Hierarchical Planning for Heterogeneous Multi-Robot Routing Problems via Learned Subteam Performance

Jacopo Banfi[1*], Andrew Messing[2*], Christopher Kroninger[3], Ethan Stump[3], Seth Hutchinson[2], and Nicholas Roy[1]

***Abstract*— This paper considers a particular class of multi-robot task allocation problems, where tasks correspond to heterogeneous multi-robot routing problems defined on different areas of a given environment. We present a hierarchical planner that breaks down the complexity of this problem into two subproblems: the high-level problem of allocating robots to routing tasks, and the low-level problem of computing the actual routing paths for each subteam. The planner uses a Graph Neural Network (GNN) as a heuristic to estimate subteam performance for specific coalitions on specific routing tasks. It then iteratively refines the estimates to the real subteam performances as solutions of the low-level problems become available. On a testbed problem of a heterogeneous multi-robot area inspection problem as the base routing task, we empirically show that our hierarchical planner is able to compute optimal or near-optimal (within 7%) solutions approximately 16 times faster (on average) than an optimal baseline that computes plans for all the possible allocations in advance to obtain precise routing times. Furthermore, we show that a GNN-based estimator can provide an excellent trade-off between solution quality and computation time compared to other baseline (non-learned) estimators.**

I. INTRODUCTION

This paper considers a particular class of multi-robot task allocation problems, where tasks correspond to heterogeneous multi-robot routing problems defined on different areas of a given environment. The goal is to minimize the time needed to complete all of the routing tasks. This class of problems is representative of a number of scenarios where allocating subteams of robots to individual areas would be beneficial. For example, in search and rescue operations spanning very large environments, battery constraints could prevent a single robot from being employed in more than one area. Alternatively, in a military scenario, strategic areas might need to be simultaneously inspected for the presence of adversaries prior to moving a convoy through them. As a final example, consider a communication-constrained patrolling scenario, where assigning subteams to individual areas can guarantee that the robots will have sufficient inter-group networking to promptly respond to the detection of an intruder. These types of problems inherently display a hierarchical structure: if we knew in advance the time needed by each possible subteam of robots to complete each possible routing task, we could first identify the optimal allocation of subteams to areas of interest, and then compute the actual subteam paths only for that assignment. A straightforward approach to addressing the first stage optimally would be precomputing the paths for all the possible assignments of subteams to tasks, which would give all the possibles routing times as a by-product. Unfortunately, even leaving aside the combinatorial nature of the assignment problem, it is often the case that the multi-robot routing problem resulting from the subteam assignment is NP-hard, and a good solution can only be obtained with computationally expensive algorithmic approaches, such as formulating the routing problem as a Mixed-Integer Linear Program (MILP), that typically require several seconds to minutes or hours to run. To reduce the overall planning time, the search for a good assignment should solve the routing tasks problems in a lazy fashion, by starting from the most promising allocations of subteams to tasks. However, knowing the potential utility of an allocation typically requires knowing its routing plan, eliminating the advantage of the lazy approach.

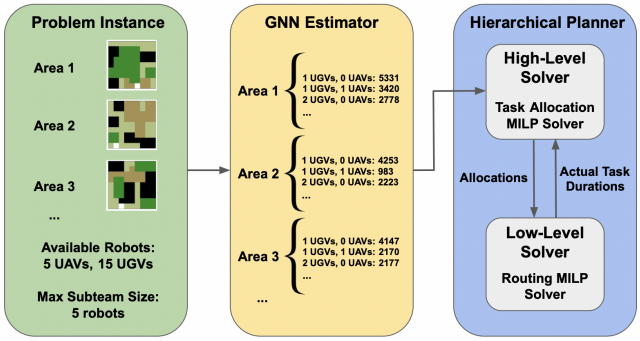

Fig. 1: The proposed hierarchical planning framework applied to our testbed problem. A GNN is first used to estimate the time needed by different subteams to inspect different areas of an environment. A high-level solver uses these estimates to compute a high-level allocation, while a low-level solver computes the actual paths with a specialized routing algorithm. The actual task durations are then used to update the GNN estimates for the high-level solver, which can then compute a new allocation with an improved set of estimates.

We note that the allocation of subteams requires only knowing the *costs* of the different routing plans for a given allocation, and not the actual plans themselves. If we can estimate these costs without solving the corresponding routing problems at the same time, we can defer computing routing plans until a tentative allocation has been decided.

Based on these observations, we present a hierarchical planner capable of breaking down the complexity of the original problem into two natural subproblems: the high-

*Equal contribution.

[1]CSAIL, Massachusetts Institute of Technology (MIT), Cambridge (MA) 02139, USA. Corresponding Author Email: jbanfi@mit.edu.

[2]IRIM, Georgia Institute of Technology (GT), Atlanta (GA) 30332, USA.

[3]DEVCOM ARL, Adelphi (MD) 20783, USA.

This material is based upon work supported under the DCIST CRA by the Army Research Laboratory under Cooperative Agreement Number W911NF-17-2-0181.

level problem of allocating robots to routing tasks, and the low-level problem of computing the actual routing paths for only a selected subset of all the possible assignments of subteams to areas. As multi-robot routing problems are typically defined on graph-represented environments, the planner uses Graph Neural Networks (GNNs) as a heuristic to estimate the subteam performance for specific coalitions on specific routing tasks. Iteratively the planner then refines these estimates to the real subteam performances as solutions of the low-level problem becomes available. We introduce a testbed problem having a heterogeneous multi-robot area inspection problem as basic routing task, for which we consider again a solution approach based on a traditional Mixed-Integer Linear Programming formulation. A schematic view of the proposed planning framework is shown in Fig. 1.

