A Hierarchical Approach for Joint Task Allocation and Path Planning

Florence Ho, Ryota Higa, Takuro Kato, and Shinji Nakadai

Abstract—**This letter addresses the joint task allocation and path planning problem, whereby a fleet of vehicles must be optimally assigned to service multiple given tasks while their planned paths must be collision-free. Such a problem composed of two tightly coupled optimization problems has a high complexity with the number of tasks and the number of vehicles, thus optimal solvers do not scale to large size instances. Therefore, we propose a novel method to solve this problem, HTAPPS, which introduces a hierarchical resolution framework. Our proposed approach decomposes a given instance into three levels of abstractions and associated amount of details that progressively filter the search space. This allows us to reduce the computational effort required when performing task allocation and multi-agent path planning jointly. We perform simulations on automated warehouse scenarios, and compare our approach to baseline solvers. The obtained results show that our proposed approach is able to solve large size instances within a limited time.**

Index Terms—**Hierarchical planning, multi-agent path planning, task allocation.**

I. Introduction

AUTOMATED logistics systems are applied across a diverse range of settings, including warehouses, truck delivery, drone delivery, and so on. The implementation of such systems will require the development of efficient techniques for processing large size instances. In this letter, we consider autonomous guided vehicles (AGVs) employed within automated warehouses. All AGVs must be optimally allocated to several given tasks according to a defined cost function, typically the total travel distance. A task is hereby considered as a location in the given warehouse to pick up a quantity of items ordered by a customer.

Thus, two tightly coupled optimization problems arise, 1) a task allocation problem, whereby the given tasks must be allocated optimally to each vehicle, and 2) a multi-agent path planning problem, whereby all conflicts between vehicles, i.e. predicted collisions in the planned paths to service the given tasks, must be avoided. Hence, we address the joint task allocation and path planning problem, whereby we aim to simultaneously determine an optimal task allocation plan for multiple vehicles while ensuring that the vehicles' paths are collision-free.

Although several works have addressed the Vehicle Routing Problem (VRP) [6] that relates to task allocation, and the Multi-Agent Path Finding (MAPF) problem [19] that relates to multi-agent path planning, few works have addressed the combination of these two problems. Since VRP and MAPF are both NP-hard problems [6], [27] to optimize, the coupling of these two problems is even more complex to solve optimally, thus the application of optimal solvers does not scale to large instances. Therefore, we propose a novel resolution approach that introduces a hierarchical processing strategy, the "Hierarchical Task Allocation and Path Planning Solver (HTAPPS)". In HTAPPS, a hierarchy composed of three ordered "levels" is introduced, whereby each level represents a certain degree of details considered in the planning. The highest level generates a high-level plan which is subsequently refined in the lower levels. With this hierarchical framework, we aim to improve the scalability in the resolution of the joint task allocation and path planning problem. We hereby assume our approach is performed at planning time, hence before any vehicle executes its plan. We propose two main contributions:

- We propose a novel approach that introduces a hierarchical framework to solve the joint task allocation and multi-agent path planning problem. We consider three levels of hierarchy, whereby each level represents a certain amount of details provided in the search. The lower the level, the higher the amount of details considered.
- Within the introduced hierarchical framework, we define several abstraction criteria and associated processes to filter and reduce potentially unnecessary computations, and make use of parallelization.

We compare the performance in runtime and costs of our proposed approach to two baseline solvers, 1) an approach that first solves a Capacitated VRP (CVRP) instance then applies a standard MAPF solver to resolve existing conflicts, and 2) the Extended Time Dependent VRP (TDVRP) approach from [1] that simultaneously solves the joint task allocation and path planning problem with the use of time-dependent costs. We perform simulations experiments based on conventional automated warehouses scenarios as used in [12], [13], [25]. The rest of the letter is structured as follows. Section II presents the background on related works in MAPF and VRP. Section III formalizes our problem. Section IV describes our proposed hierarchical approach. Section V presents the experimental evaluation and analysis. Section VI summarizes the letter and perspectives for future directions.

Received 31 March 2024; accepted 17 August 2024. Date of publication 6 September 2024; date of current version 8 October 2024. This article was recommended for publication by Associate Editor R. Gross and Editor M. A. Hsieh upon evaluation of the reviewers' comments. *(Corresponding author: Florence Ho.)*

Florence Ho and Ryota Higa are with Data Science Laboratories, NEC Corporation, Kanagawa 211-8666, Japan, and also with the Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan (e-mail: florence-ho@nec.com; r-higaryouta@nec.com).

Takuro Kato is with the TICO-AIST Cooperative Research Laboratory for Advanced Logistics, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba 305-8560, Japan (e-mail: takuro.katou@aist.go.jp).

Shinji Nakadai is with Data Science Laboratories, NEC Corporation, Kanagawa 211-8666, Japan, and also with Intent Exchange, Todaimae HiRAKU GATE, Tokyo 113-0023, Japan.

Digital Object Identifier 10.1109/LRA.2024.3455787

II. Related Works

A. Multi-Agent Path Finding (MAPF)

In MAPF, the aim is to determine collision-free paths for a given set of agents that minimize a global cost objective, such as the sum of costs or the makespan. Each agent is assumed to move from a start location to a given goal location. MAPF has mostly been studied in 2D environments, for a variety of real world applications such as Amazon warehouses [15]. Several variants of MAPF have been proposed, in particular, few works address MAPF while considering task allocation. [13], [14] propose a sequential resolution by first allocating tasks and then planning collision-free paths. The anonymous MAPF [2], [4], [8] assumes the assignment of exactly one task to each vehicle. [21] propose to solve the multi-goal MAPF with the Hamiltonian-CBS approach, but the tasks (referred as goals) assigned to each agent is determined beforehand and only their order can be changed, and it assumes that goal vertices must be visited at least once. Refs. [9] and [22] address a similar problem, the Target Assignment and Path Finding (TAPF), but they aim to sequentially assign each agent a unique task before moving to the next task. In contrast, our work assumes an offline setting and addresses the allocation of multiple tasks with demands whereby the tasks need to be assigned to the agents while considering the capacity constraints of each agent and ensuring collision-free paths. Recently, several works that address scalability issues in MAPF in automated warehouses scenarios have been proposed [11], [12], but few works have considered a hierarchical approach for resolution. [28] proposed a spatial decomposition as a hierarchy, while [20] and [24] proposed a decomposition based on move constraints to solve conflicts and conflict patterns among agents' plans respectively. All these works focus on solving MAPF only. In our work, we address a coupling of MAPF with VRP. In particular, we propose a novel hierarchical strategy for resolution that considers levels of hierarchy related to the tasks in a given instance. Another relevant problem that uses hierarchical approaches, is Task and Motion Planning (TAMP), that aims to determine a sequence of symbolic actions and associated motion plans [7], [23]. This setting focuses on the pick-up and placement of objects by manipulators and what actions to do. In our work, we focus on path planning of AGVs and we do not consider the manipulation of objects and motion planning level.

