Anytime Multi-Task Multi-Agent Pickup and Delivery under Energy Constraint

Fumiya Kudo[1] and Kai Cai[1], Senior Member, IEEE

***Abstract*—Various Multi-Agent Path Finding (MAPF) and its extension, Multi-Agent Pickup and Delivery (MAPD) algorithms have been studied in academia. In the industrial sector, however, automatic safe control of teams of robots and AGVs on factory floors and logistic warehouses for pickup and delivery operations have also been studied intensively. In this paper, we extend our previous work of online multi-task MAPD to a new problem where (i) task can be allocated to any vacant agent independent of the location of that agent — called "anytime task allocation" in this paper, and (ii) each agent is subject to energy constraint. The proposed anytime task allocation MAPD algorithm achieves $5-19\%$ shorter makespan paths compared to the baseline multi-task MAPD in wide range of agent numbers. We also examine the behavior of the proposed multi-task MAPD algorithm under various energy constraint, by changing power limits and energy charge speeds of individual agents. We find that energy charge speed has a large impact on the makespan when power limit is small. We also find that small energy charge speed typically requires a large number of agents in order to achieve the same makespan. These results demonstrate that our proposed multi-task MAPD algorithm can be useful in choosing proper agent numbers in order to achieve prescribed makespans which is more practical as compared to our previous multi-task MAPD.**

***Index Terms*—Multi-agent path finding, multi-agent pickup and delivery, energy constraints, anytime task allocation, factory automation.**

I. Introduction

THERE is a growing need for automation in logistics, particularly involving multiple moving robots (agents) in warehouses and factories due to the increasing labor shortage [1]. To address such a demand, it is crucial to develop advanced algorithms for planning and controlling multi-agent systems to handle tasks dynamically arising in complex environments. In these multi-agent systems, collision-free paths must be provided to ensure that agents reach their destinations without collisions, while minimizing the total travel time. This issue is referred to as Multi-Agent Path-Finding (MAPF), which has recently garnered significant attention [2]. MAPF is an optimization problem focused on determining collision-free paths on a graph for multiple agents with the goal of reducing the overall task-completion time known as *makespan*, or an average number of time steps required to finish all tasks known as *service time*.

Manuscript received April 18, 2024; Revised July 12, 2024; Accepted September 24, 2024. This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments.

[1]The authors are with the Department of Core Informatics, Osaka Metropolitan University, 3-3-138 Sugimoto, Sumiyoshi-ku, Osaka 558-8585, Japan (email: fumiya.kudo.wy@hitachi.com, cai@omu.ac.jp).

Digital Object Identifier (DOI): see top of this page.

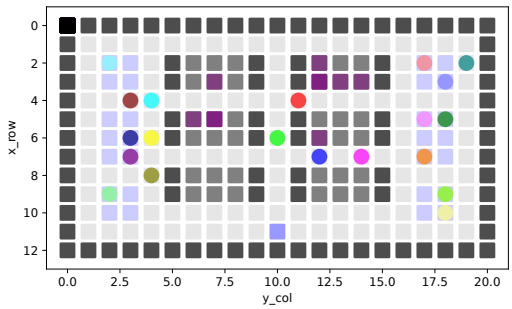

Fig. 1. An illustration of MAPF and MAPD application in logistic warehouse. Agents are depicted by colored circles. The walls and the shelves are depicted by black and gray squares, respectively. Products are placed at shelves and current products to be picked up by agents are depicted by dark purple squares. The dropoff location is depicted by the purple square at the center bottom of the field.

The extension of MAPF — Multi-Agent Pickup and Delivery (MAPD) which is the combination of delivery task allocation with the MAPF problem [3] [4] — has also been studied intensively. An illustrative application of MAPF and MAPD in logistic warehouse is shown in Fig. 1. This is a snapshot of a state, where some of the agents (colored circles) are at their initial locations while the others are moving in the field. When picking tasks arrive which request agents to pick up 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 (dark purple squares). After picking up target products, agents will 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 online/offline task arrival settings with centralized/decentralized and coupled/decoupled approaches [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. In our previous work [10], we proposed an online multi-task MAPD algorithm that effectively solved all the above mentioned issues. Two key practical aspects still await to be resolved. One is "anytime task allocation", by which we mean that a task should be able to be allocated to any vacant agent *independent of the location of that agent*. In [10], the algorithm was limited

to allocating a task to a vacant agent only when that agent is at its initial location. Second, in practice all agents are running on batteries which need to be periodically charged. Hence MAPD algorithms should respect the energy constraint. This issue is not considered in [10].

