A TSP-Based Online Algorithm for Multi-Task Multi-Agent Pickup and Delivery

Fumiya Kudo and Kai Cai, *Senior Member, IEEE*

***Abstract*—The Multi-Agent Path Finding (MAPF) and its extension, Multi-Agent Pickup and Delivery (MAPD), have received much attention in academia. In industry, on the other hand, automatic control of teams of robots and AGVs on factory floors and logistic warehouses for pickup and delivery operations have also been studied intensively. Currently, MAPD problem formulation does not fully capture important aspects of many real-world industrial applications, e.g., MAPD allocates only one task at a time for each agent, payload capacity for each agent is ignored, and pickup & dropoff operations are assumed to be done immediately. In this letter, we extend MAPD problem to a multi-task setting where each agent is allocated multiple tasks considering payload capacity as well as pickup & dropoff cost. We propose an online multi-task MAPD algorithm which is a combination of MAPF and Traveling Salesman Problem (TSP) algorithm. Comparisons between the proposed and conventional MAPD show that the proposed MAPD is able to achieve $18\%-38\%$ shorter makespan paths in wide range of agent numbers. We also examine the behavior of the proposed online multi-task MAPD by changing payload capacity distribution and pickup & dropoff cost. Simulation results indicate that increase of pickup cost can largely increase the makespan when agent number is small; on the other hand, increase of dropoff cost tend to increase the makespan when agent number is large. Our empirical study also demonstrates that the proposed online multi-task MAPD is applicable to large scale environment (e.g., agent number $= 300$) in an online manner.**

***Index Terms*—Discrete event system, multi-agent path finding, multi-agent pickup and delivery, traveling salesman problem, warehouse automation.**

I. Introduction

THERE is large demand for logistic automation by multiple moving robots (agents) in warehouses and factories due to increasing labor shortage [1]. To meet such demand, it is essential to develop sophisticated algorithms for planning and controlling multi-agent systems to serve tasks dynamically appearing in complex environment. In such multi-agent systems, collision-free paths must be provided to ensure that agents reach their destinations without collisions, while minimizing the sum of their travel times. This problem is known as Multi-Agent Path-Finding (MAPF), which has recently received a lot of attention [2]. A prominent application is Kiva systems in Amazon's warehouse and distribution centers [1]. MAPF is an optimization problem that aims to find collision-free paths on a graph for multiple agents with the objective of minimizing a task-completion time called *makespan*, or an average number of time steps to complete all tasks called *service time*.

Manuscript received 28 February 2023; accepted 10 July 2023. Date of publication 3 August 2023; date of current version 9 August 2023. This letter was recommended for publication by Associate Editor K. Yu and Editor J. Yi upon evaluation of the reviewers' comments. This work was supported in part by JSPS KAKENHI under Grants 21H04875 and 22KK0155, and in part by the 2023 Osaka Metropolitan University Strategic Research Promotion Project. *(Corresponding author: Kai Cai.)*

The authors are with the Department of Core Informatics, Osaka Metropolitan University, Osaka 558-8585, Japan (e-mail: fumiya.kudo.wy@hitachi.com; cai@omu.ac.jp).

Digital Object Identifier 10.1109/LRA.2023.3301300

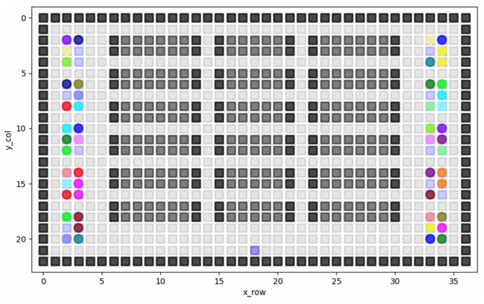

Fig. 1. Illustration of MAPF and MAPD application in logistic warehouse. Agents are depicted by colored circle at their initial locations. The walls and the shelves are depicted by black and gray squares, respectively. Products to be picked up are placed at shelves. The dropoff location is depicted by the purple square at the center bottom of the field.

There have been several extensions of MAPF: Multi-Agent Pickup and Delivery (MAPD), Task-Allocation and Path Finding (TAPF), and Lifelong MAPF. These extensions are different combinations of delivery task allocation with the MAPF problem [3], [4]. An illustrative application of MAPF and MAPD in logistic warehouse is shown in Fig. 1. This is a snapshot of the initial state, where all agents (colored circles) are at their initial locations in the field. When picking tasks arrive which request agents to pickup the target products, agents will be allocated exactly one task at a time. Then, task allocated agents will start from their initial locations and move to the shelves (gray squares) where target products are stored. After picking up target products, agents will be requested to bring the products to the goal location, i.e., dropoff location (purple square at the center bottom of the field). After finishing the assigned tasks, agents will return to their initial locations and wait for the next request.

Various approaches to solving MAPD problem have been proposed, such as coupled / decoupled, centralized / decentralized MAPF approaches in online / offline setting [3], [5], [6], [7], [8], [9]. However, these approaches do not capture important characteristics of many real-world applications such as: MAPD allocates only one task at a time for each agent, payload capacity for each agent is ignored, and pickup & dropoff operations are assumed to be done immediately. The MAPD problem with multiple tasks assignment (multi-task MAPD) appears for example, in the printed circuit board manufacturing plant. In this case, agents are requested to pickup hundreds of required chips from the shelves. Each printed board requires different combination of chips, so each agent is assigned to pick all the different chips required for one target printed board. Then, printed boards and corresponding chips are sent together to the next assembly process. For such an application, we need to take into account the following aspects at the same time: 1. shortest paths for each agent's trip to the destination, 2. efficient task execution order, and 3. avoid collision among all agents.