In routing task allocation problems containing up to 45 robots and 20 areas to inspect, we empirically show that our approach is always able to compute optimal or near-optimal (within 7%) solutions 16 times faster (on average) than an an optimal baseline that computes plans for all the possible allocations in advance to obtain precise routing times. We also show that a GNN-based estimator provides an excellent trade-off between solution quality and computation time compared to other baseline (non-learned) estimators.

II. RELATED WORK

This paper lies at the intersection of two traditional robotics research avenues — Multi-Robot Task Allocation and Multi-Robot Routing — and a more recent machine learning one — Graph Neural Networks. They are briefly discussed below relatively to the concepts that appear in this paper.

A. Multi-Robot Task Allocation

In this work, we consider a bi-level task allocation problem whose first level can be thought of as a task assignment problem, and framed into the ST-MR-IA (Single-Task Robots, Multi-Robot Tasks, Instantaneous Assignment) category of the multi-robot task allocation taxonomy introduced by Gerkey and Matarić [1], a problem often referred to as *coalition formation*. Classical ST-MR-IA problems can be formalized and solved by casting them as Set Partitioning Problems (SPPs) [2]. Two main differences set our problem apart from classical coalition formation, preventing it from being formalized as a SPP: the optimization goal (we minimize the maximum cost, instead of the sum), and the fact that in our problem the actual coalition costs of the multi-robot routing tasks are not provided as part of the problem. Multi-robot routing problems, instead, cannot be easily framed in the Gerkey and Matarić taxonomy, as noted by Korsah et al. [3]. These latter authors propose a more general taxonomy, where the routing problem we consider in the second level of this work can be classified as an ST-MR-TA (Single-Task Robots, Multi-Robot Tasks, Time-Extended Assignment) problem with in-schedule dependencies (ID). As a result, the overall problem solved by our framework is ST-MR-TA-ID.

B. Multi-Robot Routing Problems

Vehicle routing problems (VRPs) are a family of combinatorial optimization problems that generalize the famous Traveling Salesman Problem (TSP)[1] in different ways, like the presence of "time windows" for visiting locations or the availability of multiple vehicles [4]. VRPs are typically NP-hard, and have traditionally been a subject of study of the operations research community. The last decade witnessed a growing interest in this field from the robotics community as well, under flavors that more closely match the application scenarios of multi-robot systems. Examples can be found in applications like search [5], monitoring [6], coverage [7], agriculture [8], package delivery [9], [10], and, more generally, the execution of spatially distributed "tasks" [11], [12]. Oberlin et al. [11] and Prasad et al. [12] study routing problems that are similar to the basic routing task considered in our testbed problem, but with modeling assumptions that might not be applicable to many robotic scenarios. Solving routing problems by formulating them as MILPs is one of the most pupular solution approaches [5], [6], [9]–[11], and for this reason the one we use for our experimental campaign. We remark that the proposed planning framework is not designed around the specific application used as a testbed in this paper. Instead, it could be used with any of the routing problems mentioned above as base routing tasks.

C. Graph Neural Networks

Graph neural networks (GNNs) are a class of neural network architectures specifically designed to learn on graph data. A complete overview of this rapidly growing subfield of machine learning is out of the scope of this paper; the reader is referred to Wu et al. [13] for a survey, and to Dwivedi et al. [14] for a recent GNN benchmarking paper. Recently, a number of works used GNNs to learn policies for solving task allocation and scheduling problems; see [15] and the references therein. In this paper, we proceed along an orthogonal direction: allocations (our policies) are derived via a traditional planning method — Mixed-Integer Linear Programming — and GNNs are employed as a heuristic to estimate the parameters of our planning problem, i.e., the costs of executing routing tasks that involve coalitions of robots. Our proposed GNN-based estimation method is usable in conjunction with any multi-robot routing problem and associated solution algorithm, which allows our proposed framework to be very general.

III. PROBLEM DESCRIPTION

Let $\mathcal{T}$ be a set of tasks consisting of multi-robot routing problems defined on separate areas of a given environment. Here, a heterogeneous team of robots has to be deployed in order to perform these tasks. In this paper, we assume we have at our disposal n unmanned aerial vehicles (UAVs) and m unmanned ground vehicles (UGVs). We assume that robots belonging to the same set are homogeneous, but our

[1]Informally, the TSP can be stated as: "Given a set of cities and pairwise distances between them, what is the shortest tour that visits each city exactly once?"

approach is easily generalizable to account for more types of robots. We assume that each task $t \in \mathcal{T}$ can be assigned to at most a given number of robots in total. For simplicity, in this paper it is assumed that all tasks can be assigned to at most B robots. Although the proposed approach is very general, in this paper we focus on the particular class of multi-robot inspection tasks formalized in Section V, in which some tasks $\mathcal{T}_g \subseteq \mathcal{T}$ can only be carried out by teams containing at least one ground vehicle.[2] We formally define a *subteam* to be a pair (i, j), with $i, j \in \mathbb{N}_0$[3] and such that $0 \leq i \leq n$, $0 \leq j \leq m$, and $1 \leq i + j \leq B$. The set containing all the possible subteams is denoted by $\mathcal{S}$.

We also require a *routing algorithm* that, given a subteam and a routing task, computes a routing plan for that subteam and the routing task. In this paper, the chosen routing algorithm corresponds to a MILP-based solution approach, in which the solver is allowed to run for a given number of seconds. More details will be given in Section V; for the moment, we will simply refer to this MILP as the *routing MILP*. Let $\omega : \mathcal{S} \times \mathcal{T} \to \mathbb{Q}^+$ be the function associating each subteam-task pair with the corresponding routing time, when the plan is computed with the routing MILP. For infeasible assignments (subteams with no ground vehicle assigned to tasks in $\mathcal{T}_g$), ω can return an arbitrary value.