B. Vehicle Routing Problem (VRP)

In VRP, the aim is to determine routes for a fleet of vehicles to serve a set of customers while minimizing a given objective such as the total travel cost for the fleet [10]. Several assumptions can be added to the problem, leading to variants and specializations that address a wide variety of real world applications [6]. For example, the vehicles can have a limited carrying capacity of the items that must be delivered, in which case the problem is called Capacitated VRP (CVRP). The multi-agent Traveling Salesman Problem (TSP) [26] generalizes the TSP by considering multiple agents to assign but without a finite capacity unlike CVRP. In [1], an approach based on the Time Dependent VRP (TDVRP) [16] and called Extended TDVRP, was proposed to solve the joint task allocation and path planning problem. This method incorporates time-dependent costs into the VRP resolution process to account for detected conflicts, i.e. predicted collisions, that result in path replanning costs. So a task allocation plan may be modified and improved according to the determined time-dependent costs. However, this approach does not scale to large instances, having to alternate with the resolutions of the two NP-hard problems. Thus, we hereby propose an approach that scales to large instances with the novel use of a hierarchical process, while retaining the concept of time dependent costs to address the joint task allocation and path planning problem.

III. Problem Definition

In this section, we formulate the joint task allocation and path planning problem.

A. Problem Settings and Assumptions

We define the input and assumptions of our problem.

We consider an undirected graph $G = (V, E)$. V denotes the set of vertices and each vertex $v \in V$ corresponds to a possible location for agents. $E \subseteq V \times V$ denotes the set of edges, and each edge $e = (u, v) \in E$ corresponds to a move action from u to v. An edge from a vertex to itself means that an agent can wait at the vertex. The cost of an edge e is denoted as $c(e) \in \mathbb{R}$.

A task Γ_i is characterized by a pickup location $v_i \in V$ and a demand $d_i \in \mathbb{N}$ which is the quantity of an item to pick up as ordered by a customer.

An agent a_m is a vehicle. Each agent has a maximum payload capacity of items to pick up $q_m \in \mathbb{N}$ with $\forall i, q_m > d_i$, a start vertex $v_s^m \in V$, and a goal vertex $v_g^m \in V$ to reach. Since we consider automated warehouses whereby vehicles pick up items on storage shelves, we assume that the start of each agent is the location of an inventory station s, and the goal is the location of another inventory station g to bring all picked up items. Time is assumed to be discretized, and in every time step, each agent is situated in one of the vertices of G.

B. Problem Formulation

An instance consists of a set of M agents $A = \{a_1, \ldots, a_M\}$ and a set of N tasks $\Gamma = \{\Gamma_1, \ldots, \Gamma_N\}$. The aim is to determine for each agent a_m a sequence of tasks $MP_m = \{\Gamma_a, \Gamma_b, \ldots, \Gamma_c\}$ such that the cost objective $Cost$ is minimized. MP_m has an associated path $p_m = \{p(v_s^m, \Gamma_a), p(\Gamma_a, \Gamma_b), \ldots, p(\Gamma_c, v_g^m)\} = \{v_1^m, \ldots, v_L^m\}$ with $v_1^m = v_s^m$, $v_L^m = v_g^m$, and $(v_i^m, v_{i+1}^m) \in E$, $i = 1, \ldots, L-1$. $p(\Gamma_a, \Gamma_b)$ denotes a feasible path going from the location of task Γ_a to the location of task Γ_b, and composed of a sequence of vertices, i.e. in space and time. We hereby define a solution as $MP = (MP_m)_{m \in [\![1, M]\!]}$ and the cost objective as the sum of travel costs of all agents $Cost(MP) = \sum_{m=1}^{M} c(p_m)$ with $c(p_m) = \sum_{i=1}^{L-1} c((v_i^m, v_{i+1}^m))$ that returns the travel cost of the path of each agent. We define the constraints of the problem:

1) An agent with allocated tasks starts from its start location and finishes at its goal location;
2) Each task is performed by exactly one agent;
3) The sum of demands of the allocated tasks to an agent a_m must not exceed its capacity q_m;
4) The solution MP must be conflict-free on all agents' paths. A conflict occurs if two agents occupy the same location at the same timestep (node conflict), or if two agents move on the same edge of G in opposite directions during the same time interval (edge conflict).

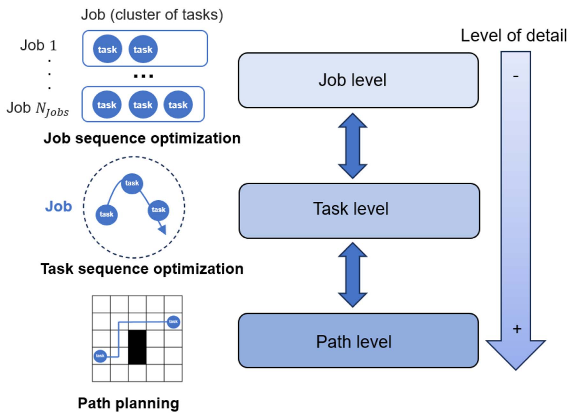


Fig. 1. Our considered hierarchical decomposition framework with a 3-level hierarchy: Job level, task level, and path level.