In this letter, we formulate a multi-task MAPD and propose an efficient online solution algorithm. The contributions of this letter are as follows:

- We extend the conventional single-task MAPD problem to a multi-task setting where i) each agent is allocated with multiple tasks in an online manner; ii) payload capacity of each agent is considered as a constraint; and iii) pickup & dropoff times are considered as cost. To our best knowledge, this extended MAPD is new and no existing algorithm can be directly applied to solve this extended problem.
- We propose an online algorithm to solve the formulated multi-task MAPD problem which is the combination of MAPF and travelling salesman problem (TSP) algorithms. The challenge is that we cannot solve the path planning problem and TSP separately since path planning and TSP are dependent on each other. In order to find optimal shortest paths, we need to recalculate collision-free paths every time when searching TSP paths, which is time consuming for online use in the real-world. Our proposed algorithm effectively deals with this challenge and achieves sufficient efficiency for online use.
- We conduct extensive experiment to compare our proposed MAPD solution with the existing algorithms. Special focus has been given to three parameters: task group arrival frequency, maximum task group size, and agent number. Empirial evidence shows that: our proposed online algorithm achieves $18\% - 38\%$ shorter makespan paths compared to the conventional MAPD algorithms (i.e., Token Passing) in wide range of task group arrival frequency, maximum task group size, and agent numbers. We also show that the proposed online multi-task MAPD can be applied to large scale environment (e.g., agent number $= 300$) in an online manner.
- Moreover we change payload capacity distribution and pickup & dropoff cost to evaluate makespan and runtime. We find by simulation that: increase of pickup cost can largely increase the makespan when agent number is small; on the other hand, increase of dropoff cost tend to increase the makespan when agent number is large.

The remaining parts of this letter are organized as follows: We first introduce related approaches and algorithms for MAPF and MAPD (Section II). Then, we present our problem setting: multi-task MAPD (Section III). Next, we describe our proposed online algorithm for solving multi-task MAPD in detail (Section IV). After that, we conduct extensive experiment in which we apply our proposed online multi-task MAPD with various parameters (Section V). Finally, we conclude the letter (Section VI).

II. Related Work

We introduce and compare existing approaches and algirithms for MAPF, MAPD and other related problems. MAPF problems and approaches can be categorized based on different aspects, e.g., online / offline task arrival setting, centralized / decentralized and coupled / decoupled approaches. Online task arrival means that the problem is a lifelong setting where tasks can enter the system at any time. Therefore, assigning agents to tasks and planning paths accordingly cannot be done in advance but rather need to be done during execution in real-time. In contrast, offline means that we know all the information about arriving tasks *a priori*. Coupled approach is a complete MAPF algorithm which can find optimal solutions [10], [11], [12]. However, finding an optimal solution is NP-hard, and consequently these optimal algorithms do not scale in the number of agents. In contrast, decoupled approach is an incomplete MAPF algorithm which finds sub-optimal solutions fast [13]. Centralized approach assumes that all agents know all information of other agents and the environment. On the other hand in decentralized approach, each agent searches its own path based on locally observable information from (typically) neighboring agents [8]. We introduce three representative MAPF algorithms.

Conflict-based Search (CBS) [10] is a coupled and centralized MAPF algorithm. CBS is based on a two-level MAPF algorithm. At the high level, CBS calculates a shortest path for each agent ignoring collisions with other agents. Then, CBS chooses and resolves a collision by generating two child nodes, each with an additional constraint that prohibits one of the agents involved in the collision from being at the collision location and timestep. Then, the low level replans the paths of the agents with the new constraints. CBS is one of state-of-the-art MAPF algorithms that has many variants [11], [12].

Cooperative A* (CA*) [13] is a decoupled and centralized MAPF algorithm. CA* uses a special type of A* called space-time A*. Space-time A* uses a reservation table which is a map consisting of two elements: timestep t, and location coordinates (x, y). CA* is based on a simple prioritized-planning scheme: Each agent computes, in the given priority order, greedily by searching for a goal, avoiding reserved states in the reservation table. Then, mark each agent's route in the reservation table. There are variants of cooperative A*: Hierarchical Cooperative A* (HCA*), Windowed Hierarchical Cooperative A* (WHCA*) [13]. CA* is widely used in practice due to its small runtime. Note that CA* is sub-optimal since it runs by predefined priority order.

Priority inheritance with backtracking (PIBT) [8] is a decoupled and decentralized MAPF algorithm. PIBT is one of the prioritized planning and is based on WHCA* [13] with a window size of one. PIBT resolves collisions by assigning one of the agents involved in the collisions a higher priority than the other agents. PIBT is scalable and can be applied to online setting since it is fully decentralized. However, PIBT may have chance to fail resolving collision since it is a decentralized approach.

In addition to the above three representative algorithms, there are many other variants of MAPF problems and approaches. ML-MAPF [14] solves MAPF by a machine learning approach. x* [15] is a window-based real-time MAPF algorithm which assumes a sparse environment. GA-based MAPF [16] is a multi-objective version of MAPF using Genetic Algorithm (GA). Multi-Goal MAPF [17] studies the problem where agents travel multiple destinations.

