Robotics and Autonomous Systems 146 (2021) 103871

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Coordination approaches for multi-item pickup and delivery in logistic scenarios

Antonello Contini, Alessandro Farinelli *

Department of Computer Science, University of Verona, Strada le Grazie 15, Verona, Italy

ARTICLE INFO

Article history:
Available online 14 August 2021

Keywords:
Multi-Robot systems
Multi-Agent Path Planning
Task allocation
Logistic applications

ABSTRACT

We focus on the Multi-Robot pickup and delivery problem for logistic scenarios that recently received significant attention from the research community. In particular we consider an innovative variant of the pickup and delivery problem where robots can deliver, in a single travel, multiple items. We propose a decentralized coordination algorithm based on a token passing approach. Our algorithm allocates delivery tasks (i.e., an aggregation of items to be delivered by a single robot) to the Multi-Robot System avoiding conflicts among the robots. In more detail, we show that our approach generates conflict-free paths for the Multi-Robot system requiring weaker assumptions on the operational area compared to previous approaches. We empirically evaluate the proposed method on three different scenarios, including the production line of a smart factory, comparing the performance of our decentralized method against two centralized approaches. Results show that our approach finds solutions of similar quality (in terms of makespan and travel distance) reducing the associated computational effort.

1. Introduction

The multi-agent pickup and delivery (MAPD) problem requires a set of agents to deliver a set of items to relevant locations. This problem is particularly important for logistic scenarios where robots should move goods from storage locations to production areas [1,2] and many approaches have been proposed to effectively coordinate the robots involved in the operations.

In particular, a series of approaches [3,4] are based on the concept of well-formed infrastructures [5], where the operational area must meet a set of sufficient conditions to guarantee that the MAPD instance is solvable. While these approaches successfully address the MAPD problem, they focus on situations where a single agent can transport a single item for every travel. This is a reasonable assumption for several important scenarios, for example large automated warehouses where robots must transport a single pod to a packaging area, like the system used by Amazon [1].

However, in other scenarios it may be beneficial to have more items delivered in one single travel if the capacity of the platform allows this to happen. For example, imagine a production line where different types of objects must be delivered to different production areas and mobile robots are used to transport such objects from a storage area to the specific production areas. If robots can carry more than one object in one single travel this would result in a more efficient transportation system (i.e., minimizing the time to transport the objects and the travel distance for the robots).

To address this issue, we consider a novel variant of the MAPD problem, where a transportation task consists of picking up multiple items, each with a (possibly) different delivery destination. We propose a decentralized algorithm to solve this variant that can allocate multi-item transportation tasks to the robots computing conflict-free paths. Moreover, our algorithm tries to optimize the makespan (i.e., the time required by the team of robots to complete the tasks), even though no optimality guarantee is given for this criterion. We consider the makespan as it is one of the most common metric used when evaluating solutions to the Multi-Agent Path Finding (MAPF) problem [6,7].

To validate our distributed approach we consider two centralized algorithms that solve the same problem and compare the algorithms in three different logistic scenarios. Such scenarios include a benchmarking grid map and two configurations of a prototype production line for a smart factory.[1] The quantitative

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: antonello.contini@univr.it (A. Contini), alessandro.farinelli@univr.it (A. Farinelli).

[1] The Industrial Computer Engineering (ICE) laboratory for Industry 4.0 of Verona University https://www.icelab.di.univr.it/?lang=en.

https://doi.org/10.1016/j.robot.2021.103871
0921-8890/

evaluation is performed in a realistic simulation environment built using ROS (Robot Operating System).

In summary, this paper makes the following contributions to the state of the art:

(i) we present a novel variant of the MAPD problem, where each transportation task requires the delivery of multiple items in a single travel. Such items maybe delivered to different locations hence the computation of paths for the robots is significantly more complex than when single items are considered.

(ii) we propose a decentralized coordination method that uses a token-based protocol to allocate the tasks and a token-wait approach to synchronize the execution of the paths by the robots. We analyse the proposed approach in terms of computational complexity and guarantees that method offers with respect to conflicts among the robots. Crucially we prove that our method guarantees conflict-free paths with fewer restrictions on the map of the working area in comparison to previous approaches;

(iii) we empirically evaluate our approach by comparing the proposed decentralized method with two competitors: a centralized method that evaluates all possible allocations of transportation tasks to robots (named global) and a centralized approach that greedily selects the best allocation of transportation tasks to robots (named greedy). Crucially, the evaluation is performed on the ROS framework by using the full navigation stack that ROS provides. Experiments have been performed on three different operational scenario including the real plants of a prototype smart factory. Results show that the proposed decentralized method provides good performances (in terms of time to complete a set of tasks and distance travelled by the robots) being superior to greedy and comparable to the global approach, while requiring significantly less computational effort.

The reminder of the paper is organized as follows: Section 2 describes the background and related work discussing the MAPD problem as well as the most related approaches for Multi-Agent Path Finding and Multi-Robot task Allocation; Section 3 describes the single-item MAPD problem and introduces the multi-item variant. Section 4 describes the proposed decentralized algorithm to solve the multi-item variant of MAPD and reports the theoretical analysis of the proposed approach in terms of complexity and guarantees. Section 5 describes the empirical evaluation of the proposed approach presenting the empirical methodology and discussing the obtained results. Finally, Section 6 concludes the paper.

2. Background and related work

The MAPD problem requires agents to perform a series of tasks, where for each task an agent must travel to a pick-up location and then to a delivery location [3]. Hence, the MAPD problem is deeply related to the Multi-Agent Path Finding (MAPF) problem, which requires multiple agents to plan conflict-free paths from their positions to their goals, as well as to the Multi-Robot Task Allocation (MRTA) problem, where a set of tasks must be allocated to a set of robots.[2] Both problems have been extensively studied in the literature and there are several techniques that can be used to address them. In the next sections we discuss the techniques that are related to our work the most.

2.1. Multi Agent Path Finding

The MAPF literature offers a wide variety of techniques that span from centralized methods [8–13] to distributed approaches [3,14–17].

Grid-like environments have been used to represent the MAPF problem [12], as they can model different logistic scenarios, including shipping container handling and order fulfilment. However, following recent literature on MAPD [3,4,18], we prefer to use a graph based approach as it is more general and can be specialized to represent grid-like environments.

One of the first approach to MAPF is Cooperative A* (CA*) proposed by Silver in [13]. In CA* each agent plans its path sequentially taking into account the paths of the previous agents. The work proposes two variants of this scheme: HCA* that consider specific heuristics to boost performances, and WHCA* where each agent plans only for a fixed number of steps into the future and re-plan at regular intervals. While our approach to route planning (see Section 4.4) shares some similarities with the basic concepts used by CA*, the work described in [13] is a centralized approach and it is not based on the concept of token-passing.

Distributed approaches to path planning have been proposed for various types of operational scenarios. For example, Gilboa and colleagues in [17] propose a distributed algorithm, for a setting where the model of the environment (a graph in this case) is initially unknown and the agents must physically move to discover new vertices. In this context, agents have a limited communication range and different schemes are proposed to divide the agents in groups that must stay in close proximity. Bhattacharya and Kumar in [19] propose a distributed optimization algorithm for multi-robot path planning to find paths in a 2D continuous environment with rendezvous constraints . In this work, each robot start with an initial plan (a list of coordinates towards its goal) and the algorithm keeps increasing the penalty weights in order to find variations on the path that optimize the constraints. While these approaches propose very interesting solutions they address different problem settings with respect to the logistic scenarios that we focus on.

In contrast, Chouhan and Nyogi in [15] propose DMAPP a token-passing algorithm where each agent plans an individual path using the FastForward planner [20] then each agent exchanges its plan with the others according to a priority order to resolve conflicts between the individual plans. When a lower priority agent receives a message from a higher priority agent the lower priority agent tries to change its plan to avoid conflicts with plans of all higher priority agents. This approach is incomplete because it evaluates a single priority order.

In general, for a team of n robots, $n!$ priority orders should be evaluated, which is computationally expensive. To address this issue, DiMPP [21] extends DMAPP showing that is sufficient to evaluate only n orders, which are variations of the same token cycle where only the cycle initiator changes.