C. Extended Considerations

We hereby focus on a baseline setting to evaluate the hierarchical process for the resolution of the joint task allocation and path planning problem. Several other additional considerations can be made that we are interested to study in future works, in particular on the problem settings assumptions.

We consider a 2D map, but other grid configurations are applicable to our presented approach. Also, we assume a set of homogeneous agents, but heterogeneous agents can also be considered. We hereby consider pickup tasks, but other types of tasks could also be defined with different constraints such as adding service times to tasks to stay for n timesteps, adding sets of locations for a task, adding dependencies between tasks, time windows, and so on.

Other practical considerations may include specific kinematic constraints for each agent, and a lifelong setting whereby tasks appear dynamically [25]. Other objectives different from the sum of travel costs can also be considered, such as the makespan objective. All these additional considerations could easily be incorporated into our framework, as the core resolution approach would remain the same.

IV. Hierarchical Task Allocation and Path Planning Solver (HTAPPS)

A. Overview

As the name suggests, our proposed approach HTAPPS, introduces a hierarchy to process instances of the joint task allocation and path planning problem.

Precisely, we consider three levels of hierarchy that we define as "Job Level", "Task Level", and "Path Level" as shown in Fig. 1. We proceed from the most abstract representation to the most detailed one so that the search space and computational effort may be reduced. Moreover, the introduction of such a hierarchical approach allows us to incorporate the following features into the processing: filtering or pruning, and parallelization.

In the highest level, the "Job level", the set Γ is divided into N_{Jobs} clusters, i.e. jobs that contain distinct tasks of Γ. This level introduces the additional constraint that a given job must be assigned to exactly one agent, i.e. two agents cannot be assigned to the same job. The goal in this level is to determine an optimal allocation of jobs according to the "high level" costs that we describe in Section IV-C. Then in the "Task Level", we consider the tasks inside each job, and the goal in this level is to determine an optimal allocation sequence of tasks according to the costs associated to the tasks of the given job. Details related to this level are discussed in Section IV-C.

In the lowest level, "Path Level", we consider the paths between pairs of task locations, and the goal in this level is to determine an optimal path between two given task locations in the given map. Here, we consider finding an optimal path according to the distance of the shortest path computed to move from a given task location to another task location namely with an A* path planner. Details related to this level are discussed in Section IV-D. All the costs considered in all the hierarchy levels refer to travel distances, as we aim to optimize the defined objective function which is the sum of travel costs. Since our problem is a combination of two coupled optimization problems, task allocation (with CVRP) and multi-agent path planning (with MAPF), we apply the proposed hierarchical framework to these two alternating phases to reduce the overall computational effort.

Thus, our proposed approach is composed of two main phases described in the following sections: 1) a task allocation phase, and 2) a conflict detection and resolution phase for conflicts between multiple vehicles' paths. Algorithm 1 shows the pseudocode of HTAPPS.

Algorithm 1: HTAPPS.

```
    Input: Agents Set $A = (a_m)_{m\in[\![1,M]\!]} = (q_m, s_m, g_m)_{m\in[\![1,M]\!]}$, Tasks Set $\Gamma = (\Gamma_i)_{i\in[\![1,N]\!]} = (v_i, d_i)_{i\in[\![1,N]\!]}$, Number of Jobs $N_{Jobs}$
    /* Initialization */
1:  $Jobs \leftarrow$ Generate_jobs($\Gamma, N_{Jobs}$)
2:  $MP, C_{HL}, P_{HL} \leftarrow$ Hierarchy_TA($A, Jobs$)
    /* Solve joint task allocation and path planning with hierarchy */
3:  $MP \leftarrow$ Hierarchy_MA($A, Jobs, MP, C_{HL}, P_{HL}$)
4:  return $MP$                              ▷ Conflict-free solution
```

B. Job Generation

First, the initialization step generates the set of jobs $Jobs$ containing N_{Jobs} (Algorithm 1 line 1). The tasks associated to each job are selected with a clustering method, such as a k-means approach, to determine clusters of tasks based on the closeness of their locations in the given map. So, we define $k = N_{Jobs}$ clusters and after running k-means, each job j has n_j associated tasks $j.tasks$. Also, a "job demand" d_j^{Job} is associated to each job j, $d_j^{Job} = \sum_{\Gamma_k \in j.tasks} d_k$, which is the sum of the demands of all the n_j tasks associated to j. In particular, we add two constraints on the tasks selected for each job by k-means: (i) $\forall j \in Jobs, n_j \geq 2$, each job must contain at least two tasks; (ii) $\forall j, d_j^{Job} \leq \min_{m\in[\![1,M]\!]} q_m$, the demand of any job is lower than any agent capacity so that the job can be assigned to any agent.

Also, N_{Jobs} is fixed such that: $\frac{\sum_{i\in[\![1,N]\!]} d_i}{\min_{m\in[\![1,M]\!]} q_m} \leq N_{Jobs} \leq \frac{N}{2}$, so that the generated jobs can be assigned to any agent within its associated capacity.

Algorithm 2: Hierarchy_TA($A, Jobs$).