MAPD (also TAPF, lifelong-MAPF) is an extension of MAPF, which requires both the assignment of agents to tasks in a lifelong setting and the planning of collision-free paths. COBRA [7] is one of the initial online and complete algotithms for MAPD. Token Passing (TP) and Token Passing with Time Swaps (TPTS) [6] are decoupled and centralized approaches based on CA* [13]. TP and TPTS are incomplete, however widely applied because they are fast enough for online implementation in the real-world. Multi-Label A* (MLA) [18] is an improved algorithm of TPTS. TCBS [5] is a complete MAPD algorithm based on CBS which can find optimal solution; however, examined environment is small (map size of 8×8, agent size < 10). M-TA-Prioritized-MAPD [3] is an offline MAPD which uses TSP for optimizing an order of task allocation. M-TA-Prioritized-MAPD achieves better throughput compared to other MAPD approaches (e.g., TPTS); however, the problem setting is offline and the computation time for calculating TSP is excluded from the evaluation. Rolling-Horison Collision Resolution (RHCR) [9] is one of state-of-the-art MAPD approach, which is based on WHCA* [13] that sets time window for searching a collision-free path.

Other related problems are Dynamic Vehicle Routing Problem (DVRP) [19] and TSP. The purpose of VRP is to search for the shortest routes for multiple agents (e.g., trucks); however VRP does not consider collision among agents. TSP is a classical NP-hard optimization problem which is well studied and has many solvers. There are two types of TSP solvers which are complete and incomplete solvers. One of the representative complete solver is the integer linear programming model with Miller-Tucker-Zemlin (MTZ) formulation [20]. On the other hand, many incomplete solvers have been proposed, e.g., Christofides algorithm [21], 2-opt method [22], and nearest neighbor method.

III. Problem Setting

In this section, we formalize a multi-task MAPD problem that is to find collision-free paths for multiple agents to accomplish tasks arriving in an online manner.

Consider an undirected connected grid graph $G = (V, E)$, whose vertices V correspond to locations $v = (x, y) \in V$ and whose edges E correspond to connections between four locations, namely north, east, west, and south that the agents can move along. Also consider a set of m agents $A = \{a_1, a_2, \ldots, a_m\}$ and let $v_i(t) \in V$ denote the location of agent a_i in discrete timestep t. Agent a_i starts in its initial location $v_i(0)$. In each timestep t, an agent either $waits$ in its current location $v_i(t)$ or $moves$ to an adjacent location. Both $move$ and $wait$ actions have unit duration.

A task s requests an agent to pick up a target product located at a shelf $v_s \in V$. In our multi-task setting, there are generally multiple tasks which request to be picked up together and are gathered in a task group $TG_l = \{s_1^l, \ldots, s_n^l\}$. Agents are allocated task groups instead of a single task. Note that a task group has at least one task. In this problem, we need to consider a trip order for the agents in order to maximize the throughput while avoid collision among them. Each agent i has payload capacity constraint $1 \leq c_i$ which sets upper bound for the maximum number of products agent i can carry. This means that the target task group can be allocated to agent i only when c_i is larger than or equal to the number of tasks in the task group. When a task group TG_l is assigned to agent i, this agent moves from its initial location $v_i(0)$ to the requested shelf locations $v_1^l, \ldots, v_n^l$ ($n \leq c_i$) to pick up the target products, and then moves to the unique goal location $v_g \in V$ to dropoff those products. The agent moves back to its initial location when the agent finishes the assigned task group. Note that conventional MAPD is a special case of multi-task MAPD which considers that every task group has only one task. A task group can be allocated only to vacant (idle) agents, i.e., agents staying at their initial location with no task group assigned. Pickup & dropoff action may take duration as $move$ and $wait$. Moreover, we consider that this multi-task MAPD problem is online, i.e., we do not know task group information *a priori*, and new task groups are added to a task group set *T* randomly at each timestep. We assume that there is a task group assigner outside of our MAPD system and a vacant agent is assigned a task group from *T* at each timestep when the agent's payload capacity is greater or equal to the number of tasks in the task group.

In planning paths for the agents, collisions between agents must be avoided. A collision occurs when two agents a_i and a_j occupy the same location at the same timestep (called vertex conflict [23]), that is, $(\exists t) v_i(t) = v_j(t)$; or traverse the same edge in opposite directions at the same timestep (called a swapping conflict [23]), that is, $(\exists t) v_i(t) = v_j(t+1)$ and $v_j(t) = v_i(t+1)$. A path is a sequence of locations with associated timesteps, that is, a mapping from an interval of timesteps to locations. The objective of multi-task MAPD is to compute collision-free paths for the agents to accomplish task groups assigned online and maximize the throughput, i.e., minimize the *makespan* (difference between first release time and latest completion time).

IV. Proposed TSP-Based Online Multi-Task MAPD

In this section, we present our proposed TSP-based online multi-task MAPD algorithm. The basic idea of our proposed algorithm is the combination of space-time A*-based MAPF and TSP solver. Our algorithm is capable of dealing with online MAPD (i.e., assigning agents to tasks and path planning need to be done during executing in real-time) and guarantees to find collision-free paths for all agents. To this end, we choose centralized and decoupled MAPF approach.

In multi-task MAPD, task group which consists of multiple tasks will be assigned so that we need to consider the execution order of assigned tasks. We use TSP solvers to tackle this problem. In order to guarantee solving TSP in real-time, we use complete TSP solver (e.g., integer programming problem with Miller-Tucker-Zemlin formulation [20]) for small task group size whereas use incomplete TSP solver (e.g., 2-opt method [22]) for large group size. On the other hand, for the path planning part, we use space-time A*. We suitably combine space-time A* and TSP solvers to construct our online multi-task MAPD algorithm.

Pseudo-code for the proposed online multi-task MAPD algorithm is shown in Algorithm 1. Here we list the key notation:

- G denotes an undirected connected grid graph.
- $A = \{a_1, a_2, \ldots, a_m\}$ denotes all m agents. Agent a_i has its initial location $= (x_i, y_i)$ and payload capacity $= c_i$.
- TASKLIST contains arrived task groups. A task group $TG_l = \{s^l_1, \ldots, s^l_n\}$ is consisted of n tasks.
- VACANTAGENTS is a list of agents with no task allocated.
- RESERVE holds reserved paths of previous task allocated agents. RESERVE is a hush map consisting of two elements: timestep t, and location coordinate (x, y)

The algorithm executes as follows.