Informally, our goal is to create a set of subteams and allocate them to the routing tasks so that (a) all tasks $t \in \mathcal{T}_g$ are assigned to subteams containing at least one ground vehicle, and (b) the time needed to complete the longest task, known as the *makespan*, is minimized. More formally, let $\mathcal{A}$ be the set of all the feasible allocations, and let $M(A)$ be the *makespan* induced by allocation $A \in \mathcal{A}$. The goal is to compute the allocation A^* with minimum makespan:

$$A^* \in \arg\min_{A \in \mathcal{A}} M(A). \tag{1}$$

As discussed in the introduction, computing the function ω for all the possible input pairs might be extremely time-consuming in general. Therefore, we assume that we also have access to an estimator of ω, namely, a function $\hat{\omega} : \mathcal{S} \times \mathcal{T} \to \mathbb{Q}^+$ that attempts to describe the same relation, and whose values can be retrieved very quickly for all the possible input pairs. We can leverage this estimator to reduce the number of actual routing problems that need to be solved, as shown in the next section.

IV. HIERARCHICAL PLANNING ALGORITHM

We start by providing a formal definition of the problem in terms of a MILP model, which we call the *task allocation MILP*, assuming that we have access to the function ω. We define the following sets of variables:

- α^t, integer: UAVs allocated to routing task t;
- β^t, integer: UGVs allocated to routing task t;
- γ_i^t, binary with $0 \leq i \leq \min(n, B)$, $i \in \mathbb{N}_0$: 1 iff i UAVs are allocated to task t;
- δ_j^t, binary with $0 \leq j \leq \min(m, B)$, $j \in \mathbb{N}_0$: 1 iff j UGVs are allocated to task t;
- ϵ_{ij}^t, binary with $(i, j) \in \mathcal{S}$: 1 iff i UAVs and j UGVs are allocated to task t;
- ζ, continuous: the makespan value.

The model can be expressed compactly, with the help of some logical and non-linear constraints, as follows:

$$\min \zeta \quad \text{s.t.} \tag{2}$$
$$\sum_{t \in \mathcal{T}} \alpha^t \leq n \tag{3}$$
$$\sum_{t \in \mathcal{T}} \beta^t \leq m \tag{4}$$
$$1 \leq \alpha^t + \beta^t \leq B \quad \forall t \in \mathcal{T} \tag{5}$$
$$\beta^t \geq 1 \quad \forall t \in \mathcal{T}_g \tag{6}$$
$$\gamma_i^t = 1 \iff \alpha^t = i \quad 0 \leq i \leq \min(n, B), \; i \in \mathbb{N}_0, \forall t \in \mathcal{T} \tag{7}$$
$$\delta_j^t = 1 \iff \beta^t = j \quad 0 \leq j \leq \min(m, B), \; j \in \mathbb{N}_0, \forall t \in \mathcal{T} \tag{8}$$
$$\epsilon_{ij}^t = \gamma_i^t \delta_j^t \quad \forall (i, j) \in \mathcal{S}, \forall t \in \mathcal{T} \tag{9}$$
$$\zeta \geq \omega\big((i, j), t\big)\epsilon_{ij}^t \quad \forall (i, j) \in \mathcal{S}, \forall t \in \mathcal{T}. \tag{10}$$

In this model, constraints (3)-(4) make sure that the number of UAVs-UGVs allocated across the different tasks is consistent with their availability; constraint (5) expresses the fact that we have to allocate at least one robot to each task and at most B; constraint (6) enforces the special requirement of allocating at least one ground robot to tasks in $\mathcal{T}_g$; constraints (7)-(8) are used to set the value of the γ_i^t and δ_j^t variables according to their definition; constraints (9) use the γ_i^t and δ_j^t variables to set the value of the ϵ_{ij}^t to 1 if and only if both γ_i^t and δ_j^t are equal to 1, representing the selection of a subteam with i UAVs and j UGVs for the execution of task t. Finally, constraints (10) are used to define the makespan value. The linear version of constraints (7)-(9) can be found in the appendix.

Now, suppose we only have access to the estimator of ω, i.e. $\hat{\omega}$. We can use this estimator to precompute all the assignment costs appearing in constraints (10). Then, we can obtain a potential allocation by solving such modification of the original task allocation MILP. Given this potential

Algorithm 1: Hierarchical Planner Main Function.

```
1  function plan(T, n, m, B, ω̂)
2      best_sol ← ∅
3      best_paths ← ∅
4      while not converged and not timed out do
5          solve task allocation MILP with ω̂
6          retrieve assignments (α^t, β^t) for all t ∈ T
7          for t ∈ T do
8              if (α^t, β^t) is a new assignment then
9                  use routing MILP to obtain ω((α^t, β^t), t), paths
10                 ω̂((α^t, β^t), t) ← ω((α^t, β^t), t)
11         update best_sol and best_paths, if needed
12     return best_sol, best_paths
```

[2] Our approach can easily handle the presence of a similar set of tasks $\mathcal{T}_a \subseteq \mathcal{T}$ requiring at least one aerial vehicle, but these will not be present in our testbed problem.