These two methods use a message passing procedure which shares many similarities with our approach, however they solve the MAPF[3] problem in the standard setting where each robot has a single destination that is an input for the problem. In contrast, our approach considers the possibility of having several destinations for each robot and, most important, it addresses also the task assignment problem, i.e., the goals for each robot are not an input for the approach but are computed by the approach starting from the transportation tasks. This is important, because while we build the multi-agent paths considering a single order we can

[2] While usually the term *agent* is more general than *robot* in this work we will use the two words interchangeably adopting the term that is most frequently used in the literature depending on the context.

[3] They name the MAPF problem as Multi-Agent Path Planning (MAPP) hence the name of the approaches.

guarantee that the output paths are conflict-free by changing the tasks allocated to the robots (see Section 4 for more details).

All approaches discussed so far are usually termed *decoupled*, i.e., the paths for the agents are not planned in the joint action space of all the agents. Rather, each individual agent plans for its own path trying to avoiding conflicts with the other agents. Coupled approaches are interesting as they can directly plan conflict-free paths however they are typically computationally intensive and not tractable for most application scenarios. Pianpak and colleagues propose in [16] a coupled decentralized algorithm. To make the approach tractable this algorithm subdivides the search space between multiple agents *spatially*, with each agent in charge of finding MAPF paths across an assigned subgraph. In each round, the agents find an agreement on which robots may cross the boundary from a subgraph to another. Then, each agents finds the paths in its subgraph to move robots across using Answer Set Programming (ASP). While this is an interesting and promising solution scheme, our approach addresses the MAPD problem that involves also the task allocation process and not only path finding. Hence, we prefer to follow the scheme adopted by previous approaches on MAPD [3,4,18] that are decoupled and use graph search techniques to find paths for the robots.

2.2. Multi Robot Task Allocation

As mentioned above MAPD is deeply related to the Multi-Robot Task Allocation (MRTA) problem.

The MRTA problem can be roughly stated as the problem of deciding which robot performs which task(s). This decision process has a great impact on the performance of the overall system and therefore has been widely studied [22,23] and it is closely related to logistic scenarios [1]. MRTA approaches span a wide variety of methodologies including Distributed Constraint Optimization Problem (DCOP) [2], probabilistic methods [24], market-based approaches [25] and auctions [26,27], as well as Token Passing techniques [3,4,28].

Token passing approaches for task assignment are the most closely related to our work. In particular, Scerri and colleagues in [28] propose a task assignment approach where each task is represented by one *token* and tokens are exchanged among the robots. When a robot receives a token it decides whether to execute the associated task or to send the token to some other team mates. The decision on whether to execute a task is a key point and it is taken based on local information that are available to the single robot (e.g., constraints on its own resources, capability to perform the task, other tasks that the robot maybe assigned to) and on a broad knowledge on the state of the system that is transmitted through the tokens (e.g., how many robots refused the token). To guide the system towards good solutions authors propose the use of a threshold on the ability to perform the task: a robots accepts a task only if its ability for the task is higher than the threshold. The main benefit for this token passing scheme is the extremely low computation and communication requirements that make the approach very well suited for large scale applications. While our approach to task allocation is also based on the concept of a token that is exchanged among the robots our focus is not so much on the possibility of allocating tasks in large scale scenario and to achieve high quality task allocation. Rather, similar to [3,4] we use the concept of a token as a way to pass information in a synchronized way through the robot team. In this perspective, we do not create a token for each transportation task but we create one token that represents all tasks and tasks that are allocated are removed from the token. Also the token is a way to communicate the current paths of the robots to the rest of the team so to avoid conflicts. The approaches presented in [3,4] will be discussed in details in Section 2.3.

2.3. Multi-Agent Pickup and Delivery

As mentioned before, MAPD [3] is deeply related to MAPF and MRTA, in fact it can be considered as a variant of MAPF where agents are assigned to delivery tasks and must visit the pickup and delivery location consecutively.

Token-passing approaches are particularly related to the MAPD problem because this type of coordination methods have been successfully applied to MAPD by a series of recent works [3,4]. These works propose the use of a token that maintains the paths already planned by the agents and the tasks that must be assigned. In more detail, Ma and colleagues in [3] propose two algorithms, TP (Token-Passing) and TPTS (Token-Passing with Task Swaps). In TP the token is passed from one agent to another and the agent that owns the token selects the next task to perform avoiding conflicts with the paths already stored in the token. TPTS extends this approach by extending the task set inside the token with the tasks that other robots have assigned but for which they have not yet reached the pickup location. The TP approach has been extended in [4] by taking into account the kinematic constraints of a real robot in order to calculate continuous movements instead of discrete vertices on a graph.

Ma and colleagues show that for well-formed MAPD instances [3] their approach is guaranteed to find conflict-free paths. Well-formed MAPD instances and based on the concept of well-formed infrastructure [5] and must respect a set of constraints. The environment is defined as a planar graph, where each vertex represents a specific location of the environment. The vertices of the graph that represent relevant locations of the environment are called *endpoints* and are divided in *home* and *task* endpoints. Home endpoints are the locations from where robots start and where they go after completing all their tasks. Task endpoints are locations where objects are either picked up or delivered.

A well-formed MAPD instance must respect three constraints: (i) the number of tasks must be finite, (ii) the number of home endpoints must not be lower than the number of robots and (iii) for any two endpoints there must be a path between them that does not traverse another endpoint. We take inspiration from this important line of work because we also use a token-based algorithm to generate conflict-free paths for MAPD.

However there are several elements that differentiate our work with respect to these previous approaches. A first important difference is the fact that we significantly relax the assumptions on the map required by our algorithm to provide conflict-free paths, in fact our MAPD instances do contain non-well formed infrastructures (see Section 4.5). A key element to achieve this is that we do not require every agent to perform a task, rather in our approach one agent could perform more tasks than another and in pathological situation a single agent could perform all tasks. Our empirical evaluation shows that in practice the load distribution across the agents is good and hence these situations are rare.

Recently Okumura and colleagues addressed both the iterative MAPF (and particularly MAPD) in [18]. They propose the Priority Inheritance with Back Tracking (PIBT) approach which solves the iterative MAPF problem with a decoupled decentralized method. In this approach, agents continuously plan their next movements based on prioritized planning. Path adjustments to avoid deadlocks are solved by using priority inheritance that is executed following a backtracking protocol.

Okumura and colleagues show that their approach guarantees that each agent will reach its destination in finite time if the environment is a graph such that each pair of adjacent vertices is part of a bi-connected component, i.e. adjacent nodes belong to a simple cycle of length 3 or more. This restriction on the environment may not be guaranteed in some operational environments as it is the case for some of our MAPD instances (see Section 4.5).

3. Problem definition

In this section we first report the definition of the single-item MAPD problem and then define the multi-item MAPD problem.

3.1. Single-item MAPD problem

The single-item MAPD problem, defined in [3], consists of a task set $\mathcal{T} = \{tk_1, tk_2, tk_3, \ldots, tk_r\}$, an environment described by an undirected planar connected graph $G = (V, E)$ and by n robots. Each task tk_i represents a task requiring the transportation of one item from a pickup location $s_i \in V$ to a delivery location $g_i \in V$. Each free robot (i.e., a robot that is not currently carrying an item) is assigned to an available task. The robot must execute the task by going to the pickup location, taking the item and then going to the delivery location. A robot can be assigned to another task if it has not reached the pickup location of the current task. After the completion of the task the robot can be assigned to a new task. At any time the robots must avoid collisions with each other, in a setting with discrete timesteps $\mathcal{T}_S = \{t_1, t_2, t_3, \ldots, t_z\}$, this can be enforced by making sure that: (i) two robots cannot occupy the same vertex in the same timestep and (ii) two robots cannot traverse the same edge in the same timestep. The instance of the problem is solved when all the tasks have been executed without any collision.

3.2. Multi-item MAPD problem

The multi-item MAPD problem differs from the single-item version [3] in the definition of the task set and the characteristics of the robots.

In the multi-item MAPD problem there is a multi-item task set $\mathcal{P} = \{p_1, p_2, p_3, \ldots, p_q\}$. The multi-item task set $\mathcal{P}$ is generated from the single-item task set $\mathcal{T}$ by a task aggregation algorithm, described in Section 4.2. Each task $p_i \in \mathcal{P}$ represents the request to pick up a set of objects from a pickup location $s_i \in V$ and deliver them to m_i delivery locations $\{g_i^1, g_i^2, g_i^3, \ldots, g_i^{m_i}\} \in V$. These destinations may differ between each object or may be the same for all of them.