1) lines 1–4: Initialize A, TASKLIST, VACANTAGENTS, and RESERVE.
2) lines 5–6: TASKGROUPGENERATOR function adds new task group TG_l, if any, to the TASKLIST at each timestep.
3) lines 7–11: If the TASKLIST and the VACANTAGENTS are not empty, task groups will be allocated to agents. CAPACITY function searches agents satisfying capacity constraint from VACANTAGENTS. RANDOM function randomly chooses agent a_i from A'. In case there is no agent in VACANTAGENTS that satisfies the capacity constraint(s) of the task group(s) in the TASKLIST, task allocation will not occur and the unassigned task group(s) are kept in the TASKLIST until the next timestep. On the other hand, if there are fewer vacant agents in VACANTAGENTS than the number of task groups in the TASKLIST, then a subset of task groups will be allocated to vacant agents that satisfy the respective capacity constraints and the rest unassigned task groups are kept in the TASKLIST until the next timestep. In ALLOCATION function, a task group TG_l will be allocated to agent a_i if the task group TG_l satisfies agent a_i's payload capacity constraint.
4) lines 12–13: CALCDISTANCEMATRIX function [13] calculates the distance matrix between tasks in the same task group TG_l using A* [24]. SOLVETSP function [20], [22] finds the shortest path p by solving TSP for the task group TG_l including the initial and the goal location of the allocated agent a_i.
5) lines 14–16: RESOLVECOLLISION function [13] converts the shortest path p to a collision free path p' as follows:
   a) If p violates RESERVE, space-time A* adds p one timestep to stay at the same location.
   b) If p causes deadlock (including swapping conflict) which means that no agent can move, space-time A* adds one timestep to stay at the initial location of the agent and restart space-time A*.
   c) UPDATE function registers the accepted path p' to the RESERVE.
   d) VACANTAGENTS.delete function deletes a_i from VACANTAGENTS.
6) line 19: DELETE function deletes allocated task groups from TASKLIST.
7) line 20: MOVEAGENTS function moves task group allocated agents one step forward.

Algorithm 1: Proposed Online Multi-Task MAPD.

```
Input: graph $G$, set of agents $A = \{a_1, \ldots, a_m\}$
Output: RESERVE
 1: initialize $a_i \leftarrow$ (initial location= $(x_i, y_i)$, capacity= $c_i$)
 2: TASK LIST $\leftarrow \emptyset$
 3: VACANT AGENTS $\leftarrow \{a_1, a_2, \ldots, a_m\}$
 4: RESERVE $\leftarrow \emptyset$
 5: for timestep= 1 to MAXTIME do
 6:   TASKLIST $\leftarrow$ TASKGROUPGENERATOR()
 7:   for $l = 0$ to size(TASKLIST) do
 8:     $A' \leftarrow$ CAPACITY(VACANTAGENTS, $TG_l$)
 9:     if size($A'$) $> 0$ then
10:       $a_i \leftarrow$ RANDOM($A'$)
11:       ALLOCATION($TG_l$, $a_i$)
12:       CALCDISTANCEMATRIX($a_i$, $G$)
13:       $p \leftarrow$ SOLVETSP($a_i$)
14:       $p' \leftarrow$ RESOLVECOLLISION($p$, $G$, RESERVE)
15:       UPDATE(RESERVE, $p'$, $a_i$)
16:       VACANTAGENTS.delete($a_i$)
17:     end if
18:   end for
19:   DELETE(TASKLIST)
20:   MOVEAGENTS($A$)
21: end for
```

Remark 1: We summarize the key designs that enable Algorithm 1 to handle multi-task MAPD.

- Since a task group generally contains multiple tasks, it must be verified if an agent has enough capacity to handle the number of tasks. This is done in line 8.
- The order of serving multiple tasks is crucial in minimizing makespan. To optimize the order, a matrix containing pairwise distances of the tasks needs to be computed — this is done in line 12.
- With the distance matrix computed in line 12, the order for a selected agent to serve the multiple tasks is optimized by a TSP solver in line 13.

Remark 2: The complexity of Algorithm 1 is $O(N^3 m C^3)$, where N is the total number of tasks (i.e., sum of tasks in all task groups), m is the agent number, C is the number of cells in the map (i.e., number of nodes in the grid graph $G = (V, E)$). To see this, observe that the main part contributing to the complexity is the for-loop of lines 7-18. In particular, the following lines of computation have high complexity.

- Line 12 (CALCDISTANCEMATRIX): $O(N^2 C^2)$
  The distance matrix has at most N^2 entries, and each entry is obtained by an A* computation of $O(C^2)$ [25].
- Line 13 (SOLVETSP): $O(N^2)$
  The TSP has at most N tasks to route, for which the 2-opt algorithm we adopt has the complexity $O(N^2)$ [22].