In this paper, we extend our previous work [10] to a new problem to address multi-task MAPD with *anytime task allocation* and under *energy constraint*. To solve this problem, we propose a new algorithm which allows a task to be allocated to any vacant agent regardless of the current location of that agent. This flexibility of anytime task allocation improves the makespan as demonstrated through extensive experiments. Realizing anytime task allocation logic is technically challenging and requires a shift of implementation paradigm from the previous work [10], as *dynamic* updates are needed for the reserved path information which were *static* in [10]. Moreover, our new algorithm additionally verifies if a vacant agent has sufficient energy to complete an allocated task; if not, the task will be allocated to a different agent. In our setting, we assume that each agent's battery is charged when it is in its initial location. To satisfy the energy constraints is not only a novel conceptual but also a nontrivial technical departure from our previous work [10], because violations of energy constraints can significantly disturb task allocation and impact the existence of viable paths. These technical challenges are successfully addressed in this work. The contributions of this paper are summarized below:

- We tackle an extended version of the multi-task MAPD problem [10] where (i) task can be allocated to any vacant agent independent of the location of that agent (i.e. anytime task allocation), and (ii) each agent is subject to energy constraint. To our best knowledge, this extended multi-task MAPD problem is new and no existing algorithm can be directly applied to solve this problem.
- We propose a new online algorithm to solve the formulated anytime multi-task MAPD problem with energy constraint. In this proposed algorithm, collision-free paths for multiple agents are computed by space-time A* [11] as in our previous work [10]. The challenge here is: in the existing space-time A* the collision-free paths are reserved in advance and the reservation information is not allowed to be changed. However, since anytime task allocation requires dynamical allocation of tasks to vacant agents, the reserved paths must be dynamically updated. Our proposed algorithm effectively solves the problem by suitably incorporating a dynamic updating mechanism into the space-time A*. In addition for energy constraint, we associate to each agent with an energy level and an energy charge speed, and our designed algorithm always checks if enough energy is left for the vacant agent to handle a task at every instant of task allocation.
- We conduct extensive experiment to compare our new multi-task MAPD algorithm with the one in [10]. Special focus is given to two parameters: agent number and task number. Empirical evidence shows that: with the anytime task allocation capability, our new algorithm achieves $5-19\%$ shorter makespan paths compared the one in [10] for a wide range of agent numbers. We also show that as the task number increases, the improvement of makespan increases as well.
- Moreover we change power limit and energy charge speed for individual agents to evaluate makespan. Experiment shows that energy charge speed has a large impact on the makespan when power limit is small. In addition, we fix power limit to investigate the relation between agent number and energy charge speed, as well as their impact on makespan. We find that small energy charge speed typically requires a large number of agents in order to achieve the same makespan. This result demonstrates that our new algorithm can be useful in choosing proper agent numbers in order to achieve desired makespans.

The remaining sections of this paper are structured as follows. We first present relevant methods and algorithms for MAPF and MAPD (Section 2). Then, we present our problem setting: online multi-task MAPD with anytime task allocation capability under energy constraint (Section 3). Next, we describe our proposed online algorithm for solving the multi-task MAPD in detail (Section 4). We then conduct extensive experiment in which we apply our proposed algorithm with various parameters (Section 5). Finally, we conclude the paper (Section 6).

II. Related Work

We present and compare various methods and algorithms for MAPF, MAPD and their related problems. Approaches to MAPF problems can be classified according to several factors, such as whether tasks arrive online/offline, the approaches are centralized/decentralized as well as coupled/decoupled. Online task arrival means that the problem is a *lifelong* setting where tasks can enter the system at any time.[1] Consequently, the allocation of agents to tasks and the planning of paths cannot be completed beforehand but must be carried out in real-time during the operation. In contrast, offline means that we know all the information about arriving tasks *a priori*. 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]. Coupled approach is a complete MAPF algorithm which can find optimal solutions [12] [13] [14]. However, finding an optimal solution is NP-hard, and consequently these optimal algorithms do not scale in agent numbers. In contrast, decoupled approach is an incomplete MAPF algorithm that quickly finds sub-optimal solutions [11]. Conflict-based Search (CBS) [12] is a representative coupled and centralized MAPF algorithm based on a two-level MAPF algorithm. CBS is one of state-of-the-art MAPF algorithms that has many variants [13] [14]. Cooperative A* (CA*) [11] is a representative decoupled and centralized MAPF algorithm which uses a special type of A* called space-time A*. CA* is widely used in

[1]Here *lifelong* means that tasks can enter the system at any time step. This is different from the anytime task allocation considered in this paper, which means that a task can be allocated to any vacant agent at anytime.

practice due to its small runtime that there are many variants: Hierarchical Cooperative A* (HCA*), Windowed Hierarchical Cooperative A* (WHCA*) [11]. Priority inheritance with backtracking (PIBT) [8] is a representative decoupled and decentralized MAPF algorithm. ML-MAPF [15] addresses MAPF based on a machine learning methodology. GA-based MAPF [16] is a multi-objective variant of MAPF that utilizes Genetic Algorithms (GA). Multi-Goal MAPF [17] studies the problem where agents travel multiple destinations. Anytime MAPF [18] [19] first finds an initial solution fast and then repeatedly replans the paths of subsets of agents. *Anytime* in these works means that a solution is continuously being improved by MAPF algorithm. In contrast, anytime task allocation considered in this paper is different and means that a task can be allocated to any vacant agent at anytime (independent of the location of that agent).

MAPD 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 algorithms for MAPD. Token Passing (TP) and Token Passing with Time Swaps (TPTS) [6] are decoupled and centralized approaches based on CA* [11]. TP and TPTS are incomplete; however, they are widely used for their speed in real-world online applications. Multi-Label A* (MLA) [20] is an enhanced version of TPTS. TCBS [5] is a complete MAPD algorithm based on CBS which can find optimal solution; however, examined environment is small (agent number < 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-Horizon Collision Resolution (RHCR) [9] is one of state-of-the-art MAPD approaches based on WHCA* [11].