The objective is to assign each task to a robot and a robot can have multiple tasks assigned. Then for each robot a path that completes all the assigned tasks must be found. A task p_i is considered completed when the assigned robot has a path to reach the pickup location s_i and subsequently reach all the m_i delivery locations. Each robot must execute tasks in the order on which they have been assigned and only one task at a time can be active.

Similar to the single-item version, planning takes place in a setting with discrete timesteps $\mathcal{T}_S = \{t_1, t_2, t_3, \ldots, t_z\}$. At each timestep each robot occupies a vertex of the graph G and there are two types of conflict that may occur between the robots. The first is a vertex conflict, where two robots occupy the same vertex at the same time. The second is an edge conflict, where two robots to go through the same edge in opposing directions. Robots must find plans that are conflict-free. The instance of the problem is solved when all the tasks have been planned and the generated plans are conflict-free.

Differently than the standard MAPD problem, a task can require a robot to transport more than one item. However, a robot has a limit on how many objects it can transport, depending on the weight or the size of the objects. For this reason we pose a limit on the transportation capacity of a robot. Specifically, we defined a capacity value C, equal for every robot.[4] We then consider a demand for every item and we assume that a robot can execute a multi-item task only if the sum of the item demand is less or equal its capacity.

[4] Having homogeneous capacity for the robots is a reasonable assumption for standard logistic scenario. It would be possible to extend the approach to different capacities for different robots, but this would require a more sophisticated approach to perform the task partitioning phase which is not the main focus of this paper.

4. Method

In this section we describe our proposed solution to the multi-item MAPD problem. Specifically we first provide an overview of the approach and then detail the most important ingredients for our method: task aggregation, task assignment and route planning. Finally we provide an analysis of the approach in terms of complexity and guarantees.

4.1. Overview

To deal with the multi-item MAPD problem we propose a distributed algorithm, called *iterative*, based on a Token Passing technique were a token $\mathscr{T}$ contains the paths of every robots and the remaining tasks. The token $\mathscr{T}$ is shared among the robots via message passing. The *iterative* algorithm takes as input a multi-item task set which is computed by a dedicated module called *task planner*. The *task planner*, takes as input the single item transportation tasks and divides them in batches called windows. The single item tasks that belong to a windows are aggregated into a set of *multi-item* transportation tasks which is then given as input to the iterative algorithm. The task planner is realized as a centralized algorithm that operates on each window separately. Specifically, for each window the task planner evaluates all partitions that meet the capacity constraints of the robots, choosing the one that minimizes a given cost function (see Section 4.2 for more details). When the multi-item tasks for a given window has been generated they will be processed by the task allocation algorithm and robots will then execute the computed paths. While the task allocation and execution phase is running the task planner can process another window to generate a new multi-item task set. The task allocation algorithm can start allocating these new task set while the robots are executing their current tasks, however there is not task preemption, i.e., tasks generated from a previous window will always be executed before the tasks generated from the next window. The task planner is completely decoupled from the iterative algorithm and it is used only as a way to generate the Multi-Item MAPD instance from a single item task set. While more advanced approaches can be used to realize a more efficient task partitioning phase, our focus in this work is on the iterative algorithm. Hence we decided to choose a simple approach for the task aggregation phase.

When the multi-item transportation tasks have been created the task planer generates a token that contains all the tasks and sends the token to one of the robots. The token is then passed from one robot to another to distribute the tasks and plan conflict free paths, hence completing the multi-item tasks.

When the paths are executed, the token keeps cycling through the robots to synchronize their movements. After a robot reaches the next location in its path it waits for the token and it inserts the information about its progress inside the token, then the robot waits for the other team member to do the same before going towards the next location. This is important because before executing the paths it is very difficult to have a clear estimation of the time required by a robot to execute a portion of the path. Hence, if robots are not synchronized conflicts may happen even if the computed paths considering the discrete timesteps $\mathcal{T}_S$ are conflict free.

We also consider two centralized algorithms called *global* and *greedy* as benchmarks for our approach. They implement different task assignment strategies (see Section 4.3). For the centralized algorithms the token is used only for synchronization purposes.

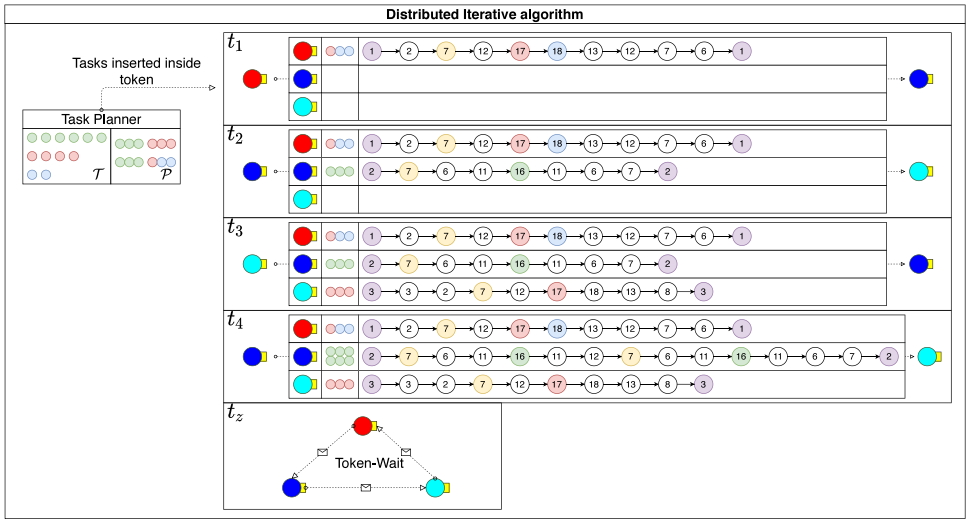


Fig. 1. Execution example for the iterative algorithms. The example is related to the grid map (See Fig. 3). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

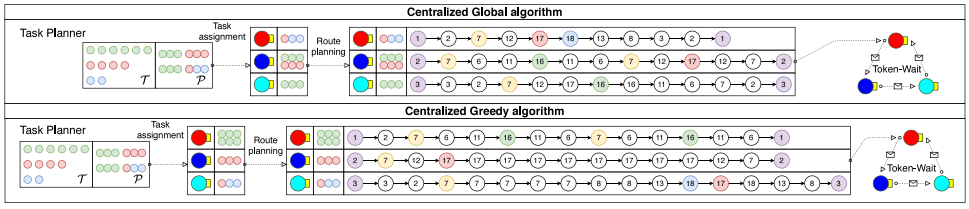


Fig. 2. Execution examples for the two centralized algorithms. The examples are related to the grid map (See Fig. 3).

For ease of explanation, all the algorithms will be described considering only one window. Figs. 1 and 2 show a visual representation of an execution example on the grid map (see Fig. 3) for the three approaches. These will be used in what follows as a running example to explain the main ingredients of our approach.

4.2. Task aggregation

This is the first step of the three algorithms. In this phase the single item task set $\mathcal{T} = \{tk_1, tk_2, tk_3, \ldots, tk_r\}$ is aggregated to form a multi-item task set $\mathcal{P}$. This is done by the task planner and then the multi-item task set $\mathcal{P}$ is passed to the task allocation algorithm. For the *iterative* algorithm that is decentralized, the $\mathcal{P}$ is inserted into the token and sent to the robots.

Let us consider a partition $\mathscr{P}_i = \{\mathcal{S}_i^1, \mathcal{S}_i^2, \mathcal{S}_i^3, \ldots, \mathcal{S}_i^{v_i}\}$ of a generic set X as a set of non-empty subsets of X such that X is a disjoint union of the subsets ($\bigcup_{\mathcal{S}_i^j \in \mathscr{P}_i} \mathcal{S}_i^j = X$ and $\forall \mathcal{S}_i^k, \mathcal{S}_i^l \in \mathscr{P}_i \;\; \mathcal{S}_i^k \neq \mathcal{S}_i^l \Rightarrow \mathcal{S}_i^k \bigcap \mathcal{S}_i^l = \emptyset$).