TABLE I
COMPARISON BETWEEN PROPOSED AND CONVENTIONAL MAPD (TASK GROUP ARRIVAL FREQUENCY=0.2)

#	Task Group Arrival Frequency	Maximum Task Group Size	Agent Number	Algorithm 1 (Manhattan)		Algorithm 1 (Euclidean)		TP	
				Makespan	Runtime/step	Makespan	Runtime/step	Makespan	Runtime/step
1	0.2	1	10	3610	5.2ms (18.8s)	3578	5.2ms (18.8s)	3638	3.0ms (11.0s)
2	0.2	1	20	2575	6.8ms (17.5s)	2575	6.8ms (17.4s)	2573	3.7ms (9.4s)
3	0.2	1	30	2573	7.3ms (18.9s)	2579	7.4ms (19.1s)	2576	4.2ms (10.8s)
4	0.2	1	40	2575	8.0ms (20.5s)	2580	8.1ms (20.9s)	2574	5.0ms (12.8s)
5	0.2	1	50	2576	8.6ms (22.3s)	2577	8.7ms (22.4s)	2577	5.4ms (14.0s)
6	0.2	10	10	1114	8.4ms (9.4s)	1101	9.0ms (9.9s)	1356	3.4ms (4.5s)
7	0.2	10	20	686	12.6ms (8.6s)	687	13.7ms (9.4s)	823	4.8ms (4.0s)
8	0.2	10	30	613	14.9ms (9.1s)	610	16.9ms (10.3s)	705	6.7ms (4.7s)
9	0.2	10	40	616	15.5ms (9.5s)	611	16.8ms (10.3s)	693	7.1ms (5.0s)
10	0.2	10	50	605	16.1ms (9.8s)	621	16.8ms (10.4s)	689	8.2ms (5.6s)
11	0.2	20	10	814	12.5ms (10.2s)	827	11.6ms (9.6s)	1278	3.6ms (4.6s)
12	0.2	20	20	559	18.3ms (10.2s)	558	18.7ms (10.4s)	850	5.0ms (4.2s)
13	0.2	20	30	500	19.9ms (10.0s)	489	25.6ms (12.5s)	742	7.2ms (5.4s)
14	0.2	20	40	496	24.1ms (12.0s)	492	24.0ms (11.8s)	705	9.8ms (6.9s)
15	0.2	20	50	495	23.0ms (11.4s)	486	24.0ms (11.7s)	712	10.0ms (7.1s)

- Line 14 (RESOLVECOLLISION): $O(N^2mC^3)$
  Each agent has at most N tasks to serve and each task takes at most C steps; hence the total number of steps for all agents to finish all tasks is upper bounded by NmC. Each of such steps may cause a collision, so there are at most NmC times of $wait$. For each $wait$, A* computation of $O(C^2)$ is done at most N times for the target agent to obtain a new (collision-free) path. Thus the total complexity of this line is $O(N^2mC^3)$.

Thus the most time-consuming computation is line 14. Since the for-loop of lines 7–18 can happen no more than N times, the total complexity of Algorithm 1 is $O(N^3mC^3)$.

V. EXPERIMENTS

In this section, we evaluate our proposed online multi-task MAPD algorithm with respect to a few key parameters: task group arrival frequency, maximum task group size, agent number, pickup & dropoff costs, and payload capacity distribution.

A. Multi-Task vs. Single-Task MAPD Algorithms

First, we compare our proposed online multi-task MAPD algorithm with the TP (Token Passing [6]) algorithm, the representative conventional MAPD algorithm. TP can only address single-task MAPD problem, so if it is used for multi-task MAPD, TP does not have function to optimize the order of picking up multiple tasks. However, to compare with our proposed algorithm, we apply TP to a multi-task MAPD problem by executing each task in the allocated task group in a random order. On the other hand, for our proposed algorithm, we consider using two types of distance in calculating distance matrix for TSP:

- (Algorithm 1 - Manhattan): use A* to calculate distance between two locations based on Manhattan distance.
- (Algorithm 1 - Euclidean): use Euclidean distance between two locations.

The size of the map is 36×22, the unique goal (dropoff) location is $(18, 21)$. For this experiment, we ignore pickup & dropoff costs (set to 0) and agents' payload capacity constraint (i.e., payload capacity $= \infty$) to all agents. We generate a sequence of 500 tasks by randomly choosing their pickup locations from the shelves. These tasks are randomly grouped into task groups. The maximum group size is set to three patterns: 1 (same as conventional MAPD problem setting), 10, and 20. We set task group arrival frequency to three patterns: 0.2, 1, and 10.[1] We also vary agent number from 10 to 50. All experimental settings are performed in 10 instances with initial positions of agents set randomly. We evaluate makespan [timestep] and runtime/step [ms] (total runtime [s]). The results are shown in Tables I, II, and III. In the sequel we write "Algorithm 1" to mean both Algorithm 1 (Manhattan) and Algorithm 1 (Euclidean) as in Tables I, II, and III.

As we see from Tables I, II, and III, regardless of task group arrival frequency or maximum task group size, the makespan gets shorter and runtime/step gets larger as agent number increases in all three methods. However in Table I where task group arrival frequency is low, improvement of the makespan saturates at agent number $= 30$, (e.g., makespan in #6–10 of Algorithm 1) because low task group arrival frequency becomes the bottle neck of the makespan, i.e., many vacant agents may have to wait for the task group arrival. Improvement of the makespan gets larger as task group arrival frequency gets higher, e.g., makespan in #15, #30, and #45 of Algorithm 1 in Tables I, II, and III. Let us take a closer look in Table III where task group arrival frequency is high.

The case where maximum task group size=1 (#31–35) is the same problem setting as conventional MAPD. In this case, MAPD algorithms do not need to consider trip order so that Algorithm 1 and TP do not have significant difference in makespan. The cases where maximum task group size $= 10$ and 20 (#36–45), we need to solve trip ordering as well as path finding. We can see that makespan gets shorter as maximum task group size increases in Algorithm 1, e.g., makespan in #31, #36, and #41 of Algorithm 1. Algorithm 1 search for collision-free paths considering task order by first solving TSP. As the result, Algorithm 1 achieve $18\% - 22\%$ shorter makespan compare to TP when maximum task group size $= 10$ while there is no significant difference between Algorithm 1 (Manhattan) and (Euclidean) (#36-40). Note again that TP uses a random order to

[1]These task arrival frequencies are lower than the runtime frequency, such that the computation time is sufficient for handling all newly arrived task groups. We shall leave the case where task arrival frequency is higher than runtime frequency to future work.