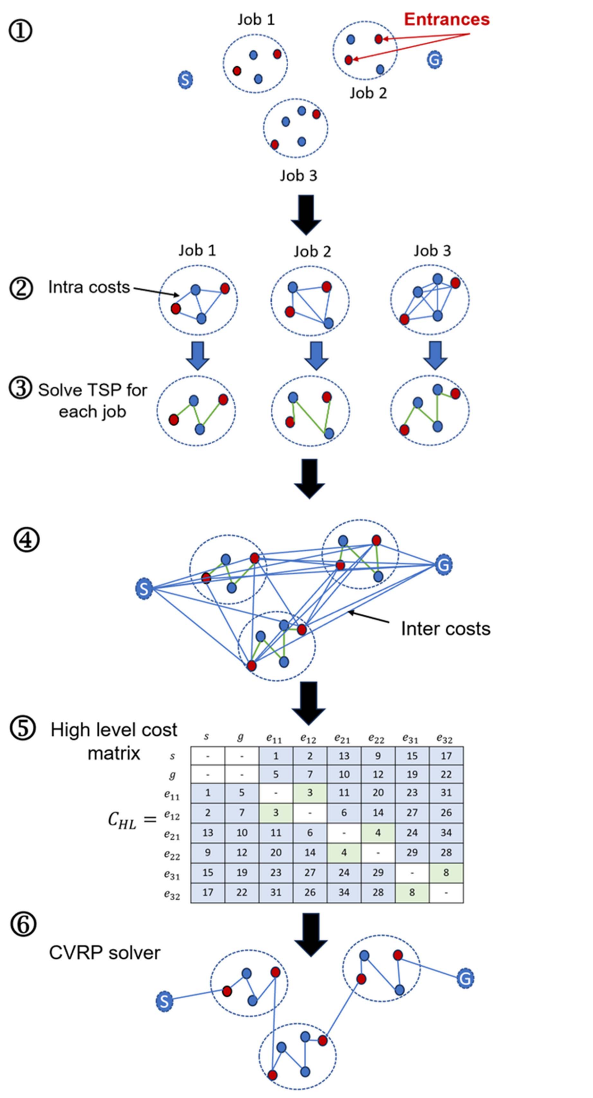
```
1: $C_{HL}, P_{HL} \leftarrow \emptyset, \emptyset$    ▷ High level cost and path matrices
2: for $j \in Jobs$    ▷ Run in parallel e.g. one thread for each job do
3:     $j.entrances \leftarrow$ HL_entrances $(j)$    ▷ Determine entrances
4:     $intra_j \leftarrow$ Compute_intra $(j)$    ▷ Compute an intra-sequence of tasks in $j$ and its cost
5:     $inter_j \leftarrow$ Compute_inter $(j, Jobs, A)$    ▷ Compute inter-costs and inter-paths with other jobs and start and goal
6:     $C_{HL}, P_{HL} \leftarrow$ Add_HL_Mat $(intra_j, inter_j, C_{HL}, P_{HL})$    ▷ Add intra- and inter-costs and paths to high level cost matrix $C_{HL}$ and path matrix $P_{HL}$ respectively
7: end for
8: $MP \leftarrow$ CVRP_solver $(A, Jobs, C_{HL}, P_{HL})$    ▷Determine jobs sequence i.e. solution
9: return $MP, C_{HL}, P_{HL}$
```

C. Hierarchy in Task Allocation

In this step, we apply the hierarchical framework to generate a task allocation solution with $Hierarchy\_TA()$ (Algorithm 1 line 2) as described in Algorithm 2 and we explain each step shown in Fig. 2 in the following.

1) At the Job level, for each job j, a pair of tasks of j is designated as "entrances" $j.entrances \subset j.tasks$ with the use of a predefined criterion (Algorithm 2 line 3). For example, such a criterion may be selecting tasks situated in the convex hull of the given job. On an abstract level, a job can be seen as a high level task that encapsulates low level tasks. Thus, to determine a solution, we need to connect all jobs in a sequence, and this is done by using the entrances. The entrances of a job do not have a fixed order of allocation, so they can be selected in any order in the resolution process. Without loss of generality, we consider two entrances per job in the following.[1]
2) We introduce in the following the distinction of *intra-costs* that represent the costs between tasks within the same job, and *inter-costs* that represent the costs between different jobs, and start and goal depots.
   In the Task level, for each job j, we compute the intra-costs.
3) Then, we determine a task sequence MP_j^{Job} for each job j optimized according to the intra-costs (Algorithm 2 line 4). To do so, we solve a TSP instance since one agent must perform a sequence of all the tasks of the job j from a start and goal locations that are the pair of entrances associated to j. We then obtain the job j intra-cost $intra_j = Cost(MP_j)$ of the path. This step is processed in parallel, for example with a thread allocated to each job.
4) Then, we compute the set of inter-costs $inter_j$ containing the costs between the job j and the other jobs, the start, and the goal depots (Algorithm 2 line 5). This step can also be processed in parallel by defining a distribution of the computations between jobs.
5) The intra-cost and inter-costs of all jobs are gathered into a "high level" cost matrix C_{HL} (Algorithm 2 line 6), whose structure is shown in Fig. 2 and a "high level" path matrix P_{HL} that contains the associated paths. In Fig. 2 and in the following, we denote e_{jk} to refer to the entrance k of a job j.
6) Finally, a CVRP solver is used to determine a feasible sequence of jobs optimized according to the high level costs in C_{HL} (Algorithm 2 line 8), and we obtain a solution MP with the sequence of tasks resulting from the sequence of jobs. The CVRP solver is able to determine

[1]In automated warehouses scenarios, the selected entrances tend to usually be located towards the "west" and "east" extremities of each given job, so two entrances are sufficient in this configuration. We observed that this criterion for selection of entrances typically adheres to the configurations to obtain optimal solutions in our considered scenarios. Other adapted criteria for entrances selection may be defined for other scenarios different from automated warehouses.

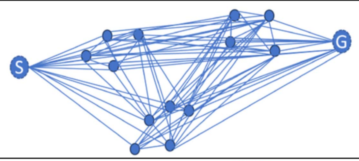
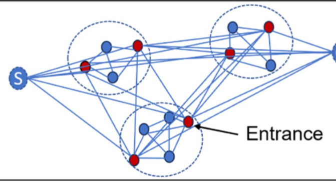


$C_{HL} =$

	s	g	e_{11}	e_{12}	e_{21}	e_{22}	e_{31}	e_{32}
s	-	-	1	2	13	9	15	17
g	-	-	5	7	10	12	19	22
e_{11}	1	5	-	3	11	20	23	31
e_{12}	2	7	3	-	6	14	27	26
e_{21}	13	10	11	6	-	4	24	34
e_{22}	9	12	20	14	4	-	29	28
e_{31}	15	19	23	27	24	29	-	8
e_{32}	17	22	31	26	34	28	8	-

Fig. 2. Hierarchical process applied in task allocation step. Example instance with three jobs with four or five tasks in each. In (5), the high level cost matrix C_{HL} contains intra-costs in the green cells and inter-costs in the blue cells. In (6), one agent is assigned a sequence of tasks in this example.