[3] In this paper, we use $\mathbb{N}_0$ to denote the set of nonnegative integers, and $\mathbb{Q}^+$ to denote the set of positive rational numbers.

allocation, we can then compute the actual times needed to complete these tasks by solving the associated routing problems with the routing MILP on each task $t \in \mathcal{T}$ and the corresponding assignment (α^t, β^t). This information can be used to update our duration estimates, recompute a set of candidate assignments, and so on, until we obtain a solution for which the corresponding actual routing times have all been computed in previous iterations. This termination condition ensures that the makespan computed by the task allocation MILP is consistent with that of the actual solution, and also that subsequent iterations will not be able to improve the current solution when using a deterministic MILP solver. This procedure constitutes the basic building block of the proposed planning algorithm: we leverage the task allocation MILP as a first level solver to compute abstract solutions, ground them by computing the actual plans with the routing MILP, and repeat until convergence. Algorithm 1 summarizes this procedure. Note, in Lines 8-10, that we can avoid needing to recompute plans and actual routing times for task-subteam pairs already encountered in previous iterations of the algorithm. Algorithm 1 might in general converge to a local minimum. However, it is possible to bound the quality of the solution with that of the estimator. The following result is easy to prove:

Proposition 1. *Let κ be the largest estimation error given by $\hat{\omega}$, i.e. $|\omega(s,t) - \hat{\omega}(s,t)| \leq \kappa$ for all the feasible assignments (s,t) with $s \in \mathcal{S}$, $t \in \mathcal{T}$, and let A_{ALG} be the allocation returned by Algorithm 1. Then, if at least one task allocation MILP was solved to optimality, $M(A_{ALG}) \leq M(A^*) + 2\kappa$.*

In order to escape from a local minimum once Algorithm 1 has converged, it is sufficient to compute additional accurate routing times by solving the routing MILP for assignments not yet explored during the search, update the corresponding original estimates, and run again Algorithm 1 with the updated estimates. This simple addition makes the planner *anytime*, guaranteeing convergence to the optimal solution when having a sufficient time budget at disposal. Different heuristics can be used to select these additional assignments. In our current implementation, the precise routing time of a single additional subteam is computed for each task by solving the corresponding routing MILP. The subteam for task t is selected as the one inducing the smallest change in terms of number of added or removed robots w.r.t. the subteam assigned to task t in the current best solution.[4]

V. ROUTING TASK EXAMPLE: HETEROGENEOUS MULTI-ROBOT AREA INSPECTION

We now proceed to the definition of the routing task used in our experimental evaluation (Section VI). A set of robots $\mathcal{R}$ has been allocated to the inspection of an area. Robots can be of two types: UAVs and UGVs; robots of the same type are assumed to be identical. The area is represented as an undirected, connected, simple graph $G = (V, E)$. Vertices represent *locations* that need to be inspected, while edges represent the existence of a route connecting two *locations*. Vertices are partitioned into two sets, F and U, denoting forested and unforested locations, respectively. Vertices in F can only be inspected by ground vehicles, while vertices in U can be inspected by both types of robots. We use $c_r^i(v)$ to denote the time needed to inspect vertex $v \in V$ with robot r, and $c_r(i,j)$ to denote the travel time of robot r along an edge $(i,j) \in E$. With a little abuse of notation, we also use $c_r(i,j)$ to denote the time needed to travel between two generic vertices $i, j \in V$ when following the shortest path computed on the graph having the original travel times as weights. The presence of two types of robots, coupled with the assumption of homogeneity among robots of the same type, implies that only two sets of travel and inspection times need to be defined in order to fully specify the problem.

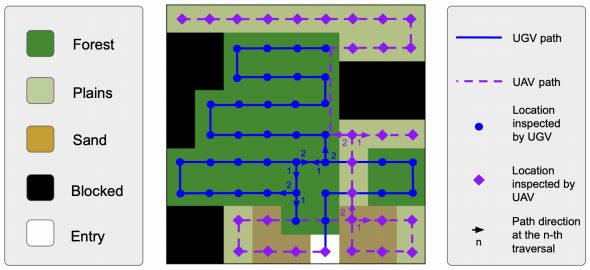

Fig. 2: Example paths for 1 UAV and 1 UGV in Area 1 in the example of Fig. 1, which is modeled as a 4-connected grid graph. "Plains" and "Sand" locations are both unforested, and the UGV takes more time to inspect the latter type. The "Entry" cell represents the vertex adjacent to the entrance. More details about the generation of this type of environment are given in Section VI-A.

The robots are initially placed at the entrance of the area to be inspected, which is denoted by a special entry vertex (not to be inspected) $s \in V$.[5] An inspection path π_r for a robot $r \in \mathcal{R}$ is defined as a sequence of inspected vertices starting at the entry vertex: $\pi_r = [v_1 = s, v_2, \ldots, v_k]$. The time needed to execute such path is given by

$$c(\pi_r) = \sum_{j=1}^{k-1} c_r(v_j, v_{j+1}) + \sum_{j=2}^{k} c_r^i(v_j). \tag{11}$$

A feasible solution to this routing problem corresponds to a path set P, containing one path for each robot, in which each vertex $v \in V \setminus \{s\}$ is inspected exactly once. Note that the paths are completely independent, and robots are not required to share any information with their teammates at execution time. An example instance and corresponding feasible solution are shown in Fig. 2. The goal is to compute the path set P with minimum makespan b defined as

$$b = \max_{r \in \mathcal{R}} c(\pi_r). \tag{12}$$

A. Routing MILP

The routing problem defined above can be formulated as a MILP. Due to space constraints, the full model is not

[4]Other heuristics could also be used. A detailed analysis of the impact of the heuristic is out of the scope of the present paper.