Recently, we studied an MAPD problem which takes into account several aspects of real-world industrial applications, including MAPD problem with multiple task allocation, payload capacity constraints, pickup & dropoff constraints [10]. This work extends the conventional single-task MAPD problem to a multi-task setting. However, the proposed algorithm in [10] did not address anytime task allocation (in that algorithm a task can be allocated to an agent only when that agent is in its initial location, which may be overly conservative), nor can it address the energy constraint of individual agents. On the other hand, there are many variants of path planning algorithms for UAVs (Unmanned Aerial Vehicles) [21] and UGVs (Unmanned Ground Vehicles) [22] considering energy constraints. The coverage path planning problem of UAVs with limited energy is studied in [21]. An integrated path planning and power management problem for a solar-powered UGV is examined in [22]. Different types of environments in MAPF including energy constraints is introduced in [23]. Furthermore, multi-task MAPD with loading capacity which may be viewed an alternative representation of the energy constraint is studied in [24]. However, these works do not consider the multi-task MAPD problem, nor do the solution algorithms have anytime task allocation capability.

III. Problem Setting

In this section, we formalize our anytime multi-task MAPD problem, whose aim is to find collision-free paths for energy-constrained agents to accomplish tasks arriving in an online manner that can be allocated to any vacant agents regardless of their locations.

Consider a graph $G = (V, E)$, whose vertices $v \in V$ correspond to physical locations in the field and whose edges in E correspond to connections between neighboring locations along which the agents can move. We assume that the graph G is bidirected (each edge is bidirectional) and connected (every vertice can reach every other vertice). Also consider a set of m agents $A = \{a_1, a_2, ..., a_m\}$ 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 $v_i(0) = v_i^{\text{init}}$. 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 (i.e. one timestep). Moreover, we consider that each agent a_i has an *energy level* $\alpha_i(t) \geq 0$ at timestep t, and a (constant) *power limit* $\bar{\alpha}_i$. We assume that every $\bar{\alpha}_i$ is large enough for every agent to complete at least one (arbitrary) task group (introduced below). When it is at its initial location v_i^{init}, its battery is charged and energy level increases as

$$\alpha_i(t+1) = \begin{cases} \alpha_i(t) + \beta_i & \text{if } \alpha_i(t) + \beta_i < \bar{\alpha}_i \\ \bar{\alpha}_i & \text{if } \alpha_i(t) + \beta_i \geq \bar{\alpha}_i \end{cases} \quad (1)$$

where $\beta_i \geq 1$ is the (constant) *charge speed* for agent a_i. If agent a_i is not at its initial location, i.e. $v_i(t) \neq v_i^{\text{init}}$, then its energy level decreases one unit per timestep (no matter its action is *move* or *wait*):

$$\alpha_i(t+1) = \alpha_i(t) - 1. \quad (2)$$

A task s requests an agent to pick up a target product located at a shelf $v_s \in V$. In our multi-task MAPD setting, there are generally multiple tasks which request to be picked up together and are gathered in a *task group* $TG_l = \{s_1^l, \ldots, s_{n_l}^l\}$. Agents are allocated task groups instead of a single task. Note that a task group has at least one task. Each agent a_i has payload capacity c_i ($i \in [1, m]$). We assume that at least one agent a_i exists who has enough payload c_i to handle the largest task group, i.e. $(\exists i \in [1, m])c_i \geq \max_l n_l$. Moreover, we consider *anytime task allocation*, meaning that at any timestep t a task group may be allocated to any agent a_i as long as the agent is vacant (with no task group already allocated and not yet finished) and has sufficient energy $\alpha_i(t)$ to complete the task group and return to its charging location v_i^{init} (i.e. $\alpha_i(t)$ will not decrease to 0 before agent a_i returns to v_i^{init}). We emphasize that anytime task allocation allows allocating a task group to an agent independent of the location of that agent: namely the agent may be idle and charging itself at its initial location, or the agent has finished its previously allocated task group and on its way to move back to its initial location.

After a task group TG_l is assigned to an agent a_i, this agent moves from its current location $v_i(t)$ to the requested shelf locations $v_1^l, \ldots, v_{n_l}^l$ to pick up the target products, and then moves to the unique goal location $v_g \in V$ to drop off those products. The agent a_i will then move back to its initial location, unless a new task group is allocated to it.

Pickup & dropoff actions may take some timesteps in practice (as the actions $move$ and $wait$, each of which takes a unit timestep). Moreover, we consider that this anytime multi-task MAPD problem is online, i.e. we do not know task group information *a priori*, and new task groups can be added to a task group set TASKLIST randomly at any timestep. We assume that TASKLIST obeys *quasi-FIFO subject to payload and energy constraints*. This means that the allocation of task groups is *tried* according to the order of their arrivals; however, an allocation is *confirmed* only when a vacant agent has enough capacity and energy. If no such an agent is currently available, the task group remains in the queue *and* keeps its order (based on its arrival time) to be tried at the next timestep.

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 [25]), 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 [25]), 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 our anytime multi-task MAPD problem is to compute collision-free paths for the agents under energy constraint to accomplish task groups allocated at anytime to vacant agents and maximize the throughput, i.e. minimize the *makespan* (difference between the first release time and the last completion time).