TABLE I
COMPARISON OF THE NUMBER OF EDGES TO COMPUTE IN THE TASK ALLOCATION PHASE BETWEEN A STANDARD SOLVER AND OUR PROPOSED APPROACH

Approach	Example: $N = 13$ tasks, $N_{Jobs} = 3$ jobs
Standard	
Proposed	

a feasible solution by considering the demand of each job j, d_j^{Job} as defined in Section IV-B, so that the allocation plan do not exceed the capacity of the given agent.

Properties. In particular, this process allows us to reduce the search space normally required to determine a solution for a CVRP instance. The standard resolution process requires computing the costs of all combinations of pairs of tasks between all N tasks including start and goal depots of the given instance, i.e. $O(N^2)$. In our approach, the entrances allow us to prune the search space to reduce the number of costs computations, i.e. $O(\max(N_{Jobs}^2, N_{Jobs} \cdot \max_{j \in Jobs} n_j^2))$, with $N_{Jobs} < N$, which is even improved with the integration of parallelization enabled by the hierarchy. In the example in Table I, the total number of edges to compute with a standard approach would be 104 edges, while with our approach, this total number would be down to 43 edges.

This step has the same properties as the used CVRP solver, and we hereby use Google OR-Tools,[2] which is not guaranteed to terminate if an instance has no solution. To avoid running an exhaustive search for too long, the solver uses a time limit which stops the search even if a solution has not been found. Usually, if an instance has a feasible solution, the Google OR-Tools solver returns a solution rapidly within the fixed time limit, and it will always return a feasible assignment if it exists.

D. Hierarchy in Multi-Agent Path Planning

After obtaining a solution, several agents may be assigned to perform sequences of tasks accordingly. Thus there may be conflicts on their paths, i.e. predicted collisions. A conflict is resolved by computing new paths for the agents involved and this may increase the total cost of the given solution. Hence, the sequence of tasks may be changed to find a solution with a lower cost. We outline the process in Algorithm 3 that is based on Conflict-Based Search (CBS) [18]. The first step is to detect conflicts between agents' paths, whereby the earliest conflict $conf = (a_i, a_j, v, t)$ between two agents a_i and a_j is detected in the current solution (Algorithm 3 line 10). The second step is conflict resolution, whereby the paths of the involved agents are replanned with an A* path planner to avoid the detected conflicts (Algorithm 3 line 18). Next, the resulting time-dependent costs

[2]https://developers.google.com/optimization/

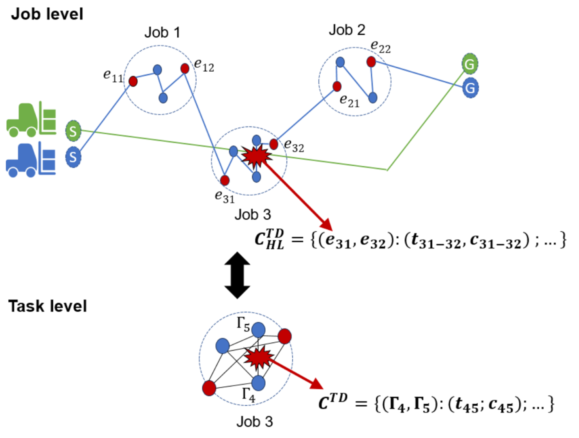

Job level
Job 1
Job 2
e_{11}
e_{12}
e_{22}
e_{21}
e_{32}
e_{31}
Job 3
S
S
G
G
$C_{HL}^{TD} = \{(e_{31}, e_{32}) : (t_{31-32}, c_{31-32}) ; \dots\}$
Task level
Γ_5
Γ_4
Job 3
$C^{TD} = \{(\Gamma_4, \Gamma_5) : (t_{45}; c_{45}); \dots\}$

Fig. 3. Hierarchical process for the multi-agent path planning phase. Example with two assigned agents, blue and green, (for ease of understanding, the jobs and tasks sequences assigned to the green agent is omitted and only its path is represented). A conflict occurs between the agents' paths, in particular, within Job 3 for the blue agent and more precisely between tasks Γ_4 and Γ_5. So, for the blue agent, at the Job level, we add a high level time-dependent cost to C_{HL}^{TD} associated to Job 3. At the Task level, we add a corresponding low level time-dependent cost to C^{TD} associated to Γ_4 and Γ_5.

and associated paths are set up and lead to the processing of a TDVRP instance.

1) Hierarchical Framework for TDVRP Resolution: In the following, we describe the adaptation of the TDVRP resolution process to the proposed hierarchical framework. Time-dependent costs reflect the costs incurred by the detected conflicts. Precisely, they are costs that vary depending on the time an agent departs from a given task location to go to another task location.

For the Task level, we define a dictionary C^{TD} that stores the time dependent costs for pairs of task locations associated to detected conflicts, and P^{TD} that stores the associated paths between these locations. If a conflict is detected for an agent departing from a task location Γ_a from time t_{ab} to a task location Γ_b, the cost $c(p(\Gamma_a, \Gamma_b))$ of the replanned path considering this conflict becomes a time dependent cost $C^{TD}[(\Gamma_a, \Gamma_b)] = (t_{ab}, c(p(\Gamma_a, \Gamma_b)))$. This cost is considered if a given agent departs from the task location Γ_a to go to the task location Γ_b at time t_{ab} (Algorithm 3 line 20 and 21). Otherwise, if an agent departs from Γ_a to go to Γ_b at a time that is not in C^{TD}, then the cost and path stored in C_{HL} and P_{HL} are applied.