The multi-item task set $\mathcal{P}$ is a partition of $\mathcal{T}$, where each multi-item task will have items with a total demand that is less or equal to the capacity of the robots. This is done by evaluating all the partitions $\mathscr{P}_1, \mathscr{P}_2, \ldots$ of the task set $\mathcal{T}$. Each partition $\mathscr{P}_i$ represents a possible aggregation of the tasks and each subset $\mathcal{S}_i^j \in \mathscr{P}_i$ contains the tasks that will form a multi-item task. The partitions that do not respect the constraint on the capacity are ignored.

A multi-item task set $\mathcal{P}_i$ is generated from the partition $\mathscr{P}_i$. For each subset $\mathcal{S}_i^j$ a multi-item task p_j is created, with each destination g_j^k being equal to the destination g_k of the task $tk_k \in \mathcal{S}_i^j$. This is shown in Fig. 1: each coloured dot in $\mathcal{T}$ is a single-item task and the colour of the dot represents the destination of the corresponding item (e.g., the green dots have vertex 16 as destination, the red dots have vertex 17 and blue light dots have vertex 18). Each multi-item task in $\mathcal{P}$ is composed of one or more items from the original task. To choose the best partition an evaluation metric on the cost of a multi-item task p_j is defined:

$$V(p_j) = \frac{\text{path}(p_j)}{\text{demand}(p_j)} \tag{1}$$

where path(p_j) is the length of the shortest path completing the multi-item task p_j (without considering other robots) and demand(p_j) is the total demand of the task.

The cost of a multi-item taskset $\mathcal{P}_i$ is:

$$V(\mathcal{P}_i) = \sum_{p_j \in \mathcal{P}_i} V(p_j) \tag{2}$$

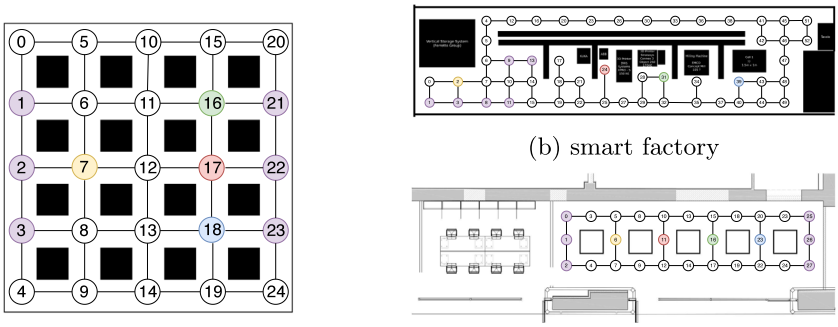
The final task set $\mathcal{P}$ will be the one with the lowest cost. This phase is executed once for each window of tasks.

4.3. Task assignment

The multi-item tasks generated by the task planner are assigned to the robots by using one of the three mentioned algorithms: *iterative*, *global* or *greedy*.

4.3.1. Iterative algorithm

Algorithm 1 reports a pseudo-code description of the iterative algorithm. The algorithm takes as input a multi-item task set $\mathcal{P}$ which has been generated by the task planner considering a given window of single item tasks. The task planner generates a token that contains the task set $\mathcal{P}$ and sends the token to the robot that has the path with the minimum number of hops. This can be determined by looking at the paths which are stored inside the token. This is an heuristic that aims at minimizing the makespan (see Section 4.4 for more details) and at balancing the load in



(a) Grid

(b) smart factory

(c) Model5

Fig. 3. The 3 maps used for the empirical evaluation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

terms of tasks between robots. We also arbitrarily define a total order o among the robots (e.g., based on their ids) which is used to break ties when more robots have paths with the same number of hops. This arbitrary order is also used to select which robot to exclude from the task assignment process when a valid path cannot be found (see below for more details). The robot that owns the token selects a task, called p_i, from the token and derives from the task the list of endpoints (essentially, the pickup location s_i followed by the delivery locations $g_i^1, g_i^2, \dots$) that must be reached to complete the task. The list is completed by adding at the end the home endpoint of the robot. This result in a list of vertices that the robot must reach in order to complete the assigned task. This list is used as an input for the path planning phase (see Section 4.4) that computes a path connecting the endpoints and avoiding conflicts with the other robots. If a valid path is found, it is stored in the token and the token is sent to the robot that now holds the shortest path. Otherwise, if such path cannot be found, then the task is re-inserted in the token and the token is sent to the next robot until a robot that can find a valid path is found. If there is no robot that can execute the task, then all the tasks in $\mathcal{P}$ are re-inserted in the token and the first robot according to the total order o is excluded from the allocation process. The algorithm is then restarted with the reduced set of robots.

If this procedure cannot produce a conflict-free solution with two or more robots, we assign all the tasks to a single robot.[5] To do this, the token is sent to the first robot of the team (according to the order o) and all the tasks are assigned to this robot. If such robot cannot find a valid path to execute all tasks then the robot re-inserts all the tasks inside the token and sends the token to the next robot. The procedure is iterated until a robot in the team can execute all the tasks. Section 4.5 shows that a valid assignment can always be found if the operational scenario meets a constraint which is weaker than the ones required by previous work (see Section 2).

Fig. 1 gives a visual representation of an execution of the iterative algorithm in the grid map (see Fig. 3). The box called *Task Planner* shows the single item task set $\mathcal{T}$ and the resulting $\mathcal{P}$. Once $\mathcal{P}$ has been generated the Task Planner creates the token and sends it to the red robot. The arbitrary order o in this example is $\{red, blue, cyan\}$. Boxes t_1 to t_4 show the task assignment and path planning phases. In box t_1 the red robot chooses the first multi-item task and plans a path that goes to the pick-up location (yellow vertex) and then to the delivery locations (red and blue vertices). Then the red robot sends the token to the blue robot. In box t_2 the blue robot chooses a multi-item task and calculates a conflict-free path. This path considers the one already planned by the red robot and does not generate any conflict with such path. Then the blue robot passes the token to the cyan robot. The cyan robot chooses in box t_3 one of the remaining tasks and plans its path, making sure that conflicts are avoided with the paths of the other two robots. Notice that this robot waits a turn in vertex 3, before going to the pick-up location, to make sure that the paths are conflict free. Finally the token is sent back to the blue robot (i.e., the one that has the shortest path) that selects the last multi-item task. This time the blue robot changes parts of its path to complete the second task right after the first one without going to its home endpoint, as shown in box t_4. After reaching the green delivery location the robot goes to the pick-up location and then again to the delivery location. Finally, box t_z represents the synchronized execution of the paths achieved by using the token as a synchronization mechanism.

4.3.2. Global centralized algorithm

This algorithm aims at computing the conflict-free paths that minimize the maximum number of hops for all the robots with a brute force approach. This is done by evaluating all the partitions of the multi-item task set $\mathcal{P}$ with a size equal to the number of robots n. Each of these partitions $\mathscr{A}_i = \{\mathcal{A}_i^1, \mathcal{A}_i^2, \dots, \mathcal{A}_i^n\}$[6] represents a possible way to divide the tasks between the robots because each subset $\mathcal{A}_i^j$ of the partition contains the tasks that must be executed by one and only one robot. This does not tell us which robot will execute a particular subset because there are $n!$ possible assignments, where n is the number of robots. To account for all these assignments we must consider all the permutations of $\mathscr{A}_i$. The algorithm evaluates each partition and for each partition evaluates each possible permutation. The best partition's permutation is the one that gives the paths with the least number of hops.

The paths are calculated for each robot in sequential order. The route planning algorithm is executed for each robot and takes into account the paths of the previous robots. If the route for a robot cannot be found then the algorithm moves to the next permutation and when all the permutations have been evaluated it goes to the next partition. If a valid path cannot be found even after evaluating all the permutations of all the partitions then a robot is removed and the process is restarted. The computational effort required to execute this algorithm is strongly dependent on the

[5] The described procedure removes one robot at a time following an arbitrary order, hence it is not complete: it may exist a conflict free solution with two or more robots that our procedure did not check. However, to guarantee completeness we should check all the possible $n!$ orders for the robots hence incurring a significant computational overhead. Considering that in our experiments the case where a single robot executes all tasks never appeared, we prefer to consider only the order o in this procedure.

[6] Note that these are the partitions of the $\mathcal{P}$ that was selected by the task planner and not the partitions of the original task set $\mathcal{T}$

number of multi-item tasks and hence on the number of single-item tasks . As a consequence the window size (i.e., the number of single-item tasks that the method allocates in a batch) must be chosen appropriately in order to obtain acceptable computation times (see Section 5).