IV. PROPOSED ANYTIME MULTI-TASK MAPD ALGORITHM UNDER ENERGY CONSTRAINT

In this section, we present our proposed algorithm to solve the anytime multi-task MAPD problem under energy constraint. Our proposed algorithm is the extension of the TSP-based online multi-task MAPD proposed in our previous work [10]. The algorithm in [10] in turn extends the conventional single-task MAPD [6] to a *multi-task* setting where each agent is allocated with multiple tasks, payload capacity of each agent is considered as a constraint, and pickup & dropoff times are considered as cost. In multi-task MAPD, we need to consider a trip order for the agents in order to maximize the throughput while avoid collision among them. Our previous work deals this problem by suitably integrating space-time A*-based MAPF and TSP solver. In this paper, we further improve our previous algorithm in [10] by implementing *anytime task allocation logic* as well as extending the problem setting to consider *energy constraint*. The pseudo-code of this algorithm is presented in Algorithm 1. We color-coded the parts of the pseudo-code where major changes are made from the prior algorithm in [10]. Below is key notation used:

- VACANTAGENTS is a list of vacant agents to which a task group can be allocated. Such agents have no task group allocated and may be in any location (in particular need not be in the initial location as required in [10]).
- RESERVE holds reserved paths of already task-allocated agents for them to complete their tasks and return to their initial locations. RESERVE is a hash map consisting of two elements: timesteps and location coordinates.

Now we describe the main ideas for Algorithm 1 to realize two new functionalities: anytime task allocation and compliance to energy constraint.

Algorithm 1 Anytime Multi-Task MAPD under Energy Constraint

Input: graph G, set of agents $A = \{a_1, \ldots, a_m\}$, initial locations v_i^{init}, initial energy levels α_i^{init}, power limits $\bar{\alpha}_i$, energy charge speeds β_i, payload capacities c_i ($i \in [1, m]$)
Output: RESERVE

1: initialize agent $a_i \leftarrow (v_i^{\text{init}}, \alpha_i^{\text{init}}, \bar{\alpha}_i, \beta_i, c_i)$
2: TASKLIST $\leftarrow \emptyset$
3: VACANTAGENTS $\leftarrow \{a_1, a_2, \ldots, a_m\}$
4: RESERVE $\leftarrow \emptyset$
5: **for** $t = 1$ to MAXTIME **do**
6: TASKLIST $\leftarrow$ TASKGROUPGENERATOR()
7: **for** $l = 0$ to size(TASKLIST) **do**
8: $A' =$ CAPACITY(VACANTAGENTS, TG_l)
9: **for** $i = 0$ to size(A') **do**
10: $M =$ CALCDISTANCEMATRIX(a_i, TG_l, G)
11: $p \leftarrow$ SOLVETSP(a_i, M)
12: $p' \leftarrow$ RESOLVECOLLISION(p, G, RESERVE)
13: **if** length(p') $\leq \alpha_i(t)$ **then**
14: ALLOCATE(TG_l, a_i)
15: UPDATE(RESERVE, p', a_i)
16: TASKLIST.delete(TG_l)
17: VACANTAGENTS.delete(a_i)
18: **break**
19: **end if**
20: **end for**
21: **end for**
22: MOVEAGENTS(A)
23: ENERGYUPDATE(A)
24: VACANTAGENTUPDATE(A, VACANTAGENTS)
25: **end for**

Anytime Task Allocation: In our previous work [10], the next task group is allowed to be allocated to an agent only when the agent completes the previously allocated task group *and* returns to its initial location. On the other hand in Algorithm 1, the next task group can be allocated to an agent as soon as the agent completes the previously allocated task group, i.e. after the agent drops off products at the goal location (line 24). This logic allows agent to be allocated with the next task group anytime during the agent en route to its initial location. Our previous work [10] uses space-time A* which reserves the paths to avoid collision. The idea for implementing anytime task allocation logic is to *dynamically* update the reservation information by overwriting the previously reserved paths to the new ones (line 15).

Energy Constraint: In Algorithm 1, to comply with the energy constraint on each agent, it is checked if there is enough energy left for an agent to complete a task group (line 13). By completion of a task group, the agent should be able to drop off the products at the goal location as well as return to its initial location for charging. A task group can also be allocated when agent is being charged at its initial location as long as the agent has enough energy to complete the task

group. The energy levels of all agents are updated at every time step (line 23).

The main parts of Algorithm 1 are explained as follows.

1) lines 9-20: For each vacant, capacity-sufficient agent a_i, verify if it has enough energy to complete the task group TG_l and return to its initial location a_i^{init}; if so, TG_l will be allocated to a_i (the first such agent this verification becomes true). To this end, lines 10-12 first computes a collision-free path for a_i to complete TG_l and then return to a_i^{init}. CALCDISTANCEMATRIX function [11] calculates the distance matrix between tasks in the same task group TG_l using A* [26]. SOLVETSP function [27] [28] finds the shortest path p by solving TSP for the task group TG_l including the goal location and the current location $v_i(t)$ of agent a_i. RESOLVECOLLISION function [11] converts (if possible) 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* quits resolving conflict for the current agent, i.e. breaks the current for-loop and moves onto the next vacant, capacity-sufficient agent.