For the Job level, we also define a high level time-dependent costs and associated paths dictionaries, C_{HL}^{TD}, P_{HL}^{TD} that are updated along C^{TD} and P^{TD} when a conflict is detected (Algorithm 3 line 22). Similarly to C_{HL} in Section IV-C, C_{HL}^{TD} is composed of two types of costs, intra and inter time-dependent costs. Since each job has associated entrances as described in Section IV-C, if a conflict occurs on the path within a job j as shown in Fig. 3, we define an intra time-dependent cost, for example, $C_{HL}^{TD}[(e_{j1}, e_{j2})] = (t_{j1\_j2}, c_{j1\_j2})$, with t_{j1_j2} the time when an agent enters job j from its entrance e_{j1}, and exits by its entrance e_{j2}, and an associated cost $c_{j1\_j2} = \sum_{(\Gamma_i, \Gamma_k) \in MP_j^{Job} | t_{j1\_j2}} c_{ik}$ the sum of the costs of the task sequence MP_j^{Job} of j corresponding to the time of entrance t_{j1_j2} from e_{j1}. Similarly, if a conflict occurs on the path between two jobs j and k, with their entrances e_{j1} and e_{k1} respectively, we define an inter time-dependent cost $C_{HL}^{TD}[(e_{j1}, e_{k1})] = (t_{j1\_k1}, c_{j1\_k1})$.

For each level in the hierarchy, there are associated ways to improve a solution during the search: at the Job level, we can only change the sequence of jobs for each agent or the partitioning of jobs between agents, while at the Task level, we can only change the sequence of tasks inside a given job. To determine an improved solution with the time dependent costs, we apply a metaheuristic solver, Local Search (LS) [3], which is flexible and adaptable to many optimization problems, as optimal solvers do not scale in the resolution of TDVRP with the number of tasks [16]. LS is often used to solve such complex optimization problems with efficiency, but without theoretical guarantees on optimality, and may lead to suboptimal solutions [5]. LS starts from an initial solution and replaces it when a better solution is found in its neighborhood until a stopping condition becomes valid. Moreover, when searching neighborhoods, several solutions may have no prospects of improvements. To avoid processing such solutions, we consider properties on the considered moves, that restrict the search. In particular, when processing a detected conflict, we consider the spatial location of the given conflict, so we avoid the moves that only affect the tasks or jobs that are situated after the given conflict since they will have no impact on the solution. We first apply LS by searching through different combinations of jobs sequences that may lead to an improved solution (Algorithm 3 line 23). In this step, we gather two types of obtained solutions into MP_set, (i) solutions that have an improved cost from the current solution, and (ii) solutions that have the same cost as the current solution. We gather jobs according to a predefined criterion, such as the number of detected conflicts, into $Jobs\_set$. Then, the process moves to the Task level. In this step, for each candidate solution obtained in MP_set, LS is applied to solve a Time Dependent TSP (TDTSP) [16] to modify the sequence of tasks inside the given jobs in $Jobs\_set$ (Algorithm 3 line 24). We hereby consider the time-dependent costs associated to tasks inside a given job with C^{TD}. This step is also processed in parallel, with a thread run for each candidate solution of MP_set.

Properties. As this step is based on the CBS algorithm, it has the same properties on completeness as CBS. CBS has been proven to return a solution if one exists [17] but is not guaranteed to identify an unsolvable problem. CBS always generates a feasible solution if it exists since all constraints on the detected conflicts during the search are stored in each node of the search tree. So our step returns a feasible solution if it exists but cannot identify unsolvable instances.

Algorithm 3: Hierarchy_MA($A, Jobs, MP, C_{HL}, P_{HL}$).

```
1:  $C^{TD}, P^{TD}, C^{TD}_{HL}, P^{TD}_{HL} \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$
2:  $R.solution = MP$; $R.cost = Cost(MP)$;
    $R.constraints = \emptyset$                          ▷ Root node
3:  $R.matrix = (C_{HL}, P_{HL}, C^{TD}_{HL}, P^{TD}_{HL}, C^{TD}, P^{TD})$
4:  Insert $R$ to $OPEN$
5:  while $OPEN \neq \emptyset$ do
6:    $P \leftarrow OPEN.pop()$     ▷ Node with lowest solution cost
7:    if $P$ has no conflict then
8:      return $P.solution$    ▷ Return conflict-free solution
9:    else
10:     $conf \leftarrow$ Earliest conflict $(a_i, a_j, v, t)$ in $P.solution$
11:     for each agent $a_i$ in $conf$ do
12:       $N \leftarrow$ new node
13:       $N.constraints \leftarrow P.constraints + (a_i, v, t)$
14:       $N.matrix \leftarrow P.matrix$
15:       $N.solution \leftarrow P.solution$
16:       $(\Gamma_a, \Gamma_b) \leftarrow Get\_Locations(a_i, conf)$  ▷ Get pair of task locations between where the conflict occurs for $a_i$
17:       $t_{ab} \leftarrow Get\_tstart(a_i, p(\Gamma_a, \Gamma_b))$  ▷ Get start time of path from $\Gamma_a$ to $\Gamma_b$
18:       $p(\Gamma_a, \Gamma_b) \leftarrow Path\_Plan(\Gamma_a, \Gamma_b, t_{ab}, N.constraints)$   ▷ Replan path from $\Gamma_a$ to $\Gamma_b$
19:       $C^{TD}, P^{TD}, C^{TD}_{HL}, P^{TD}_{HL} = Get\_TD(N.matrix)$
20:       $C^{TD}[(\Gamma_a, \Gamma_b)] \leftarrow C^{TD}[(\Gamma_a, \Gamma_b)] \cup \{(t_{ab}, c(p(\Gamma_a, \Gamma_b)))\}$
21:       $P^{TD}[(\Gamma_a, \Gamma_b)] \leftarrow P^{TD}[(\Gamma_a, \Gamma_b)] \cup \{(t_{ab}, p(\Gamma_a, \Gamma_b))\}$   ▷ Update time dependent cost and path dictionaries
22:       $C^{TD}_{HL}, P^{TD}_{HL} \leftarrow$Update_HL_TD($conf, Jobs, N.matrix$)   ▷ Update high level time dependent cost and path dictionaries (cf. explanation in Section IV-D)
          /*Update solution w/ TDVRP solver (LS)*/
23:       $MP\_set, Jobs\_set \leftarrow$ LS_Jobs($A, Jobs, N.matrix, N.solution$) ▷ Apply Local Search to Job level
24:         $N.solution \leftarrow$ LS_Tasks($A, MP\_set, Jobs\_set, N.matrix$)  ▷ Apply Local Search to Task level
25:         $N.cost = Cost(N.solution)$
26:         Insert $N$ to $OPEN$
27:     end for each
28:   end if
29: end while
```

V. Experiments

We evaluate the performance of our proposed HTAPPS approach on automated warehouses instances with configurations including corridors and rows of shelves as obstacles as shown in Fig. 4, and also considered in several recent works [12], [13], [25].