[5]For simplicity, it is assumed in this paper that the time needed by the robots to reach the entry location s once the final allocations have been computed is negligible compared to the time needed by the robots to inspect the whole area.

shown. We simply note that our formulation is inspired by the three-index, flow-based formulation for the multiple TSP discussed by Bektas [16], and adds the necessary changes to accommodate for (a) the presence of two types of agents, (b) the addition of vertices inspection times, and (c) the minimization of the makespan defined in Eq. (12) instead of the sum of the travel costs.

When solving this MILP model in our experiments, we initialize the solver with the solution obtained by a simple greedy algorithm which iteratively adds to one of the robots' paths the vertex inducing the smallest increase in the objective function. This greedy algorithm can therefore also be used as a baseline estimator for the solution provided by the routing MILP, as it consistently provides upper bounds. Similarly, the routing MILP can be relaxed to a linear program, and the solution to the LP can also be used as an estimator, as it consistently provides lower bounds.

B. GNN Estimator

We now present a more sophisticated estimator based on the usage of a neural network. This is composed of three main building blocks, as shown in Fig. 3.

The **GNN block** replicates the Gated Graph Convolutional Network (GCN) architecture in the version proposed by Bresson and Laurent [17]. This network is chosen as the basic building block for our architecture since it was recently shown by Dwivedi et al. [14] to outperform several other architectures at both graph regression and TSP solution generation. Gated GCNs belong to the family of "message passing" GNNs, whose general update rule for the features of node i between layers l and $l+1$ can be written as

$$h_i^{l+1} = f(h_i^l, \{h_j^l : j \in \mathcal{N}(i)\}, \{e_{ij}^l : j \in \mathcal{N}(i)\}),$$

where h_i^l (h_j^l) denotes the feature vector of node i (j) at layer l, $\mathcal{N}(i)$ is the set of neighbors of i in the graph, e_{ij}^l is the feature vector of edge (i, j) at layer l, and f is the network-specific layer mapping. A similar general update rule can be defined for edge features as well.

In this block, the original vertex and edge features are initially embedded into d-dimensional features h_i^0, e_{ij}^0 via a simple linear projection. In this paper, we choose $d = 78$ as done by Dwivedi et al. [14]. Graph convolutions are then applied for L layers, without changing the features' dimension. After the last layer, a standard mean readout layer is used after the last layer to obtain an output of size d.

In the particular area inspection problem considered in this work, the vertex features initially given as input to the GNN block are 3-dimensional: the first two dimensions encode the UGV and UAV inspection times (with UAV inspection time set to 0 in case the vertex belongs to the set F), while the remaining dimension describes whether a vertex is the one adjacent to the depot or not with a binary encoding. Input edge features are instead 2-dimensional and encode UAV and UGV travel times. However, note that an area admits two alternative representations: one based on the original graph G, and one based on the associated complete (fully-connected) graph where travel times between any two

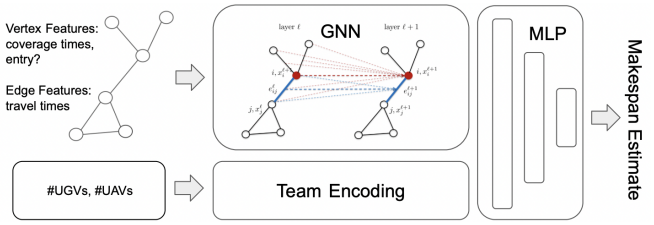

Fig. 3: GNN for predicting the makespan of multi-robot routing tasks. GNN layers image taken from [18].

vertices are obtained by computing shortest paths on the original graph with UAVs and UGVs travel times as weights. The latter encoding results in more memory occupancy and longer training times, but avoids the GNN block the need to recover some of the numerical inputs provided to the routing MILP, i.e. the travel costs for pairs of vertices not directly connected by an edge. Therefore, when the environment is represented as a complete graph, UAV travel times are set to 0 if any of the adjacent vertices belongs to the set F. A standard mean readout layer is used after the last layer to obtain an output of fixed size d. In the remainder of the paper, we will use "GNN" to denote the estimator obtained by training the proposed architecture on the original graphs, and "GNN-C" to denote the estimator obtained by training on the associated complete graphs.

The **Team Encoding block** is a simple linear layer with input size 2, encoding the number of UAVs and UGVs at disposal.

The **MLP block** is a Multilayer Perceptron fed with the concatenation of the outputs of the GNN and Team Encoding blocks, i.e. an input of size 80. Our current architecture consists of 2 more hidden layers with size 41 and 20. The output is 1-dimensional and represents the makespan.

VI. NUMERICAL EXPERIMENTS

We evaluate the proposed hierarchical planning framework via numerical experiments on problems where routing tasks are defined on two different types of environments. Regardless of the environment type, we assume that each task can be assigned to a team of up to five robots. The experimental campaign consists of two parts. The first part aims at assessing the performance of different variants of the network architecture presented in Section V-B. The second part evaluates the planning framework in its entirety. In all the experiments, we use the GUROBI MILP solver [19] with default parameters. The hierarchical planner is run with a 30 minutes timeout, with a deadline of 30 seconds given to GUROBI to solve the routing MILP. (In all the considered planning instances, GUROBI was always able to obtain the optimal solution to the task allocation MILP in less than one second.) All experiments are run on a computer equipped with an Intel i7-7740X CPU, an Nvidia GeForce GTX 1080 Ti GPU, and 32 GB of RAM.