   Once a collision-free path p' is found, line 13 checks if the length of the p' is equal to or smaller than agent a_i's current energy level $\alpha_i(t)$. If so (i.e. a_i has sufficient energy to follow path p' to complete TG_l and return to a_i^{init}), then task group TG_l is allocated to a_i by the ALLOCATE function on line 14. The UPDATE function on line 15 is the key to achieve anytime task allocation logic, by suitably updating the RESERVE as follows:
   c) If a path of agent a_i was already reserved in RESERVE (i.e. a_i has completed its previously allocated task group but has not yet returned to its initial location), then that registered path is first removed and the newly accepted path p' (computed on line 12) is added to RESERVE.
   d) If there is no path of agent a_i in RESERVE (i.e. a_i has returned to its initial location), then the newly accepted path p' is added to RESERVE.

   This UPDATE function sharply distinguishes from our previous algorithm in [10] where RESERVE is static until an agent returns to its initial location.
2) line 22-24: Once the task groups are either all allocated or only partially allocated due to lack of vacant agents with sufficient payload capacities or energy levels, MOVEAGENTS function moves task-allocated agents one step forward based on the reserved paths in RESERVE. After the moves, ENERGYUPDATE function updates energy levels $\alpha_i(t)$ of every agent a_i according to (1) and (2). VACANTAGENTUPDATE function updates the list of vacant agents, namely those that have completed their allocated task groups after the moves. [2]

[2]We assume that all computation between line 6 and 24 can be completed in one timestep. This assumption is empirically justified in Section V below.

Remark 1: The complexity of Algorithm 1 is $O(N^3m^2C^3)$, where N is the total number of tasks (i.e. sum of tasks in all task groups), m is the number of agents, C is the number of cells in the field, i.e. number of nodes in the graph $G = (V, E)$. To see this, observe that the main part contributing to the complexity is the for-loop of lines 7-20. In particular, the following lines of computation have high complexity:

- Line 10 (CALCDISTANCEMATRIX): $O(N^2C^2)$
  The distance matrix has at most N^2 entries, and each entry is obtained by an A* computation of $O(C^2)$ [29].
- Line 11 (SOLVETSP): $O(N^2)$
  The TSP has at most N tasks to route, for which the 2-opt algorithm we adopt has the complexity $O(N^2)$ [28].
- Line 12 (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 12. Since the for-loop of lines 7-20 can happen no more than Nm times, the total complexity of Algorithm 1 is $O(N^3m^2C^3)$. Note that this complexity is m factor higher than the complexity of our previous algorithm in [10]. This factor comes from checking if a vacant agent has enough energy levels for completing the task group to be allocated and then return to its initial location, which was not needed in [10] where agents are impractically assumed to have infinite energies.

Remark 2: We provide a correctness proof for Algorithm 1: i.e. eventually all task groups are completed and all agents return to their initial locations. First by the assumption that the graph G is bidirected and connected, every agent can reach any location in G, so there is a feasible path for every agent to complete any task group and return to its initial location. By the property of the space-time A* algorithm [11], which we employ for planning paths of multiple agents, no collision or deadlock will exist (indeed, potential collision/deadlock is avoided by delaying an agent by a number of timesteps). Moreover, the incoming task groups are stored in TASKLIST that obeys quasi-FIFO subject to payload and energy constraints. This implies that earlier-arrival task groups in TASKLIST will always be tried for allocation to an agent before later-arrival task groups. Since it is assumed that there exists at least an agent that has enough payload to handle the largest task group and every agent's power limit is enough to complete any one of the task groups, we conclude that a vacant agent with enough payload and energy will eventually be available to take any task group in TASKLIST. Therefore, eventually all task groups can be completed and all agents return to their initial locations.

V. EXPERIMENTS

In this section, we evaluate our proposed anytime multi-task MAPD algorithm (anytime MT-MAPD) with respect to a few key parameters: task group arrival frequency, agent number,

task number, energy level, and energy charge speed. The size of the field is 36×22, and the goal (dropoff) location is $(18, 21)$.

A. Anytime Task Allocation

First, we compare anytime MT-MAPD (Algorithm 1) with our previous multi-task MAPD algorithm (TSP-based MT-MAPD) in [10] and PIBT [8], in terms of makespan and runtime/step. In this experiment we focus on the effect of the newly implemented anytime task allocation logic, and for this reason assume that agents always have sufficient energy among all algorithms. The effect of energy constraint will be studied in the next experiment below.

We generate a sequence of 1000 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 5. We set task group arrival frequency from 0.2 to 10. We also vary agent number from 10 to 50.

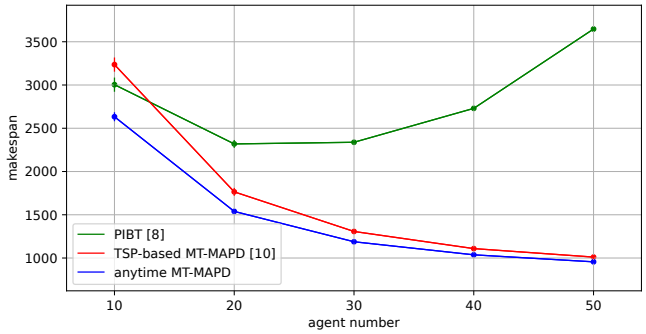

Fig. 2. Makespan comparison among anytime MT-MAPD, TSP-based MT-MAPD [10], and PIBT [8]. Agent number is varied from 10 to 50, while task group arrival frequency and task number are set to 10 and 1000, respectively. All experimental settings are performed in 100 instances with initial locations of agents set randomly. Vertical bars on each plot show standard deviations. The vertical axis starting from 896 is lower bounded by the shortest makespan value of all cases in 100 instances. (Note that PIBT is a decentralized MAPF algorithm that tends to become inefficient in resolving collisions/deadlocks as the number of agents increases.)