Fig. 2 shows a solution found by the global centralized algorithm for the same multi-item task set $\mathcal{P}$ used in the example of the iterative algorithm. It is possible to see that the allocation is different (the blue robot is allocated to two tasks) and as expected the makespan is lower compared to the iterative solution (12 hops against 14). Similar to the iterative algorithm the execution of the paths is synchronized through the use of the token.

While this algorithm is not particularly useful from a practical standpoint (searching all the possible permutations is tractable only for very small instances in terms of number of robots, as shown in Section 4.5), this is a good benchmark for the iterative solution, as this algorithm finds the best possible assignment.

4.3.3. Greedy centralized algorithm

This approach is very similar to the global centralized approach, because all the permutations of all the partitions will be eventually evaluated. However, in this case the search stops whenever the algorithm finds an assignment that can generate conflict-free paths. All the remaining assignments are not evaluated.

This algorithm is again used as a benchmark for the iterative algorithms, as it offers an approach that, on average, requires much less computation than the global approach. The approach does not guarantee optimality (in terms of minimizing the number of hops) but it does guarantee completeness (it will always find a conflict free allocation if one exists). The computational complexity of the worst case for this algorithm is the same as the global approach (see Section 4.5 for more details).

Fig. 2 shows a solution found by the greedy centralized algorithm for the same multi-item task set $\mathcal{P}$ used in the example of the iterative and global centralized algorithm. Also in this case the allocation is different and the makespan is higher compared both to the iterative and centralized (16 hops against 12 and 14). This is just a single example of execution but this trend is confirmed by the empirical analysis reported in Section 5.

Algorithm 1: Iterative Distributed Algorithm

inputs : Multi-item task set $\mathcal{P}$, team of robots $\mathcal{R}$ of size n
// Each robot in parallel
$\mathcal{W}_i = \emptyset$ // The list of endpoints of robot r_i
// Allocation phase: the token is sent to the first robot by the task planner and contains the task set $\mathcal{P}$
while *the token* $\mathcal{T}$ *contains at least a task* **do**
  // the robot r_i has the token
  task $p \leftarrow \mathcal{T}$.pop_task()
  $\mathcal{W}_i$.add($p.s$) // $p.s$ is the pickup location
  foreach $g^j \in p$ **do**
    $\mathcal{W}_i$.add($p.g^j$) // $p.g^j$ is the j-th delivery location
  end
  $\mathcal{W}_i$.add(initial_vertex) // Home vertex
  op_i = spacetime_astar($\mathcal{W}_i$, G, $\{op_1, \ldots, op_{i-1}\}$)
  if op_i *is a conflict-free path* **then**
    $\mathcal{T}$.add_path(op_i)
    // $\mathcal{T}$ is sent to robot r_j, where $|op_j| \leq |op_k|\ \forall k < n$
  else
    $\mathcal{T}$.push_task(p) // r_i restores the task in the token
    // $\mathcal{T}$ is sent to robot r_j, where $j \neq i \wedge |op_j| \leq |op_k|\ \forall k < n, k \neq i$
  end
end

4.4. Route planning

Our approach for route panning is based on a variation of the A* algorithm that finds a path going through a list of endpoints $\mathcal{W} = \{w_1, w_2, w_3, \ldots\} \in V$ and avoids conflicts with other robots. The algorithm explores a state space where each state is defined by the location on the graph $G = (V, E)$, the current timestep and the next endpoint to visit. A state $st = (u, t, w)$ is linked to another state $st' = (v, t + 1, w')$ if $(u, v) \in E$ and location v is reachable in the next timestep without resulting in a conflict. Notice that $\forall u \in V \ \ (u, u) \in E$, hence a robot is also allowed to wait in its place. It is important to note that after a robot completes its path it will stay on the last visited vertex unless it receives a new path to execute. As mentioned before, for each multi-item task p_i we have an associated list of endpoints $[s_i, g_i^1, g_i^2, \ldots, g_i^{m_i}]$. The list of endpoints for a series of task is defined as the concatenation of each task endpoint list. As mentioned before, we add at the end of the list the home endpoint of the robot. This is important because we want the robot to complete its path in a location where it cannot obstruct the paths of the other robots. As show in Section 4.5, the home endpoint of the robot is always a location that satisfy this constraint. This also gives the guarantee that the path for new tasks can always be planned (explained in detail in Section 4.5).

The path planning algorithm for a given robot i (procedure spacetime_astar in Algorithm 1) takes as input a list of endpoints $\mathcal{W}_i$, the graph G, and the paths of the other robots $\{op_1, \ldots, op_{i-1}\}$ and calculates a conflict-free path for the robot if one exists or returns a failure. The algorithms return the shortest path for a robot in terms of the least number of hops. We made this choice because our main aim is to minimize the makespan (as said in the introduction) and given our synchronization approach to the movements of the robots this is equivalent to minimizing the number of hops. An admissible heuristic has been defined to have the guarantee that the paths returned by the path planning are the shortest possible. Because the aim is to minimize the number of hops, it makes sense to use as heuristic the minimum number of hops necessary to complete a path between any two vertices, regardless of other robots. More formally, given two vertices $u, v \in V$ the heuristic function $h(u, v)$ return the minimum number of hops required to reach vertex v from vertex u. This heuristic function can easily be precomputed by using any all-pairs shortest path algorithm, in this case the Floyd–Warshall algorithm has been used.

4.5. Analysis of the approach

A key point of our method is that we can guarantee conflict-free paths even if the MAPD instance is not well-formed [3], in particular we relax the constraint that a path between any two endpoints must not traverse another endpoint. This constraint can be restrictive for several operational scenarios and in particular some of the scenarios used here do not meet such constraint. This can be seen in Fig. 3, that shows the graph structure of all the maps together with the endpoints. Endpoints are represented as coloured vertices, where the purple vertices are home endpoints for the robots, the yellow vertex are the pickup locations and the others are the delivery locations. Vertices 2 and 22 in the grid map do not meet the constraint mentioned above, this is also true for vertices 1 and 26 in the model5 map. Finally, in the smart factory map it is impossible to reach vertices 1 and 3 without going through vertex 2 or vertex 8.

Moreover, our scenarios does not satisfy the condition that all pairs of adjacent nodes belong to a simple cycle of length 3 or more, which is required by PIBT [18] to guarantee that every agent always reaches its destination within finite time. For example, vertices 24 and 25 in the smart factory map do not meet such constraint.

To address this issues, we relax the constraints on the MAPD instances and we can guarantee that valid paths can be found by posing the following constraint:

Constraint 4.1. *There is at least one home endpoint from which all the task endpoints can be reached without traversing other home endpoints.*

This poses a much weaker constraint with respect to the third constraint for well-formed infrastructure and in fact all the MAPD scenarios that we use in our empirical evaluation do meet this constraint but as mentioned above some of these are not well-formed infrastructures.

For what concerns the constraint required by PIBT [18], we observe that such constraint is related to the topology of the graph and not to the MAPD scenario (i.e., it is not related to the position of home endpoints). Hence, if the topology of the graph meets the required constraints the endpoints can be located without any further restriction. However, if the topology of the graph does not meet the constraint it is not possible to find a configuration of endpoints that respects the constraint. Now, in most cases changing the topology of the graph is harder than changing the location of endpoints, because the former requires a physical change in the layout of the warehouse while the latter requires to move, for example, the starting position of the robots. Hence while the two constraints are applied to different elements we believe that Constraint 4.1 is easier to respect in practical scenarios.

Assuming that the above constraint is met we can show that all the algorithms we consider here (global, greedy and iterative) will always find a valid solution for the MAPD. This is detailed in the following theorems.