A. Setup

We compare the proposed HTAPPS approach to two baseline approaches that solve the joint task allocation and path planning problem, namely i) Standard CVRP solver + MAPF solver (denoted CVRP+MAPF) that first generates a mission plan with a CVRP solver, then solves all existing conflicts with a MAPF solver without changing the initial mission plan, and ii) Extended TDVRP (denoted Ext. TDVRP) that simultaneously solves the joint task allocation and path planning problem as in [1].

All approaches are implemented in Python and run on a laptop computer Intel Core i9-9980HK CPU @2.40GHz with 32GB RAM. For the applications of a CVRP solver, we used the Google OR-Tools library. We set a runtime limit of 10 minutes for the 21×35 map instances, and 20 minutes for the 33×46 map instances. For each experiment setting, we generate 30 instances with randomly generated tasks locations. We fix the maximum capacity q for all agents to 8 tasks. We vary the numbers of tasks N from 40 to 720 tasks as indicated in Table II. We fix the number

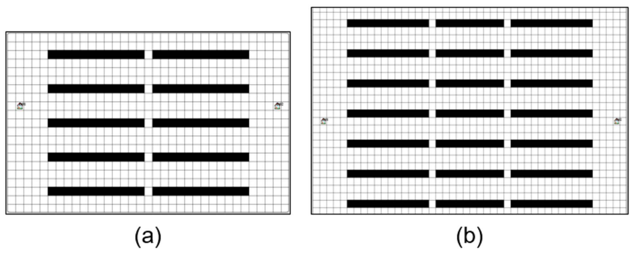


Fig. 4. 4-neighbor grid maps that represent simulated warehouse instances configurations (a) 21×35 cells map and (b) 33×46 cells map. The black cells represent static obstacles, e.g. shelves with items stored.

TABLE II
PARAMETERS SETUP FOR BOTH MAPS INSTANCES

#Tasks (N)	**#Jobs** (N_{Jobs})	**Min. #Agents** (M_{min})
40	10	5
80	20	10
120	30	15
240	60	30
400	100	50
*480	*120	*60
*600	*150	*75
*720	*180	*90

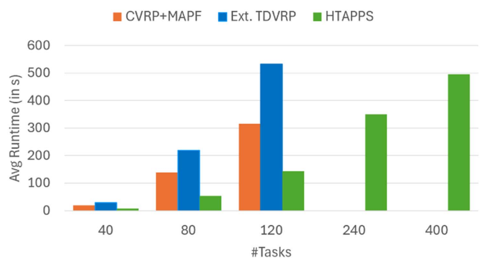
*denotes the parameters used for the 33×46 map instances only.

of jobs $N_{Jobs} = \frac{N}{4}$, which is in the range of possible values of N_{Jobs} as mentioned in Section IV-B. For indication, provided the number of tasks N, we also indicate a minimum number of allocated agents M_{min} by dividing the number of tasks N by the capacity $q = 8$. This represents the number of agents needed if all agents were to be assigned tasks at full capacity. Because agents are allocated to each task according to a CVRP solver, the actual number of allocated agents may be higher than M_{min} in certain instances, with some agents not being assigned tasks at full capacity.

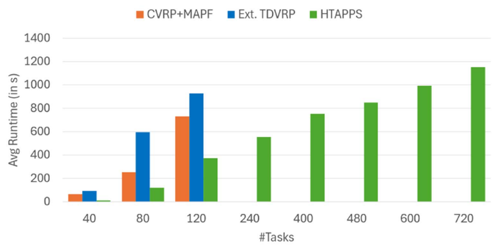
B. Results Analysis

1) Runtime: As shown in Fig. 5, in both maps, the runtime of HTAPPS is significantly lower than the two other baselines. For instances with a small number of tasks such as 40 tasks, the amount of difference is small. From 80 to 120 tasks, the gap in runtime becomes significantly larger, with HTAPPS being in average three to five times faster than Ext.TDVRP and CVRP+MAPF. From 240 tasks, whereby the instances become larger and denser, both baselines timed out, while the proposed HTAPPS approach was able to finish within the time limit. Overall, HTAPPS has the lowest runtime, while Ext. TDVRP has the highest runtime.

2) Cost: In Fig. 6, we observe that the average solution costs obtained with HTAPPS are higher than Ext. TDVRP. The average difference in costs varies from around 5% to around 15% difference between the two approaches. As expected, Ext. TDVRP always returns the lowest costs among the three compared approaches. On the other hand, HTAPPS and CVRP+MAPF both return higher costs, with HTAPPS possibly returning close

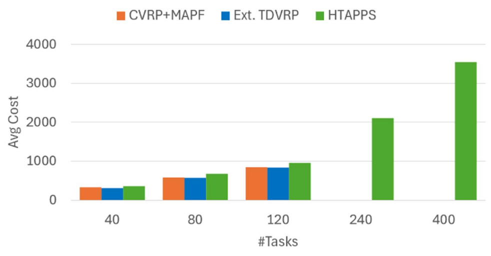


(a) 21×35 map instances.

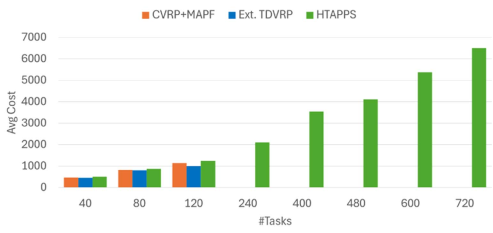


(b) 33×46 map instances.