A. Grid Graphs

We start by considering routing tasks defined on areas represented as a grid graphs, an abstraction widely used in

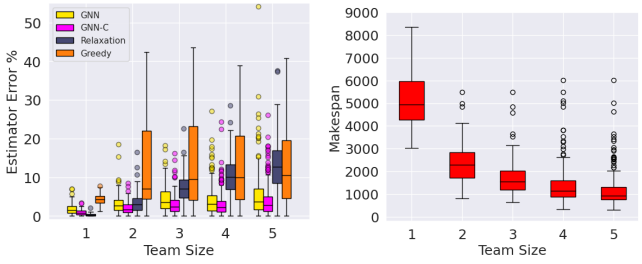

Fig. 4: Grid graphs – Relative errors given by the different estimators on the GNNs test set. GNN-C denotes the GNN trained on complete graphs. For each team size, from left to right: GNN, GNN-C, Relaxation, Greedy. The test set makespan distribution is also shown for reference.

the multi-robot routing literature [5], [6], [10]. In particular, each area is represented as a 9x9 square grid graph with holes modeling untraversable locations. In order to generate environments with a realistic spatial correlation, each area vertex is associated with a location type by applying random rectangular "patches" on a 9x9 image representing the 9x9 grid graph, which are then used to denote the presence of untraversable locations, forested locations, and unforested locations. Unforested locations can, in turn, represent "plains" locations or "sandy" locations, with UGVs taking more time to cover the latter type. Example grid areas are shown in the leftmost block of the diagram in Fig. 1 and in Fig. 2. For each area, travel and inspection times are randomly generated as follows (assuming a generic "time unit" as unit of measurements):

- **UAVs travel time** (between two cells): integer randomly chosen between 5 and 10;
- **UGVs travel time**: integer randomly chosen between 10 and 20;
- **UAVs clear time** (one cell): three times its travel time;
- **UGVs clear time – plains**: four times its travel time;
- **UGVs clear time – sand**: UGVs clear time – plains with a random increase between 30 and 50%;
- **UGVs clear time – forest**: UGVs clear time – plains with a random increase between 60 and 80%.

1) Grid Graphs – GNN Validation: We curate a dataset of composed of 6k samples with a 4200 train – 900 validation – 900 test split, obtained as follows. First, 1000 grid graphs are generated as discussed above, with a 700 train – 100 validation – 100 test split; then, for each graph, we choose randomly 6 subteam configurations for a subteam of maximum size 5. This is done in order to make sure that the network will be exposed to the impact on performance given by different subteams operating on the same graph.

We train different variants of the architecture outlined in Section V-B following the procedure outline by Diwedi et al. [14] —with the Mean Absolute Error (MAE) as the loss function— by considering different combinations of: number of GNN layers ($L = 6$ or $L = 12$), graph type (original graph or complete version), and types of features (vertex only or with edge features). In general we observe, as expected, that increasing the number of layers and the use of edge features always results in better performance for a given architecture. We also observe better performance when training on complete graphs, which however comes at the expenses of longer training times (3 hours vs. 30 minutes). We select the two GNNs that obtain the best test MAEs across all those trained on the original graphs and the complete graphs for the actual planning experiments. These both have $L = 12$ GNN layers and use edge features.[6]

Fig. 4 shows how the chosen network architectures greatly outperform our two baselines, the MILP relaxation and the greedy heuristic, on the test set. Note that the greedy error can also be interpreted as the improvement on the solution quality obtained by solving the MILP model with the greedy solution as initial guess, which is quite significant in spite of the small deadline given to the solver (MIP gaps typically lie in the range 0-15% and increase with the team size).

2) Grid Graphs – Planner Results: We consider planning instances where the number of aerial vehicles is set to the number of areas minus one. This creates challenging problems, in which the planner has to decide which area(s) should be left without aerial vehicles, which can cover unforested areas very efficiently, and which areas should receive extra ground vehicles. The planner is evaluated with different estimators: the GNN and GNN-C estimators, the greedy heuristic estimator, and the MILP relaxation estimator. The quality of the solutions is compared against the ones of an optimal baseline that solves the task allocation MILP with the actual ω function. However, obtaining this function for a single instance of the experiments reported below could take more than 2hrs, while the GNN-based estimator, the greedy estimator, and the MILP relaxation estimator never took more than 35 seconds, 2 seconds, and 2 minutes, respectively, to compute $\hat{\omega}$. When summing these times to the time needed by the anytime hierarchical planner to produce a solution within 7% of the optimal one, we observed speed-ups of approximately 16x and 15x on average when the solution was obtained with the GNN and GNN-C estimators, respectively.

In the first set of experiments, we fix the number of areas to 10, and study the planner performance for 20, 30, and 40 robots on 25 randomly generated instances. Fig. 5 shows the optimality loss of the hierarchical planner run with different estimators at different times, until the 30 minutes timeout is reached. Looking at the initial optimality loss, i.e. the optimality loss obtained for the first allocation returned when Algorithm 1 is run for the first time, we observe generally better performance for the GNN, GNN-C, and MILP relaxation estimators compared to the greedy one. All these estimators already provide near-optimal solutions for teams of 20 robots, and GNN-C also provides them for teams of 30 robots. Interestingly, the relaxation estimator provides better results than the GNN-based ones initially for teams of 40 robots — a perhaps surprising result since GNNs are able to better approximate ω (see again Fig. 4). This

[6]We also trained architectures with $L = 12$ on smaller training sets containing 500 and 600 graphs. Increasing the number of graphs from 500 to 600 had a very limited impact on the networks performance. Instead, when moving from 600 to 700 graphs, we observed in all cases but one (vertex only features) a significant improvement in performance (between 10% and 25%, averaged across 4 different random seeds).