Theorem 4.2. *The global and greedy algorithms always find an allocation of tasks that gives conflict-free paths that solves the multi-item MAPD problem instance*

Proof. Because the global and greedy algorithms tries all the possible allocations of tasks, they will also evaluate the allocations in which a single robot takes all the tasks and in particular they will evaluate the allocation in which such robot will start from the endpoint mentioned in Constraint 4.1. Because all the other robots stay still, such allocation enables the robot to reach all the pickup and delivery vertices without conflicts. □

Theorem 4.3. *The iterative algorithm always finds an allocation of tasks that gives conflict-free paths that solves the multi-item MAPD problem instance*

Proof. In case the iterative algorithm is forced to use a single robot, it searches for the one that can execute all the tasks by itself. Because of Constraint 4.1 such robot exists and a valid plan is found. □

Even though this guarantee can be given in the worst case that a single robot takes all the tasks, our results show that for all the tested MAPD instances all robots execute at least one multi-item task. This is also apparent from the balancing results among the robots which we report in Tables 2 and 4.

For what concerns the computational complexity of our algorithms, we observe that the task partitioning phase requires the enumeration of the partitions of a set, the number of partition of a set of $n+1$ elements is the Bell number B_{n+1} that can be found with this recursive relationship:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

As for the route planning algorithm, since this is a variant of the A* algorithm, the complexity in the worst case can be defined (assuming an adjacency list implementation on a connected graph $G = (V, E)$ and a limit of T hops) as

$$O(|E| \log(|V|T|\mathcal{W}_{max}|))$$

where, $|\mathcal{W}_{max}|$ represents the maximum number of endpoints that can be given in input to the path planning algorithm. This limit depends on how many delivery endpoints are present in the map and any task that visits all the delivery endpoints will give this number of endpoints in input to the route planning algorithm.

As for the iterative algorithm, the task aggregation phase must be executed for each window and for each window the aggregation might return a different number of multi-item tasks. Let us indicate with d the maximum number of multi-item tasks generated from a window of r single-item tasks by the aggregation process. For each task the route planning algorithm must be executed.

$$O(wB_r + wd|E| \log(|V|T|\mathcal{W}_{max}|))$$

In this formula, and the others below, the first part represents the task aggregation phase, while the second is the complexity of the task allocation and path planning phase combined. Where, w is the number of windows and, as mentioned above, r is the number of single-item tasks ingested by the task aggregation algorithm.

For the two offline centralized algorithms we must take into account the enumeration of all the permutations of each partition. The Johnson–Trotter algorithm generates all the possible permutations with a complexity of $O(n!)$, where n is the size of the set. In this case, the route planning algorithm is executed for each permutation of each partition are considered, hence resulting in a complexity of:

$$O(wB_r + wB_d!|E| \log(|V|T|\mathcal{W}_{max}|))$$

Where, as mentioned before, w is the number of windows, r is the number of single-item tasks and d is the maximum number of multi-item tasks. As mentioned in Section 4.3.3, the greedy algorithms has the same complexity in the worst case, however in our empirical evaluation it terminates long before global because it takes the first valid solution.

5. Empirical evaluation

To test our approach we implemented the three algorithms described in Section 4 using the ROS framework. In particular, we employ a modified version of the patrolling_sim [29] package[7] and we have created three maps and different task sets to test our algorithms with multiple configurations. The ROS framework provides a navigation stack, i.e., a set of modules that implement key functionalities for navigation of mobile platforms such as localization, path planning and obstacle avoidance. Using the navigation stack enables us to do simulations in a realistic scenario, where each robot senses the environment through laser scanners and localizes itself on the map with a particle filter system (i.e., using the AMCL module of the ROS navigation stack). Moreover, the robots are guided by the move_base module that provides path planning and obstacle avoidance. It is important to note that move_base cannot safely drive the robots in complex situations and will fail most of the time in presence of cluttered environments and moving obstacles (e.g., other robots). Hence we cannot simply give to the robot a series of endpoints (e.g., pickup location and destinations) but must coordinate their movements to guarantee conflict-free paths. On the other hand, move_base is very useful to avoid collisions in face of possible unexpected situations. Such unexpected situations can be caused, for example, from localization errors, which can occur frequently when dynamic obstacles are present in the working environment.

[7] A github repository containing the code for our approach can be found at this link https://github.com/antonellocontini/LogisticAgent.

Table 1
Summary of the three algorithms' performance in the single window scenario, analysis performed on 10 task sets of 12 single-item tasks.

Model5	2	4	6	stats
Single window global	**319.70 ± 64.15** **160.46 ± 28.78**	***213.90 ± 27.76*** **178.85 ± 26.26**	**210.90 ± 33.28** ***202.90 ± 26.23***	Time [s] $\sum$ Dist [m]
Single window iterative	324.00 ± 47.71 163.62 ± 25.95	234.60 ± 29.06 183.59 ± 34.08	221.30 ± 27.73 218.70 ± 37.24	Time [s] $\sum$ Dist [m]
Single window greedy	433.70 ± 183.82 162.57 ± 28.92	*372.10 ± 77.43* 188.82 ± 28.75	*308.10 ± 74.68* 218.20 ± 36.26	Time [s] $\sum$ Dist [m]
Grid	2	4	6	stats
Single window global	**195.40 ± 24.03** ***145.92 ± 18.58***	***134.50 ± 13.76*** ***178.45 ± 22.39***	***114.90 ± 8.13*** ***206.72 ± 24.70***	Time [s] $\sum$ Dist [m]
Single window iterative	200.10 ± 21.56 149.87 ± 16.65	153.10 ± 22.49 188.78 ± 23.22	135.40 ± 10.58 221.31 ± 24.10	Time [s] $\sum$ Dist [m]
Single window greedy	*308.00 ± 47.80* *158.38 ± 21.33*	*247.50 ± 44.99* *206.42 ± 23.17*	*185.00 ± 46.72* 227.09 ± 33.49	Time [s] $\sum$ Dist [m]
Smart-factory	2	4	6	stats
Single window global	**744.50 ± 145.46** **251.32 ± 46.68**	**505.90 ± 113.42** 263.70 ± 47.83	***448.00 ± 119.15*** **282.82 ± 55.84**	Time [s] $\sum$ Dist [m]
Single window iterative	791.70 ± 110.33 252.65 ± 46.04	564.10 ± 85.17 **263.30 ± 48.44**	530.60 ± 127.78 286.39 ± 45.73	Time [s] $\sum$ Dist [m]
Single window greedy	*945.00 ± 101.46* 251.45 ± 48.74	*918.70 ± 179.93* 270.95 ± 44.53	*718.70 ± 258.85* 291.75 ± 48.11	Time [s] $\sum$ Dist [m]

Bold = Best result. *Italic* = Rejected paired-sample t-test with a 5% significance level. The t-test is always done comparing the population of the highlighted value with the iterative algorithm's one.

A video showing an execution of the iterative algorithm in simulation is associated with this paper.[8] The video is executed on the smart factory map. Notice that at second 20 the move_base module of the cyan robot performs a recovery procedure to avoid the blue robot that is deemed too close. This introduces a delay in the execution of the path for the cyan robot, however thanks to the token-wait synchronization, the robots are still able to correctly perform the paths without any collision.

A video showing an execution of the iterative algorithm on two RB-KAIROS robots operating in the ICE laboratory is associated with this paper.[9] Each robot is assigned to a pickup and delivery task. Both robots reach the pickup location (seconds 0:20 and 0:38), they then move to the delivery location (0:48 and 1:22) and return back to their home location. The pickup and delivery locations are the same for both robots. From second (0:08 to second 0:20) one of the robots waits for the other to reach the pickup location, while at second 0:58 the same robot waits for the other to clear the corridor before going to the delivery location. The video validates the use of our approach on real platforms in their operative scenario.

5.1. Empirical methodology

Our aim is to test our algorithms on different situations so we generated 3 maps: a grid map, and two maps representing two configurations of the ICE laboratory, called model5 and smart factory. In each map there are 6 starting locations for the robots, 1 pickup location and 3 delivery locations. We tested our system with 27 different configurations, where a configuration is defined by the algorithm, the map and the number of robots (2, 4 or 6). Two different scenarios have been tested. In the first scenario 10 task sets have been defined, each with a single window composed of 12 single-item tasks, each item has a random demand value. For the second scenario the same number of task sets has been used, but each task set has 30 task, while the window size has been reduced in order to have more than one window. For each configuration the algorithms have been run in both scenarios, obtaining a run for each combination of configuration and task set. For each run we record the makespan, the number of tasks completed, the number of transported items and the total distance travelled by each robot.

Table 2
Load balancing of the three algorithms in a single window scenario, analysis performed on 10 task sets of 12 single-item tasks.