TABLE II
COMPARISON BETWEEN PROPOSED AND CONVENTIONAL MAPD (TASK GROUP ARRIVAL FREQUENCY=1)

#	Task Group Arrival Frequency	Maximum Task Group Size	Agent Number	Algorithm 1 (Manhattan)		Algorithm 1 (Euclidean)		TP	
				Makespan	Runtime/step	Makespan	Runtime/step	Makespan	Runtime/step
16	1	1	10	3642	5.6ms (20.5s)	3564	5.7ms (20.3s)	3605	3.5ms (12.5s)
17	1	1	20	1925	8.8ms (17.0s)	1920	8.8ms (17.0s)	1949	4.7ms (9.1s)
18	1	1	30	1422	12.0ms (17.1s)	1418	12.3ms (17.4s)	1420	6.3ms (8.9s)
19	1	1	40	1214	16.2ms (19.7s)	1200	16.3ms (19.6s)	1224	9.4ms (11.5s)
20	1	1	50	1137	24.4ms (27.8s)	1133	24.1ms (27.3s)	1136	16.9ms (19.2s)
21	1	10	10	1115	8.3ms (9.2s)	1087	9.0ms (9.8s)	1318	3.4ms (4.5s)
22	1	10	20	639	13.3ms (8.5s)	644	15.1ms (9.7s)	801	5.4ms (4.3s)
23	1	10	30	522	18.3ms (9.6s)	513	19.8ms (10.1s)	639	7.8ms (5.0s)
24	1	10	40	458	23.3ms (10.7s)	470	23.8ms (11.2s)	568	11.7ms (6.7s)
25	1	10	50	435	29.4ms (12.8s)	452	29.7ms (13.4s)	556	19.5ms (10.8s)
26	1	20	10	806	12.6ms (10.2s)	811	14.3ms (11.6s)	1203	3.7ms (4.5s)
27	1	20	20	517	20.8ms (10.8s)	529	19.7ms (10.4s)	785	5.2ms (4.1s)
28	1	20	30	453	25.0ms (11.3s)	444	26.9ms (11.9s)	708	8.0ms (5.7s)
29	1	20	40	418	31.2ms (13.0s)	426	32.6ms (13.9s)	645	16.5ms (10.6s)
30	1	20	50	419	37.7ms (15.8s)	421	37.0ms (15.6s)	629	20.6ms (13.0s)

TABLE III
COMPARISON BETWEEN PROPOSED AND CONVENTIONAL MAPD (TASK GROUP ARRIVAL FREQUENCY=10)

#	Task Group Arrival Frequency	Maximum Task Group Size	Agent Number	Algorithm 1 (Manhattan)		Algorithm 1 (Euclidean)		TP	
				Makespan	Runtime/step	Makespan	Runtime/step	Makespan	Runtime/step
31	10	1	10	3611	5.7ms (20.4s)	3579	5.7ms (20.4s)	3638	3.5ms (12.7s)
32	10	1	20	1921	8.8ms (16.9s)	1915	8.8ms (16.9s)	1937	4.9ms (9.5s)
33	10	1	30	1428	12.0ms (17.1s)	1431	12.1ms (17.4s)	1419	6.7ms (9.6s)
34	10	1	40	1208	16.9ms (20.5s)	1192	16.7ms (20.0s)	1196	10.7ms (12.7s)
35	10	1	50	1117	25.6ms (28.6s)	1125	27.2ms (30.6s)	1118	18.7ms (20.9s)
36	10	10	10	1069	8.4ms (9.0s)	1104	9.1ms (10.0s)	1331	3.4ms (4.5s)
37	10	10	20	640	14.1ms (9.0s)	641	15.4ms (9.9s)	793	5.0ms (4.0s)
38	10	10	30	508	18.6ms (9.4s)	515	19.9ms (10.2s)	622	8.0ms (5.0s)
39	10	10	40	458	24.4ms (11.2s)	458	25.6ms (11.7s)	559	12.0ms (6.7s)
40	10	10	50	430	33.9ms (14.6s)	438	35.5ms (15.6s)	550	22.2ms (12.2s)
41	10	20	10	778	12.7ms (9.9s)	819	13.3ms (10.9s)	1239	3.5ms (4.3s)
42	10	20	20	527	18.2ms (9.6s)	537	19.6ms (10.5s)	797	5.3ms (4.2s)
43	10	20	30	428	26.2ms (11.2s)	441	26.0ms (11.5s)	683	9.5ms (6.5s)
44	10	20	40	404	35.1ms (14.2s)	411	35.7ms (14.7s)	654	17.4ms (11.4s)
45	10	20	50	391	42.1ms (16.5s)	402	42.4ms (17.1s)	628	24.4ms (15.3s)

pick up multiple tasks in an assigned task group. The case where maximum task group size = 20 shows the same trend to that of maximum task group size = 10 where Algorithm 1 achieved $34\% - 38\%$ shorter makespan compare to TP (#41–45). Runtimes(/step) for Algorithm 1 are larger than TP. Nevertheless, these runtimes(/step) are less than 50 [ms] in all agent number cases which is sufficient for online use (runtime/step in #31-45 of Algorithm 1).

Runtime/step tends to increase when task group arrival frequency, maximum task group size, and agent number increase. This is because if task group arrival frequency or agent number increases, collision between agents will be more likely to happen due to the crowded field, which generally makes path finding more time consuming. On the other hand, TSP solver needs more time to solve the problem as maximum task group size increases.