We evaluate makespan [timestep] and runtime/step [ms] (total runtime [s]). The comparison of the makespan when task group arrival frequency is 10 is shown in Fig. 2. We can see that anytime MT-MAPD achieves the shortest makespan compared to the TSP-based MT-MAPD [10] and PIBT [8] for a wide range of agent numbers. In specific, anytime MT-MAPD achieves $5-19\%$ shorter makespan compared to the TSP-based MT-MAPD [10], which shows the benefit of using anytime task allocation logic. Moreover, for a fixed number of tasks (1000 in this case), the improvement of makespan is more substantial when agent number is smaller; this is because fewer agents means that each agent needs to be allocated with more tasks, and anytime logic allows such allocation to happen soon after the agents finish their previous tasks (and thus save more time without needing to go back to their initial locations). The detailed results are presented in Table I. "Makespan" and "(min)" in Table I indicate the average and the lower bound (calculated by the shortest makespan) values in 100 instances, respectively. It is verified that anytime MT-MAPD achieves 8.3% shorter makespan and 14.1% lower total runtime on average as compared to the TSP-based MT-MAPD [10] for a wide range of agent numbers and task group arrival frequencies. Total runtime also becomes shorter because makespan gets shorter.

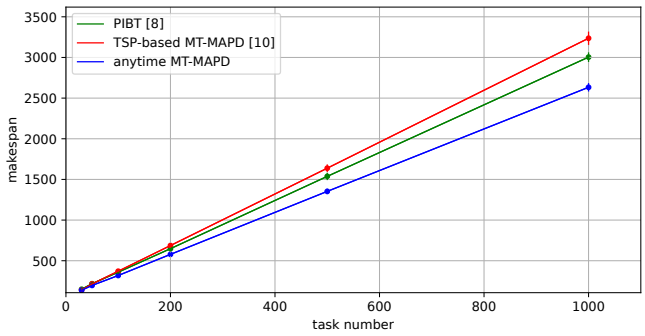

Fig. 3. Makespan comparison among anytime MT-MAPD, TSP-based MT-MAPD [10], and PIBT [8]. Task number is varied from 30 to 1000, while agent number and task group arrival frequency are set both to 10. All experimental settings are performed in 100 instances with initial locations of agents set randomly. Vertical bars on each plot show standard deviations. The vertical axis starting from 107 is lower bounded by the shortest makespan value of all cases in 100 instances.

In addition, we examine the effect of different task numbers on the makespan. For this, we set agent number and task group arrival frequency both to 10, and vary task number from 30 to 1000. The result is shown in Fig. 3. We can see that the makespan of anytime MT-MAPD is shorter than the TSP-based MT-MAPD [10] and PIBT [8] for a wide range of task numbers, thanks again to anytime task allocation logic. Observe that the improvement of makespan increases linearly as task number increases; this is due to a similar reason to the one explained above: Namely, as the number of task increases, more tasks need to be allocated to agents, and in such situations anytime logic allows these allocations to occur sooner and thereby achieves more efficiency.

To summarize this first experiment, anytime MT-MAPD with anytime task allocation always achieves better makespan as compared to the TSP-based MT-MAPD [10]; the effect of this anytime logic is more substantial when the ratio of task number to agent number is large.

B. Energy Constraint

In this second experiment, we study the behavior of anytime MT-MAPD under energy constraint, in particular with respect to different power limits $\bar{\alpha}_i$ and charge speeds β_i.

We set task number to 1000, task group arrival frequency to 0.5, and agent number to 30. We vary power limit $\bar{\alpha}_i$ from 300 to 2000 and charge speed β_i from 1 to 10. The experiment result is displayed in Fig. 4. Each curve shows the makespan (vertical axis) with respect to different combination of power limits (horizontal axis) and charge speeds (several values shown in the legend). The bottom black line (makespan$=1209$) is the case with no energy constraint (i.e. $\bar{\alpha}_i = \infty$), displayed here for the purpose of comparison.

It is observed that, compared to the case without energy constraint, makespan under energy constraint is larger in all