Model5	2	4	6	stats
Single window global	0.10 0.28 0.07	1.43 0.44 0.39	0.69 0.49 0.32	σ(Distance [m]) σ(Items) σ(Tasks)
Single window iterative	0.73 1.13 0.35	1.24 0.75 0.30	2.02 0.69 0.38	σ(Distance [m]) σ(Items) σ(Tasks)
Single window greedy	34.64 3.54 2.19	22.12 2.59 1.55	10.12 1.34 0.81	σ(Distance [m]) σ(Items) σ(Tasks)
Grid	2	4	6	stats
Single window global	0.86 0.00 0.07	1.63 0.61 0.26	0.76 0.47 0.29	σ(Distance [m]) σ(Items) σ(Tasks)
Single window iterative	3.22 1.13 0.35	1.03 0.75 0.30	1.75 0.69 0.38	σ(Distance [m]) σ(Items) σ(Tasks)
Single window greedy	66.85 6.65 4.31	30.51 3.40 2.05	12.30 1.38 0.86	σ(Distance [m]) σ(Items) σ(Tasks)
Smart-factory	2	4	6	stats
Single window global	0.61 0.42 0.07	2.41 0.41 0.24	3.41 0.38 0.29	σ(Distance [m]) σ(Items) σ(Tasks)
Single window iterative	2.93 1.13 0.35	5.32 0.75 0.30	7.96 0.69 0.38	σ(Distance [m]) σ(Items) σ(Tasks)
Single window greedy	69.54 4.38 2.47	54.90 3.20 1.92	22.17 1.38 0.86	σ(Distance [m]) σ(Items) σ(Tasks)

[8] The video is available at this link: http://profs.sci.univr.it/~farinelli/videos/MAPD-sim.mp4.

[9] The video is available at this link: http://profs.sci.univr.it/~farinelli/videos/MAPD-kairos.mp4

Table 3
Summary of the three algorithms' performance in the multiple window scenario, analysis performed on 10 task sets of 30 single-item tasks. Window size is 11 for the iterative algorithm and 4 for the other two.

Model5	2	4	6	stats
Global	*837.10 ± 73.72* *375.38 ± 30.14*	**528.50 ± 36.63** *407.95 ± 35.31*	495.30 ± 42.04 *454.19 ± 36.18*	Time [s] $\sum$ Dist [m]
Iterative	**781.40 ± 78.18** **340.76 ± 35.86**	535.60 ± 55.50 **384.63 ± 38.08**	**485.40 ± 60.47** **430.59 ± 40.41**	Time [s] $\sum$ Dist [m]
Greedy	*1067.80 ± 84.49* *378.30 ± 28.99*	*667.20 ± 54.34* *408.83 ± 35.34*	*687.00 ± 59.71* 415.53 ± 36.32	Time [s] $\sum$ Dist [m]
Grid	2	4	6	stats
Global	*540.50 ± 40.68* *128.80 ± 6.66*	350.50 ± 20.24 *148.84 ± 7.00*	**293.70 ± 16.47** **165.38 ± 6.68**	Time [s] $\sum$ Dist [m]
Iterative	**499.40 ± 36.91** **117.90 ± 9.32**	**340.80 ± 27.74** **139.50 ± 11.96**	295.60 ± 26.73 169.07 ± 16.32	Time [s] $\sum$ Dist [m]
Greedy	*632.60 ± 51.61* *130.36 ± 6.42*	*411.30 ± 10.89* 146.31 ± 8.98	*411.70 ± 17.78* *146.31 ± 8.79*	Time [s] $\sum$ Dist [m]
Smart-factory	2	4	6	stats
Global	*1441.50 ± 110.97* *687.52 ± 50.76*	998.70 ± 130.78 *739.02 ± 61.45*	887.90 ± 56.59 *802.33 ± 67.27*	Time [s] $\sum$ Dist [m]
Iterative	**1356.60 ± 150.74** **631.46 ± 69.09**	**926.50 ± 105.27** **683.18 ± 78.31**	**880.20 ± 84.26** **759.20 ± 77.74**	Time [s] $\sum$ Dist [m]
Greedy	*1592.50 ± 136.23* *690.09 ± 56.67*	*1234.50 ± 93.12* *736.62 ± 52.63*	*1283.30 ± 73.41* *772.30 ± 56.86*	Time [s] $\sum$ Dist [m]

Bold = Best result. *Italic* = Rejected paired-sample t-test with a 5% significance level. The t-test is always done comparing the population of the highlighted value with the decentralized algorithm's one.

Table 4
Load balancing of the three algorithms in a multiple window scenario, analysis performed on 10 task sets of 30 single-item tasks. Window size is 11 for the iterative algorithm and 4 for the other two.

Model5	2	4	6	stats
Global	0.30 1.27 0.99	2.98 1.28 0.90	6.88 1.50 1.02	σ(Distance [m]) σ(Items) σ(Tasks)
Iterative	6.05 0.71 0.64	3.33 1.08 0.55	8.51 1.08 0.78	σ(Distance [m]) σ(Items) σ(Tasks)
Greedy	90.83 7.92 4.95	48.38 4.95 3.04	59.55 5.09 3.46	σ(Distance [m]) σ(Items) σ(Tasks)
Grid	2	4	6	stats
Global	7.98 3.39 1.84	1.74 1.41 0.41	2.30 1.09 0.72	σ(Distance [m]) σ(Items) σ(Tasks)
Iterative	3.85 1.41 1.06	2.01 1.40 0.94	1.56 0.94 0.64	σ(Distance [m]) σ(Items) σ(Tasks)
Greedy	26.97 8.06 5.09	16.25 4.93 3.04	22.77 5.44 3.75	σ(Distance [m]) σ(Items) σ(Tasks)
Smart-factory	2	4	6	stats
Global	2.91 0.99 0.28	5.65 0.72 0.39	2.51 0.62 0.17	σ(Distance [m]) σ(Items) σ(Tasks)
Iterative	0.86 0.00 0.07	7.89 0.85 0.33	3.45 0.22 0.21	σ(Distance [m]) σ(Items) σ(Tasks)
Greedy	73.81 4.95 2.69	100.92 5.01 3.12	122.66 5.36 3.67	σ(Distance [m]) σ(Items) σ(Tasks)

5.1.1. Single window scenario

Table 1 shows the average makespan (expressed in seconds) and the average of the sum of the distance travelled by all the robots (expressed in metres). Results show that the global algorithm offers the best performance in terms of makespan and travel distance (as expected). Results also show that the iterative algorithm does not differ significantly from the global one. In fact, only few configurations show a statistically significant difference according to the t-test (the numbers reported in italics in the table represent cases where the t-test null hypothesis was rejected). However, it is important to note that the global algorithm is extremely intensive from a computational point of view: most of the configurations with 6 robots required hours or even days to give the best solution, while the iterative algorithm finds a comparable solution in seconds. The greedy algorithm exhibits a run time similar to the iterative method but its makespan is often considerably worse. This is because while the iterative algorithm distributes the task in an even way, the greedy algorithm selects the first valid assignment it finds, and in many cases this results in unbalanced allocations where few robots perform most of the tasks, hence the higher makespan. Table 2 shows the load balancing of the three algorithms in terms of standard deviation of distance travelled by each robot, distribution of tasks and items. Even in this aspect iterative and global are comparable, while greedy offers the worst balancing. This confirms the above discussion.

5.1.2. Multiple windows scenario

For the multiple window scenario the window size must be taken in consideration. This influences the performance of the algorithms, in particular for global and greedy, where the computation time grows rapidly with the number of tasks. For the iterative algorithm a window size of 11 has been chosen, while for the other two the window has been set to 4 tasks. The choice of window size is limited by the associated computational effort, where a bigger window will require higher computation and will provide better performance. This is particularly relevant for the centralized algorithms which have a higher computational overhead and hence can be executed only on windows of smaller size. For this reason, we do not evaluate the iterative algorithm with 4 tasks sets, as a key benefit of this approach is the lower computational overhead that allows the execution on larger windows.

