,⊥) // move/stay without push
      end if
    else
      v_i(t+1) ← v*_i, apply Ce and Cp.
      return (valid,⊥) // move without push
    end if
  end while
  if a_s ≠ ⊥∧a_t = ⊥∧p_s(t) = p'_d ∧ v_i(t) = sg(a_s)∧
    ¬(hasMAPDtask(a_i)∧v_i(t) = sg(a_i))∧
    (v_i(t) = s_i ∨ v_i(t) = g_i)) // complete MAPD subgoal first
  then a_t ← a_i,
    a_s partially initialize swap task for a_s and target a_i.
  else if a_t ≠ ⊥ then
    a_s partially initialize swap task for a_s and swept agent a_i.
  end if
  v_i(t+1) ← v_i(t)
  return (invalid,a_t) // stay by failing to move
end function
```

a_f: first pusher for new $f_i(v)$, a_s: initiator candidate a_t: target to initiate swap task in return path of recursion p_d: dominant priority, $sg(a_i)$: first subgoal of a_i $v_i(t)$: location of agent a_i at time step t	
Task initiation	6, 13-16, 28-29, 34, 40, 44, 47-54, 56
Task termination	17, 31, 37, 43, and implicit cancel
$f_i(v)$	6, 13-14, 19-20, 22, 28
subgoal, sub-mode	10-11
one push	7-8, 32-33, 38-39

Figure 9: Extension to PIBT (time step t).

Monotonically Increasing Priority Values

To maintain the consistency of priority values among agents, we only allow each agent a_i to initiate a swap task only if agent a_i has a priority value higher than target and all swept agents. As the result of the initiation of a swap task, the target agent to retreat must have a priority higher than its initiator agent. After the target agent retreats, it asks its initiator to be its controller again. The operation must also not affect other agents. Therefore, the target agent should be inserted between the initiator agent and an agent who has the minimum priority higher than the initiator agent. In addition, we permit agents to multiply ask to retreat in arbitrary time steps if necessary. When an agent initiates a new swap task, its old swap task is overwritten if one exists.

We can employ the following hierarchical priority value p_i of agent a_i. Here time step t is omitted.

$$p_i = pe_i + cp \cdot pa_i + ca \cdot pn_i/pd_i, \tag{1}$$

where $pe_i \gg cp \cdot pa_i \gg ca \cdot pn_i/pd_i$. pe_i is the elapsed time from the update of the first subgoal of the agent. pa_i is the additional value to break ties of agents. pn_i and pd_i are integer values. pn_i/pd_i is employed to adjust priority values among two agents. These values are initialized as $pe_i = 0$, $pn_i = 0$ and $pd_i = 1$. pa_i is initially based on agent a_is' identifier. pe_i is reset to zero after agent a_i arrived its first subgoal location. Otherwise, pe_i is incremented in each time step. When agent a_i raises agent a_j's priority than a_i, the following update is performed after $p'_j \leftarrow p'_i \leftarrow p_i$.

$$pn'_i \leftarrow 3pn_i + 1 \tag{2}$$
$$pd'_i \leftarrow 3pd_i \tag{3}$$
$$pn'_j \leftarrow 3pn_i + 2 \tag{4}$$
$$pd'_j \leftarrow 3pd_i \tag{5}$$

It also increases agent a_i's priority to avoid to generate the same priority for different agents.

In actual implementation, pn_i and pd_i frequently exceed the precision of variables. To avoid such situations, we iteratively reorder the agents by $pa_i + ca \cdot pd_i/pn_i$ and update pa_i by the ordering. Then pn_i and pd_i are reset as $pn_i = 0$ and $pd_i = 1$. The required frequency of this update depends on the precision of the variables and the number of initiated swap tasks. The reset of the priority values must be synchronized among an entire system in a decentralized implementation.

Actually, we developed the proposed algorithm under this type of priority values and finally replaced it by the priority inversion. Employing monotonically increasing values is a standard approach to control systems with multiple components. The frequency of the reset under uint64_t variables was sufficiently acceptable in our preliminary experiment.