As a whole, Algorithm 1 shows a similar trend that either using Manhattan distance or Euclidean distance for calculating distance matrices does not make a significant difference to the makespan. Comparison between Algorithm 1 and TP shows that Algorithm 1 achieves significantly shorter makespan in the case where the order of picking up multiple tasks needs to be considered, i.e., maximum task group size > 1. Runtime/step for Algorithm 1 is larger than TP. Nevertheless runtime/step is still less than 50 [ms] in all agent number cases which is sufficient for online use.

TABLE IV
VARIOUS PAYLOAD CAPACITY DISTRIBUTION

μ	σ	Makspan	Runtime/step[ms] (Runtime[s])
0.2	0.25	2979	8.5ms (25.3s)
0.5	0.25	1073	14.9ms (16.0s)
0.6	0.25	523	28.6ms (15.0s)
0.7	0.25	417	41.6ms (15.4s)
1.0	0.25	403	38.9ms (15.7s)
1000	0.25	395	42.5ms (16.8s)
-	-	391	39.9ms (15.6s)

B. Agents' Payload Capacity

Next we evaluate our proposed online multi-task MAPD algorithm with different payload capacity constraints of the agents. Table IV shows the evaluation result. We set maximum task group size to 20, task group arrival frequency to 10, and agent number to 50. Map size, goal location, generated tasks are the same as the preceding experiment (in Section V-A). We use gaussian distribution to set payload capacity values for each agent. The parameters for the gaussian distribution are μ and σ. We set σ to 0.25 where μ increases from 0.2 to 1000. For example, $(\mu, \sigma) = (0.2, 0.25)$ means $(\mu, \sigma) = (0.2\times$ maximum task group size, $0.25\times$ maximum task group size$)$ = (4, 5) that agents have payload capacity 4 in average and 5 in standard deviation. Note that we set one agent's capacity = maximum task group size in order to guarantee that at least one agent can

TABLE V
PROPOSED MAPD (10 AGENTS) WITH VARIOUS PICKUP & DROPOFF

Agent Number	Pickup	Dropoff	Makespan	Runtime/step
10	0	0	1067	12.0ms (12.8s)
10	0	1	1095	11.1ms (12.2s)
10	0	5	1164	10.4ms (12.1s)
10	1	0	1152	10.6ms (12.2s)
10	1	1	1160	12.4ms (14.3s)
10	1	5	1216	11.7ms (14.2s)
10	5	0	1398	11.0ms (15.4s)
10	5	1	1414	9.7ms (13.7s)
10	5	5	1467	9.8ms (14.4s)

TABLE VI
PROPOSED MAPD (50 AGENTS) WITH VARIOUS PICKUP & DROPOFF

Agent Number	Pickup	Dropoff	Makespan	Runtime/step
50	0	0	434	49.5ms (21.5s)
50	0	1	471	57.9ms (27.3s)
50	0	5	749	199.4ms (149.3s)
50	1	0	463	48.3ms (22.4s)
50	1	1	471	70.5ms (33.2s)
50	1	5	759	203.2ms (154.3s)
50	5	0	583	86.5ms (50.5s)
50	5	1	596	97.3ms (58.0s)
50	5	5	804	199.8ms (160.6s)

be allocated to any task group. We again evaluate makespan and runtime/step [ms] (total runtime [s]).

We see from Table IV that makespan gets shorter as μ increases. The buttom row at Table IV shows the case without payload capacity constraint for comparison purpose, i.e., payload capacity $= \infty$. Makespan value gets close to the one without payload capacity constraint around $\mu = 0.7$. The case when μ is small (e.g., $\mu = 0.2$), payload capacity for many agents are likely to be smaller than maximum task group size, which cannot satisfy capacity constraint for many task groups. This is similar to the situation where there are small numbers of agents, and thus the makespan is large. In contrast, the case when μ is large (e.g., $\mu = 0.7$), most of the agents have payload capacity larger than arriving task groups, thereby achieving makespan close to the one without payload capacity constraint.

C. Pickup & Dropoff Costs

We further evaluate our proposed online multi-task MAPD algorithm with innegligible pickup & dropoff costs. Tables V and VI show the evaluation results. Maximum task group size and task group arrival frequency are set to 10, while agent number is set to 10 (Table V: small case) and 50 (Table VI: large case). Pickup & dropoff cost are set to three patterns: 0 (no cost, for comparison purpose), 1, and 5. Map size, goal location, generated tasks are same as the first experiment (in Section V-A). We again evaluate makespan and runtime/step [ms] (total runtime [s]).

Both Tables V and VI show that if pickup or dropoff cost increases, makespan increases. Between runtime/step and pickup & dropoff cost, we can see that: In Table V where agent number is small, there is no significant difference in runtime/step in all pickup or dropoff cost cases. On the other hand in Table VI where

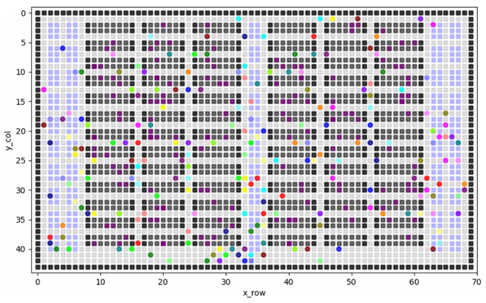

Fig. 2. Example of large scale case: 70×43 field with 200 agents.