The effect on the performance is evident in the results of Table 3, where often the best results are obtained by the decentralized algorithm (and often the difference is statistically significant according to the t-test). This is because the reduced size of the window for the global algorithm limits the aggregation of the tasks. A reduced window forces the global algorithm to consider fewer allocation possibilities, which in turn results in worse performances. The greedy algorithm is, again, the worst. This is also evident in the load balancing in Table 4. Greedy obtains the worst results in terms of distribution, again because the load is assigned mostly to a single robot. The decentralized algorithm also shows better balancing in terms of tasks, items and travelled distance. In terms of distribution of tasks and items the decentralized algorithm is superior to the others, with lower standard deviation in all cases. This is less clear in terms of travelled distance, where in some cases the global algorithm performs better than the iterative. However the difference between the global and the decentralized algorithms is not as pronounced as the one between the decentralized and the greedy algorithm.

6. Conclusions and future work

We proposed a novel variant of the MAPD problem where robots can transport multiple items in one travel. We devised a distributed approach that guarantees a solution to multi-item MAPD instances requiring weaker constraints with respect to previous approaches and particularly with respect to the concept of well formed infrastructures MAPD instances used in relevant literature.

We evaluated our approach against two centralized competitors in a simulated environment based on the ROS framework and considering realistic maps for the operational environment. Results show that our approach obtains similar results to the global centralized algorithm with much less computational effort. This work paves the way for several interesting research directions, which include the extension of the algorithm to settings where unexpected events may require to re-allocate the tasks or change the computed paths.

Fundings

The research has been partially supported by the project “Dipartimenti di Eccellenza 2018-2022”, funded by the Italian Ministry of Education, Universities and Research (MIUR).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2021.103871.

References

[1] P.R. Wurman, R. D'Andrea, M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Mag. 29 (1) (2008) 9.

[2] A. Farinelli, N. Boscolo, E. Zanotto, E. Pagello, Advanced approaches for multi-robot coordination in logistic scenarios, Robot. Auton. Syst. 90 (C) (2017) 34–44, http://dx.doi.org/10.1016/j.robot.2016.08.010.

[3] H. Ma, J. Li, T.K.S. Kumar, S. Koenig, Lifelong multi-agent path finding for online pickup and delivery tasks, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, SãO Paulo, Brazil, May 8-12, 2017, 2017, pp. 837–845, URL http://dl.acm.org/citation.cfm?id=3091243.

[4] H. Ma, W. Hönig, T.K.S. Kumar, N. Ayanian, S. Koenig, Lifelong path planning with kinematic constraints for multi-agent pickup and delivery, in: AAAI Conference on Artificial Intelligence, 2019, pp. 7651–7658.

[5] M. Čáp, J. Vokřínek, A. Kleiner, Complete decentralized method for online multi-robot trajectory planning in well-formed infrastructures, in: International Conference on Automated Planning and Scheduling, 2015, URL https://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10504.

[6] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: Definitions, variants, and benchmarks, in: SOCS, 2019, pp. 151–158.

[7] P. Surynek, A. Felner, R. Stern, E. Boyarski, An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives, in: SOCS, 2016, pp. 145–146.

[8] G. Wagner, Subdimensional expansion: A framework for computationally tractable multirobot path planning, 2018), http://dx.doi.org/10.1184/R1/6723329.v1, URL https://kilthub.cmu.edu/articles/thesis/Subdimensional_Expansion_A_Framework_for_Computationally_Tractable_Multirobot_Path_Planning/6723329/1.

[9] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artificial Intelligence 219 (2015) 40–66.

[10] K. Vedder, J. Biswas, X*: Anytime multiagent planning with bounded search, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2019, pp. 2247–2249.

[11] R. Luna, K.E. Bekris, Push and swap: Fast cooperative path-finding with completeness guarantees, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One, IJCAI'11, AAAI Press, 2011, pp. 294–300, http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-059.

[12] J. Yu, Average case constant factor time and distance optimal multi-robot path planning in well-connected environments, Auton. Robots 44 (2020) 469–483, URL https://doi.org/10.1007/s10514-019-09858-z.

[13] D. Silver, Cooperative pathfinding, in: Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'05, AAAI Press, 2005, pp. 117–122.

[14] V.R. Desaraju, J.P. How, Decentralized path planning for multi-agent teams with complex constraints, Auton. Robots 32 (2012) 385–403, URL https://doi.org/10.1007/s10514-012-9275-2.

[15] S.S. Chouhan, R. Niyogi, Dmapp: A distributed multi-agent path planning algorithm, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2015, pp. 123–135.

[16] P. Pianpak, T.C. Son, Z.O. Toups, W. Yeoh, A distributed solver for multi-agent path finding problems, in: Proceedings of the First International Conference on Distributed Artificial Intelligence, 2019, pp. 1–7.

[17] A. Gilboa, A. Meisels, A. Felner, Distributed navigation in an unknown physical environment, in: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, 2006, pp. 553–560.

[18] K. Okumura, M. Machida, X. Défago, Y. Tamura, Priority inheritance with backtracking for iterative multi-agent path finding, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), 2019, pp. 535–542.

[19] S. Bhattacharya, V. Kumar, Distributed optimization with pairwise constraints and its application to multi-robot path planning, Robot.: Sci. Syst. VI 177 (2011).

[20] J. Hoffmann, Ff: The fast-forward planning system, AI Mag. 22 (3) (2001) 57.

[21] S.S. Chouhan, R. Niyogi, Dimpp: a complete distributed algorithm for multi-agent path planning, J. Exp. Theor. Artif. Intell. 29 (6) (2017) 1129–1148.

[22] B.P. Gerkey, M.J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, Int. J. Robot. Res. 23 (9) (2004) 939–954.

[23] G. Korsah, A. Stentz, M. Dias, A comprehensive taxonomy for multi-robot task allocation, Int. J. Robot. Res. 32 (12) (2013) 1495–1512.

[24] D. Portugal, R.P. Rocha, Cooperative multi-robot patrol with bayesian learning, Auton. Robot. 40 (5) (2016) 929–953, http://dx.doi.org/10.1007/s10514-015-9503-7.

[25] A. Hussein, A. Khamis, Market-based approach to multi-robot task allocation, in: 2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR), 2013, pp. 69–74, http://dx.doi.org/10.1109/ICBR.2013.6729278.

[26] A. Farinelli, L. Iocchi, D. Nardi, Distributed on-line dynamic task assignment for multi-robot patrolling, Auton. Robots 41 (2017) 1321–1345, URL https://doi.org/10.1007/s10514-016-9579-8.

[27] X. Chen, P. Zhang, G. Du, F. Li, A distributed method for dynamic multi-robot task allocation problems with critical time constraints, Robot. Auton. Syst. 118 (2019) 31–46, http://dx.doi.org/10.1016/j.robot.2019.04.012.
[28] P. Scerri, A. Farinelli, S. Okamoto, M. Tambe, Allocating tasks in extreme teams, in: Proc. of AAMAS 05, Utrecht, Netherland, 2005, pp. 727–734.
[29] D. Portugal, L. Iocchi, A. Farinelli, A ROS-based framework for simulation and benchmarking of multi-robot patrolling algorithms, 2019, pp. 3–28, http://dx.doi.org/10.1007/978-3-319-91590-6_1.

Antonello Contini has graduated from the University of Verona with a Master's Degree in Computer Science and Engineering in March 2020.

Since then he is a research assistant at the Department of Computer Science, working with Dr. Alessandro Farinelli on the development of coordination algorithms for Multi-Robot systems.

His research interests comprise Multi-Agent and Multi-Robot intelligent systems with a focus on the path planning and coordination problems.

In particular, he has worked on the Multi-Agent Pickup and Delivery (MAPD) problem and on this topic he co-authored a paper for the 2020 AAMAS conference.

Alessandro Farinelli is a full professor at University of Verona, Department of Computer Science, since December 2019. His research interests focus on developing novel methodologies for Artificial Intelligence systems applied to robotics and cyber–physical systems. In particular, he focuses on multi-agent coordination, decentralized optimization, reinforcement learning and data analysis for cyber–physical systems. Alessandro Farinelli was principal investigator for several national and international research projects in the broad area of Artificial Intelligence. His research contributions target mainly international journals in the area of Artificial Intelligence (e.g., Artificial Intelligence journal and Journal of Artificial Intelligence Research) and Autonomous Robotic Systems (Autonomous Robots and Robotics and Autonomous Systems). The main scientific conferences he contributes to (both as organizer and speaker) include the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), the International Joint Conference on Artificial Intelligence (IJCAI) and the International Conference on Intelligent Robots and Systems (IROS).