TABLE I
COMPARISON BETWEEN PROPOSED AND CONVENTIONAL MAPD

Task Group Arrival Frequency	Agent Number	TSP-based MT-MAPD [10]		anytime MT-MAPD (Proposed)		PIBT [8]	
		Makespan (min)	Runtime/step [ms]([s])	Makespan	Runtime/step	Makespan	Runtime/step
0.2	10	3244 (3103)	62.6ms (203.0s)	2655 (2551)	45.3ms (120.4s)	3034 (2881)	59.9ms (181.7s)
0.2	20	1822 (1716)	15.2ms (27.7s)	1787 (1679)	12.8ms (23.0s)	2381 (2182)	46.0ms (109.8s)
0.2	30	1781 (1665)	14.1ms (25.1s)	1786 (1692)	14.0ms (25.1s)	2418 (2158)	50.9ms (126.5s)
0.2	40	1784 (1660)	14.6ms (26.0s)	1789 (1691)	15.2ms (27.3s)	2754 (2244)	62.4ms (174.6s)
0.2	50	1783 (1694)	15.9ms (28.4s)	1787 (1662)	16.8ms (30.1s)	3245 (2516)	71.8ms (234.6s)
1	10	3229 (3083)	110.8ms (357.8s)	2630 (2535)	104.1ms (273.8s)	2999 (2988)	109.7ms (329.0s)
1	20	1776 (1676)	96.7ms (171.7s)	1551 (1482)	95.3ms (147.9s)	2328 (2056)	114.2ms (266.0s)
1	30	1314 (1243)	90.4ms (118.9s)	1197 (1131)	86.2ms (103.3s)	2328 (2130)	119.6ms (278.4s)
1	40	1118 (1032)	85.9ms (96.0s)	1047 (977)	85.1ms (89.1s)	2743 (2335)	126.5ms (347.0s)
1	50	1027 (952)	87.6ms (90.0s)	969 (913)	83.4ms (80.8s)	3621 (3192)	221.6ms (800.3s)
10	10	3236 (3044)	153.4ms (496.6s)	2633 (2519)	122.5ms (322.5s)	3004 (2970)	192.4ms (578.0s)
10	20	1765 (1665)	155.2ms (273.9s)	1539 (1470)	116.5ms (179.3s)	2319 (2171)	181.4ms (420.5s)
10	30	1307 (1237)	169.3ms (221.2s)	1188 (1123)	114.0ms (135.4s)	2338 (2118)	132.6ms (310.2s)
10	40	1109 (1038)	145.7ms (161.6s)	1037 (971)	114.3ms (118.6s)	2730 (2415)	139.3ms (380.2s)
10	50	1011 (963)	157.2ms (158.9s)	956 (896)	111.7ms (106.8s)	3647 (2893)	262.7ms (960.7s)

combinations of power limits and charge speeds. This is expected, since energy constraint forces a vacant agent to be abandoned from task allocation if it does not have enough energy to complete that task group. Moreover, we can see that for smaller power limits, charge speed plays a significant role in affecting makespan. This is because with small power limits, agents frequently need to go back to their initial locations to recharge themselves before they can be allocated with a new task group; then the faster their powers can be charged, the faster they can be ready for task allocation, which result in shorter makespan.

In addition, as the power limit increases, not only makespan becomes smaller but also the influence of charge speed on the makespan becomes negligible. This is due to the fact that with large power limits, agents can go on completing many task groups without recharging, so very few times of recharging is needed which makes charge speed insignificant. We note that the smallness/largeness of power limit is relative to the environment, i.e. field size, agent number, and task number.

To conclude the observations from this second experiment, we see that if either the power limit or the charge speed is large (relative to the environment), the makespan tend to be close to the case without energy constraint.

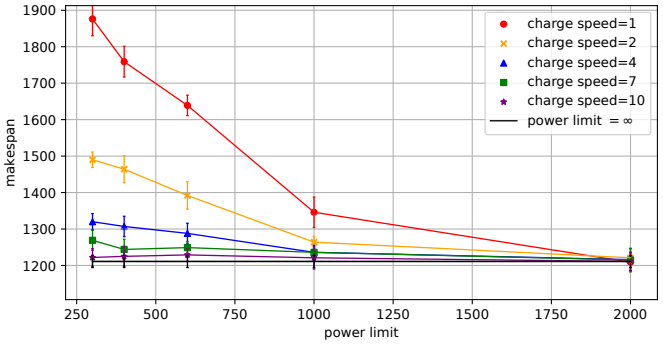

Fig. 4. Makespan achieved by anytime MT-MAPD with different values of power limits and charge speeds. Task number, task group arrival frequency, and agent number are set to 1000, 0.5, and 30, respectively. All experimental settings are performed in 10 instances with initial locations of agents set randomly. Vertical bars on each plot show standard deviations. The vertical axis starting from 1143 is lower bounded by the shortest makespan value for the case with no energy constraint (i.e. $\bar{\alpha}_i = \infty$) in 100 instances.

C. Agent Number

Finally, we investigate the impact of different agent numbers on makespan under energy constraint. As shown in the preceding experiment (Fig. 4), energy constraint has more impact for small power limits, so in this last experiment we focus on power limit $= 300$ and several different values of charge speed (shown in the legend of Fig. 5). We set task number to 1000, task group arrival frequency to 0.5, and vary agent number from 5 to 60. The obtained result is displayed in Fig. 5. The bottom black curve is the case with no energy constraint (i.e. power limit $= \infty$), displayed for the purpose of comparison.

Similar to Fig. 4, in all cases where energy constraint exist, makespan is larger than that in the case where no energy constraint is placed (for the same reason already explained in the preceding subsection). Nevertheless, it is interesting to observe that as the agent number increases, makespan reduces monotonically to become very close to the makespan achieved without energy constraint (and the role played by charge speed also becomes negligible). The reason for this observation is as follows. First, more agents means that each agent is allocated with fewer task groups, which in turn means that each agent needs very few times of recharging, so change speed is less important. Second, when there are more agents in the field, the field would become denser which would typically result in performance saturation. Interestingly, in this case of small power limit (300), agents cannot be in the field for very long, which has the effect of reducing the field density. Hence this balance between agent number and power limit results in performance under energy constraint converging to that without energy constraint.