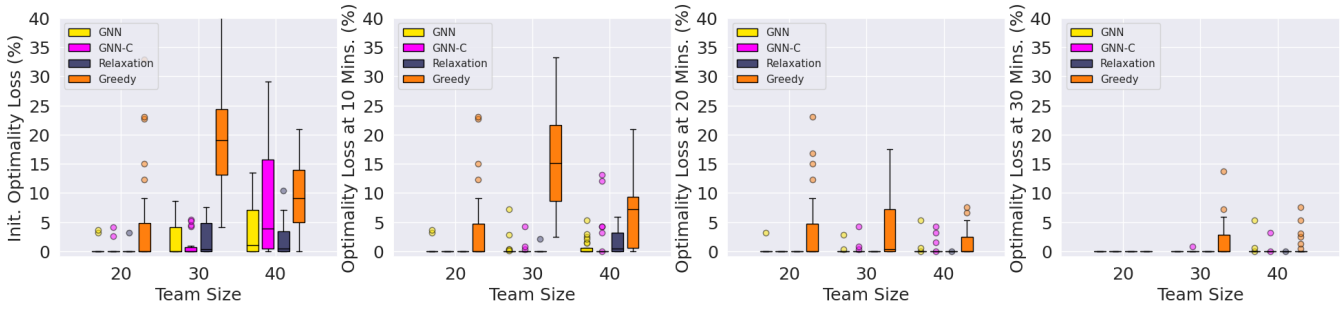

Fig. 5: Grid graphs – planner results: varying the team size (20, 30, 40 robots) for a fixed number of areas (10). Results are shown as traditional box plots (circles represent outliers). For each team size, from left to right: GNN, GNN-C, Relaxation, Greedy.

happens because the MILP relaxation estimates are still able to capture well the impact of the addition/removal of robots to the team across different areas in spite of a consistent (and often significant) underestimation of the makespan. At 10 minutes, both GNN-based estimators now provide optimal or near-optimal solutions in the majority of cases, and in general slightly better results than the MILP relaxation estimator for 40 robots. Letting the planner run for 20 and 30 minutes leads to more and more instances solved to optimality by the GNN, GNN-C, and MILP relaxation estimators, as well as to an increase in the quality of the solutions provided by the greedy estimator.

In the second set of experiments, we fix the number of robots to 40, and study the planner performance for problems containing 10, 15, and 20 areas, again on 25 randomly generated instances. Due to space constraints, plots are not shown. We observe results consistent with the ones above. The GNN and MILP relaxation estimators still provide a significantly better performance than the greedy one. We observe that increasing the number of areas does not seem to have a significant impact on the ability to produce quality solutions and to converge to the optimal ones for the GNN-based and MILP relaxation estimators.

B. Polypixel Environment

We also consider planning instances built as an abstraction of a complex simulated environment dubbed Polypixel, shown in Figure 6. In this case, we assume that a map of the environment is given in advance, with a number of potential locations in each area where inspections will be carried out. The actual inspection locations (top left area: between 20 and 40; top right area: between 30 and 50; bottom area: between 20 and 40) will however be revealed only at planning time. In this case, areas are directly represented with the associated complete graphs, with travel times obtained by assuming that UGVs move at a speed of 5 m/s, while UAVs at 10 m/s. Inspections always take UAVs 5 seconds, while UGVs take 10 seconds to inspect an unforested location and 20 or 30 seconds to inspect a forested one.

1) Polypixel – GNN Validation: We curate a dataset composed of 6300 samples with a 4140 train - 1080 validation - 1080 test split, obtained by randomly sampling 350 graphs from each area and 6 subteams configurations for a subteam of maximum size 5 for each graph. We train different

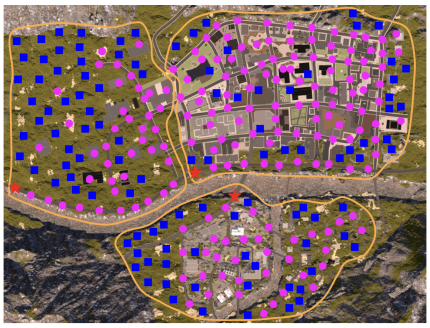

Fig. 6: The Polypixel environment (2.5 $\times$ 3.3 Km^2), which has been divided into three areas of interest (yellow). Blue squares: forested locations, pink circles: unforested locations. Red star: entry point (unforested).

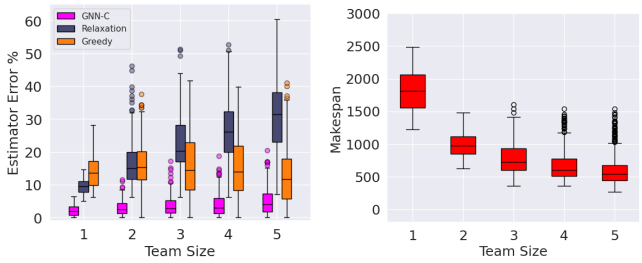

Fig. 7: Polypixel – relative errors given by the different estimators on the GNNs test set. For each team size, from left to right: GNN-C, Relaxation, Greedy. The test set makespan distribution is also shown for reference.

GNN variants (on complete graphs only) as discussed in Section VI-A.1. More layers and the usage of edge features have a moderate impact due to the uniformity of the instances obtained from the same area. The best network overall is one with $L = 12$ and edge features, whose error on the test set is shown in Fig. 7. This is used for the planning experiments below. Interestingly, this network was also able to provide reasonable estimates on graphs drawn from a different distribution obtained by arranging a new set of vertices in a triangle shape in the same three areas: the average estimation error never exceeded 16%.