Fig. 5. Comparison of average runtime.



(a) 21×35 map instances.



(b) 33×46 map instances.

Fig. 6. Comparison of average solution cost.

costs to CVRP+MAPF in certain instances. As described in Section IV, HTAPPS does not have theoretical guarantees on the optimality of the obtained solution due to its inherent hierarchical structure. The hierarchy filters information to reduce the search space, hence improving the runtime and scalability, while the search space restrictions may prevent finding certain improvements in a given solution. However, the performance of HTAPPS presents a trade-off between runtime and solution costs that varies in function of its chosen parameters namely N_{Jobs} that we discuss in the next subsection.

3) Analysis of HTAPPS Parameters Influence on Performance: First, the value of N_{Jobs} influences the solutions costs optimality by controlling the jobs generation process. N_{Jobs} can be selected within the possible range of values defined in Section IV-B, and the aim is to find a balance in runtime and solution optimality by fixing a value of N_{Jobs} that is not too close to the bounds of the defined range, as these would return high runtimes. We hereby empirically selected a value of $N_{Jobs} = \frac{N}{4}$, as it provided the best trade-offs in runtime and costs in average by comparing the performances with different values of N_{Jobs}. Also, our approach may produce further suboptimal solutions in scenarios where a mismatch between vehicle capacity and job demand happens. By definition, a job can be assigned to one vehicle if its demand does not exceed the vehicle's capacity. So, it may not be possible to assign several jobs to a vehicle due to the sum of their demands exceeding its capacity, while one task belonging to a job could have possibly be assigned within the vehicle's capacity, leading to a better solution cost.

The clustering strategy also influences solution optimality, as it determines the jobs compositions in tasks and is related to N_{Jobs}. We observed that the proximity criterion used for clustering typically adheres to the corridor-like configuration to obtain optimal solutions in our considered warehouse scenarios.

VI. Conclusion

In this letter, we proposed a novel resolution approach to solve the joint task allocation and path planning problem. The proposed approach consists in a hierarchical process, whereby three levels of decomposition are considered. Each level allows us to reduce the search space for task allocation and multi-agent path planning, and filter information considered as unnecessary. In particular, this allows us to incorporate efficient processing techniques such as parallelization. We evaluate our proposed method in automated warehouses instances, and compare its scalability with two baseline approaches that solve the joint task allocation and path planing problem. We empirically showed that our approach is able to process in a limited runtime the large and dense instances considered, unlike the baseline solvers.

References

[1] A. Aggarwal, F. Ho, and S. Nakadai, "Extended time dependent vehicle routing problem for joint task allocation and path planning in shared space," in *2022 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 12037–12044.

[2] Z. A. Ali and K. Yakovlev, "Improved anonymous multi-agent path finding algorithm," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 17291–17298.

[3] F. Arnold and K. Sörensen, "Knowledge-guided local search for the vehicle routing problem," *Comput. Operations Res.*, vol. 105, pp. 32–46, 2019.

[4] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5816–5823, Jul. 2021.

[5] R. Elshaer and H. Awad, "A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants," *Comput. Ind. Eng.*, vol. 140, 2020, Art. no. 106242.

[6] B. Golden, S. Raghavan, and E. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*, vol. 43. Berlin, Germany: Springer, 2008.

[7] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *IEEE Trans. Robot.*, vol. 39, no. 1, pp. 239–252, Feb. 2023.

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

[9] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proc. 17th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2018, pp. 757–765.

[10] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *Eur. J. Oper. Res.*, vol. 59, no. 3, pp. 345–358, 1992.

[11] C. Leet, J. Li, and S. Koenig, "Shard systems: Scalable, robust and persistent multi-agent path finding with performance guarantees," in *Proc. 36th AAAI Conf. Artif. Intell.*, 2022, pp. 9386–9395.

[12] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proc. 35th AAAI Conf. Artif. Intell.*, 2021, pp. 11272–11281.

[13] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Proc. 18th Int. Conf. Auton. Agents Multi-Agent Syst.*, 2019, pp. 1152–1160.

[14] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proc. 15th Int. Conf. Auton. Agents Multi-Agent Syst.*, 2016, pp. 1144–1152.

[15] H. Ma et al., "Overview: Generalizations of multi-agent path finding to real-world scenarios," in *Proc. Workshop Multi-Agent Path Find.*, 2016.

[16] C. Malandraki and M. Daskin, "Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms," *Transp. Sci.*, vol. 26, pp. 185–200, 1992.

[17] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 563–569.

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

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

[20] T. Sugawara, S. Kurihara, T. Hirotsu, K. Fukuda, and T. Takada, "Predicting possible conflicts in hierarchical planning for multi-agent systems," in *Proc. 4th Int. Conf. Auton. Agents Multi-Agent Syst.*, 2005, pp. 813–820.

[21] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 14, pp. 12409–12417.

[22] Y. Tang, Z. Ren, J. Li, and K. Sycara, "Solving multi-agent target assignment and path finding with a single constraint tree," in *Proc. IEEE Int. Symp. Multi-Robot Multi-Agent Syst.*, 2023, pp. 8–14.

[23] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *2017 IEEE Int. Conf. Robot. Automat.*, 2017, pp. 4044–4051.

[24] T. Walker, D. Chan, and N. Sturtevant, "Using hierarchical constraints to avoid conflicts in multi-agent pathfinding," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2017, vol. 27, pp. 316–324.

[25] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *2022 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 9964–9971.

[26] C. Yang and K. Y. Szeto, "Solving the traveling salesman problem with a multi-agent system," in *2019 IEEE Congr. Evol. Comput.*, 2019, pp. 158–165.

[27] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 3612–3617.

[28] H. Zhang et al., "A hierarchical approach to multi-agent path finding," in *Proc. Int. Symp. Combinatorial Search*, 2021, vol. 12, pp. 209–211.