We end this subsection by remarking on a reverse problem: Suppose that we are given a desired makespan to be achieved, and our task is to determine the agent number needed to achieve the given makespan (where agents are subject to certain level of energy constraint). Such a problem may be effectively resolved by drawing in Fig. 5 a horizontal line for the value of the desired makespan, and according to the

power limit and charge speed of the available agents, we can read off the needed number of agents. Such *design problem* of agent number may be useful in situations where there are abundant agents to be used, and for each specific task with a desired performance, the required number of agents needs to be determined. As demonstrated in Fig. 5, our proposed algorithm can be useful for addressing such problems as well.

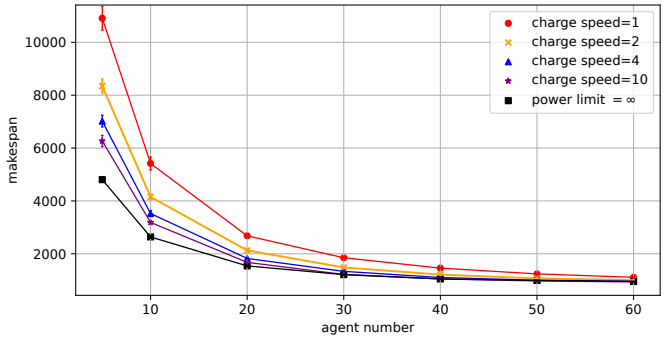

Fig. 5. Makespan achieved by anytime MT-MAPD with different values of agent numbers and energy charge speeds. Task number, task group arrival frequency and power limit are set to 1000, 0.5, and 300, respectively. All experimental settings are performed in 10 instances with initial locations of agents set randomly. Vertical bars on each plot show standard deviations. The vertical axis starting from 926 is lower bounded by the shortest makespan value for the case with no energy constraint (i.e. $\bar{\alpha}_i = \infty$) in 100 instances.

VI. Conclusions

In this paper, we have developed a new online multi-task MAPD algorithm which distinguishes itself from the literature with two novelties: anytime task allocation and under energy constraint. Extensive experimental evaluations have shown the effectiveness of our algorithm on improving makespan as compared to the existing work. Properties and behaviors of the algorithm with respect to varying numbers of agents, tasks, power limits, and charge speeds are thoroughly examined. For future work, we aim to extend our problem setting to uncertain and dynamic environment where there may exist many uncontrolled agents, such as humans or human-driven vehicles.

References

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

[2] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *Journal of Artificial Intelligence Research*, 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 *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2019.

[4] G. Lodigiani, "State of the art on: Multi-agent path finding and multi-agent pickup and delivery," *Politecnico di Milano*, 2021.

[5] C. Henkel, J. Abbenseth, and M. Toussaint, "An optimal algorithm to solve the combined task allocation and path finding problem," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 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," *arXiv preprint arXiv:1705.10868*, 2017.

[7] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proceedings of the international conference on automated planning and scheduling*, vol. 25, 2015, pp. 324–332.

[8] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[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 *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[10] F. Kudo and K. Cai, "A tsp-based online algorithm for multi-task multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, 2023.

[11] D. Silver, "Cooperative pathfinding," in *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, vol. 1, no. 1, 2005, pp. 117–122.

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

[13] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding," in *Eleventh Annual Symposium on Combinatorial Search*, 2018.

[14] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy cbs: implicit conflict-based search using lazy clause generation," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 155–162.

[15] T. Huang, S. Koenig, and B. Dilkina, "Learning to resolve conflicts for multi-agent path finding with conflict-based search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 246–11 253.

[16] J. Weise, S. Mai, H. Zille, and S. Mostaghim, "On the scalable multi-objective multi-agent pathfinding problem," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[17] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, 2021, pp. 12 409–12 417.

[18] K. Vedder and J. Biswas, "X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs," *Artificial Intelligence*, vol. 291, p. 103417, 2021.

[19] T. Huang, J. Li, S. Koenig, and B. Dilkina, "Anytime multi-agent path finding via machine learning-guided large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9368–9376.

[20] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 181–185.

[21] H. Wang, D. Shi, Y. Wu, L. Li, N. Li, and J. Xu, "A balanced shadow-following coverage path planning approach under energy constraints," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021, pp. 1599–1606.

[22] A. Kaplan, N. Kingry, P. Uhing, and R. Dai, "Time-optimal path planning with power schedules for a solar-powered ground robot," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 1235–1244, 2017.

[23] J. Geens, J. Ignoul, W. Lenaerts, and E. Goossens, "Implementing multi-agent system behaviours for overcoming energy constraints and obstacles in the packet-world," in *Proceedings Scientific Training in Multi-Agent Systems Workshop STMAS 2021*, 2021, p. 85.

[24] B. Coltin, "Multi-agent pickup and delivery planning with transfers." Ph.D. dissertation, Carnegie Mellon University, USA, 2014.

[25] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[26] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

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

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

[29] B. Korte and J. Vygen, *Combinatorial Optimization*. Springer, 2006.