2) Polypixel – Planner Results: We create a set of 25 instances containing 12 areas in total, obtained by randomly drawing 4 different sets of locations for each area. The initial optimality loss and the one obtained at 15 minutes for teams of 25, 35, and 45 robots, with a number of UAVs fixed to 11 (number of areas minus one, as above), are shown in Fig. 8. This time, the GNN-C estimator provides significantly better results than both the MILP relaxation and greedy ones for

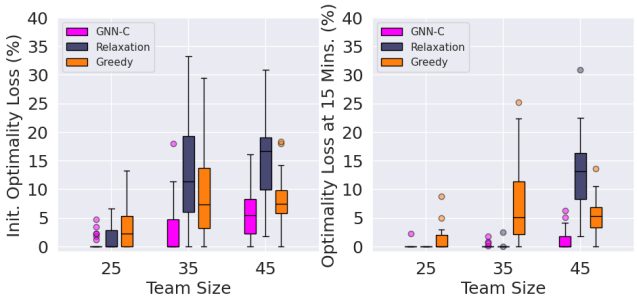

Fig. 8: Polypixel – planner results: varying the team size (25, 35, 45) for a fixed number of areas (12). For each team size, from left to right: GNN-C, Relaxation, Greedy.

the first computed allocation. Differently from the results obtained on grid instances, for teams of 35 and 45 robots the greedy estimator is now showing better performance than the MILP relaxation one. At 15 minutes, both the GNN-C and MILP relaxation estimators provide optimal or near-optimal solutions for teams of 25 and 35 robots. However, for teams of 45 robots, the MILP relaxation estimator still provides solutions whose objectives are more typically more than 10% far from the optimal one, while all the solutions provided by the GNN-C estimator are within 7% of the optimal one. Contrarily to the MILP relaxation and greedy estimators, the GNN-C estimator is able to offer strong performance in both the considered classes of environments. In the Polypixel instances, the GNN-C estimator was also able to provide a solution with an objective within 7% of the optimal one with an approximately 18x speed-up on average w.r.t. the optimal, non-hierarchical baseline.

VII. CONCLUSIONS

We have presented a hierarchical planner for multi-robot task allocation problems, where tasks are defined as routing problems that have to be solved with subteams of robots. We have proposed a GNN architecture able to compute reasonably accurate estimates of allocation costs given by a MILP solution approach applied to a multi-robot heterogeneous area inspection problem. The proposed planner enables the efficient computation of quality plans in routing task allocation problems with up to 45 robots and 20 areas. Our planner can effectively deal with static allocation problems, but is not capable of reasoning about possible temporal constraints between routing tasks. The ability to leverage the same kind of high-level actions in this context remains a promising avenue for future research.

APPENDIX
TASK ALLOCATION MILP DETAILS

Constraints (7)-(8) can be expressed by a combination of three linear constraints with the help of two auxiliary binary variables as follows. Let η, θ be the auxiliary variables, and let c be a constant such that $0 < c \leq 1$. Taking constraint (7) as an example:

$$\alpha^t \geq i\gamma_i^t + (i+c)\eta \tag{13}$$
$$\alpha^t \leq i\gamma_i^t + \min(n, B)\eta + (i-c)\theta \tag{14}$$
$$\gamma_i^t + \eta + \theta = 1. \tag{15}$$

The product enforced by constraint (9) can also be expressed with three linear constraints:

$$\epsilon_{ij}^t \leq \gamma_i^t \tag{16}$$
$$\epsilon_{ij}^t \leq \delta_j^t \tag{17}$$
$$\epsilon_{ij}^t \geq \gamma_i^t + \delta_j^t - 1. \tag{18}$$

REFERENCES

[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int J Robot Res*, vol. 23, no. 9, pp. 939–954, 2004.

[2] E. Balas and M. W. Padberg, "Set partitioning: A survey," *SIAM Review*, vol. 18, no. 4, pp. 710–760, 1976.

[3] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *Int J Robot Res*, vol. 32, no. 12, pp. 1495–1512, 2013.

[4] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[5] E. Feo-Flushing, L. M. Gambardella, and G. A. Di Caro, "Spatially-distributed missions with heterogeneous multi-robot teams," *IEEE Access*, vol. 9, 2021.

[6] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to persistent monitoring tasks," *IEEE T Robot*, vol. 32, no. 5, pp. 1106–1118, 2016.

[7] B. Gilhuly and S. L. Smith, "Robotic coverage for continuous mapping ahead of a moving vehicle," in *Proc. CDC*, 2019, pp. 8224–8229.

[8] T. C. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Multi-robot routing algorithms for robots operating in vineyards," in *Proc. CASE*, 2018, pp. 14–21.

[9] N. Kamra and N. Ayanian, "A mixed integer programming model for timed deliveries in multirobot systems," in *Proc. CASE*, 2015, pp. 612–617.

[10] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE T Robot*, vol. 32, no. 5, pp. 1163–1177, 2016.

[11] P. Oberlin, S. Rathinam, and S. Darbha, "A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem," in *Proc. ACC*, 2009, pp. 1292–1297.

[12] A. Prasad, S. Sundaram, and H.-L. Choi, "Min-max tours for task allocation to heterogeneous agents," in *Proc. CDC*, 2018, pp. 1706–1711.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE T Neur Net Lear*, vol. 32, no. 1, pp. 4–24, 2021.

[14] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[15] Z. Wang and M. Gombolay, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509–4516, 2020.

[16] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[17] X. Bresson and T. Laurent, "An experimental study of neural networks for variable graphs," in *ICLR Workshop Track*, 2018.

[18] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[19] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com