TABLE VII
LARGE SCALE CASE

Agent Number	Makespan	Runtime/step[ms] (Runtime[s])
30	2630	17.5ms (46.1s)
50	1696	26.9ms (45.7s)
100	1120	48.0ms (53.8s)
150	997	98.8ms (96.5s)
200	963	225.1ms (216.8s)
250	984	324.4ms (319.2s)
300	1022	513.6ms (524.9s)

agent number is large, increase of dropoff cost drastically increases runtime/step. This is because agents visiting the dropoff location can easily get crowded in the case where agent number is large and cause traffic jam. Between makespan and pickup & dropoff cost, we can see that: In Table V where agent number is small, an increase of dropoff cost slightly increases makespan (less than 10% in all cases) while an increase of pickup cost increases makespan around 30% when pickup cost $= 5$. This is because in the case where there are small number of agents, increase of pickup cost increases makespan more than dropoff cost since pickup event occurs more frequently than dropoff event and only limited number of agents can respond to the pickup requests. In contrast in Table VI where agent number is large, an increase of dropoff cost largely increases makespan (around $40\% - 70\%$ when dropoff cost $= 5$) while an increase of pickup cost increases makespan around $10\% - 30\%$ when pickup cost $= 5$. This is because in the case where there are large number of agents, an increase of dropoff cost can cause much traffic jam at the dropoff location and therefore has a larger impact on the makespan.

As a whole, we find that runtime/step tends to increase in a situation where agent number and dropoff cost are large. We also find that increase of pickup cost can largely increase the makespan when agent number is small; on the other hand, increase of dropoff cost tends to largely increase the makespan when agent number is large.

D. Large Scale Environment

Finally we evaluate our proposed online multi-task MAPD algorithm with large-scale environment and large numbers of agents. Fig. 2 shows an example snapshot and Table VII shows the evaluation result. We set the size of the map to 70×43, maximum task group size to 10, task group arrival frequency to 100, and task number to 2000. We vary the agent number from

30 to 300, and evaluate makespan and runtime/step [ms] (total runtime [s]).

We see that makespan decreases as agent number increases; however, makespan gradually saturates from agent number 200 to 300 because of the high density of agents. Path finding becomes more difficult and time consuming in the situation when density of the field is large (i.e., agent number is large) because high density situation is more likely to cause collisions. Runtime becomes rather large at agent number 200 and above; nevertheless, runtime/step is still less than 1 [s]. The reason why runtime/step gets large above agent number $= 200$ is due again to the high density of agents as mentioned above. This problem occurs regardless of the scale of the problem, i.e., the map size or an agent number. Nevertheless, we see from the experiment that the proposed online multi-task MAPD algorithm finds valid paths in large scale case (e.g., 300 agents) in an online manner by setting proper agent number in terms of field density.

VI. Conclusion

In this letter, we have extended MAPD to a multi-task setting with consideration of payload capacity and pickup & dropoff cost in an online manner and proposed an efficient online multi-task MAPD algorithm. The experiments showed that our proposed algorithm achieved shorter makespan paths than the conventional MAPD algorithm, i.e., Token Passing.

For future work, we aim to extend our problem setting/ solution by considering power constraint for agents, assigning new task groups to non-vacant agents, tasks being grouped according to the system state, agents being selected based on their capacities/locations, and multiple dropoff locations.

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–19, 2008.

[2] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: A complete multi-agent pathfinding algorithm," *J. Artif. Intell. Res.*, vol. 51, pp. 443–492, 2014.

[3] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2019, pp. 1152–1160.

[4] G. Lodigiani, "State of the art on: Multi-agent path finding and multi-agent pickup and delivery," Politecnico di Milano, Italy, 2021. [Online]. Available: http://www.honours-programme.deib.polimi.it/2020-2/Deliverable1/CSE_Lodigiani_SOTA.pdf

[5] C. Henkel, J. Abbenseth, and M. Toussaint, "An optimal algorithm to solve the combined task allocation and path finding problem," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 4140–4146.

[6] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proc. 16th Conf. Auton. Agents MultiAgent Syst.*, 2017, pp. 837–845.

[7] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2015, pp. 324–332.

[8] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artif. Intell.*, vol. 310, 2022, Art. no. 103752.

[9] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11272–11281.

[10] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, 2015.

[11] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding," in *Proc. 11th Annu. Symp. Combinatorial Search*, 2018, pp. 2–9.

[12] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy CBS: Implicit conflict-based search using lazy clause generation," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2019, pp. 155–162.

[13] D. Silver, "Cooperative pathfinding," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2005, pp. 117–122.

[14] T. Huang, S. Koenig, and B. Dilkina, "Learning to resolve conflicts for multi-agent path finding with conflict-based search," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11246–11253.

[15] K. Vedder and J. Biswas, "X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs," *Artif. Intell.*, vol. 291, 2021, Art. no. 103417.

[16] J. Weise, S. Mai, H. Zille, and S. Mostaghim, "On the scalable multi-objective multi-agent pathfinding problem," in *Proc. IEEE Congr. Evol. Comput.*, 2020, pp. 1–8.

[17] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 12409–12417.

[18] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker, "A multi-label A* algorithm for multi-agent pathfinding," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2019, pp. 181–185.

[19] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *Eur. J. Oper. Res.*, vol. 225, no. 1, pp. 1–11, 2013.

[20] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *J. ACM*, vol. 7, no. 4, pp. 326–329, 1960.

[21] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Management Sciences Research Group, Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. 388, 1976.

[22] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Res.*, vol. 6, no. 6, pp. 791–812, 1958.

[23] R. Stern et al., "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proc. 12th Annu. Symp. Combinatorial Search*, 2019, pp. 151–158.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[25] B. Korte and J. Vygen, *Combinatorial Optimization*. Berlin, Germany: Springer, 2006.