Learning to Allocate Time-Bound and Dynamic Tasks to Multiple Robots using Covariant Attention Neural Networks

Steve Paul
Department of Mechanical and Aerospace Engineering,
University at Buffalo,
Buffalo, NY, 14260, USA
email: stevepau@buffalo.edu

Souma Chowdhury[1]
Professional Member ASME
Department of Mechanical and Aerospace Engineering,
University at Buffalo,
Buffalo, NY, 14260, USA
email: soumacho@buffalo.edu

In various applications of multi-robotics in disaster response, warehouse management, and manufacturing, tasks that are known apriori and tasks added during runtime need to be assigned efficiently and without conflicts to robots in the team. This multi-robot task allocation (MRTA) process presents itself as a combinatorial optimization (CO) problem that is usually challenging to be solved in meaningful timescales using typical (mixed)integer (non)linear programming tools. Building on a growing body of work in using graph reinforcement learning to learn search heuristics for such complex CO problems, this paper presents a new graph neural network architecture called the Covariant Attention Mechanism (CAM). CAM can not only generalize but also scale to larger problems than that encountered in training, and handle dynamic tasks. This architecture combines the concept of Covariant Compositional Networks used here to embed the local structures in task graphs, with a context module that encodes the robots' states. The encoded information is passed onto a decoder designed using Multi-head Attention mechanism. When applied to a class of MRTA problems with time deadlines, robot ferry range constraints, and multi-trip settings, CAM surpasses a state-of-the-art graph learning approach based on the attention mechanism, as well as a feasible random-walk baseline across various generalizability and scalability tests. Performance of CAM is also found to be at par with a high-performing non-learning baseline called BiG-MRTA, while noting up to a 70-fold improvement in decision-making efficiency over this baseline.

Keywords: Multi-Robot Task Allocation, Dynamic Tasks, Reinforcement Learning, Graph Neural Networks

1 Introduction

Multi-robot task allocation (MRTA) is the process of efficiently assigning tasks to autonomous robots, particularly in complex operations with numerous tasks and time constraints. Considerations in MRTA encompass factors such as robot capabilities, task requirements, communication constraints, and overall mission goals. This paper focuses on conflict-free MRTA problems optimizing metrics such as task completion % and overall effort [1]. The work draws motivation from applications with high tasks-to-robot ratios, time-sensitive tasks, and dynamic task emergence, including on-demand last mile delivery [2], disaster response (search and rescue, relief delivery, etc.) [3–6], critical infrastructure inspection or service restoration [7], reconnaissance [8], warehouse management [9], and manufacturing & job shop scheduling [10–12]. MRTA methods can also be applied to transportation fleet planning and coordination of other physical multi-agent systems [13–15]. In such applications, operations with tight task deadlines and dynamic tasks necessitate a fast and generalizable decision-making process, making longer horizon planning suboptimal and potentially risky.

1.1 Conventional methods for MRTA. In this paper, we focus on a specific class of MRTA problems that belong to the Single-task Robots, Single-robot Tasks (SR-ST) category as defined by [1, 16]. According to the iTax taxonomy [1], these problems belong to the In-schedule Dependencies (ID) category. A feasible and conflict-free task allocation assigns each task to only one robot [3]. These MRTA problems can be formulated as Integer Linear Programming (ILP), mixed ILP, or Integer Non-Linear Programming (INLP) problems, depending on the application's criteria functions [17, 18]. When tasks are defined by location, the MRTA problem becomes analogous to the Multi-Traveling Salesmen Problem (mTSP) [19] and its generalized version, the Vehicle Routing Problem (VRP) [20]. Existing solutions to mTSP and VRP [21, 22] address similar characteristics relevant to MRTA, such as tasks with time deadlines and multiple tours per vehicle, with applications in operations research and logistics [23, 24]. ILP-based mTSP formulations and solution methods have been adapted to task allocation problems in the multi-robot domain [25]. Although ILP-based approaches can theoretically provide optimal solutions, the NP-hard nature of SR-ST problems [26, 27] leads to rapidly increasing computational effort as the number of robots and tasks grows [18, 28]. For instance, for the SR-ST problem considered in this paper, the cost of solving the exact integer programming formulation of the problem scales with $\mathcal{O}(n^3m^2h^2)$, where n, m, and h represent the number of tasks, the number of robots, and the maximum number of tours per robot, respectively [3]. As a result, most practical online MRTA methods, including auction-based methods [29, 30], bigraph matching methods [3, 31, 32], and metaheuristic-based methods [33–35], use heuristics for solving the problem. In some of these cases, optimality gaps are reported for smaller test cases by comparing with exact optimization (e.g., INLP) solutions.

Most existing works on task allocation that address dynamic tasks consider a smaller number of tasks, typically ≤ 200, and do not demonstrate adequate scalability. For example, [36] introduces an algorithm for dynamically allocating tasks to multiple agents under time window constraints and task completion uncertainty, however, the maximum number of tasks considered is 200 and provides no evidence for scalability. The Bipartite graph matching algorithm introduced in [3] has also demonstrated its use for dynamic tasks, however, does not scale well computationally for

[1]Corresponding Author.
Version 1.18, July 9, 2024

larger-sized problems. The auction-based approach proposed in [37] reported results for scenarios with less than 50 tasks.

1.2 Learning over Graphs. Recently, Graph Neural Networks (GNN) based learning approaches are getting popular for solving Combinatorial Optimization (CO) problems e.g., TSP, VRP, Max-Cut, Min-Vertex, and MRTA [38–48]. One of the main benefits of learning-base methods over traditional methods (such as metaheuristic algorithms, market-based approaches, graph matching, etc,.) is that learning-based methods can be implemented to newer classes of problems with minimal expert tuning as long as the reward formulation captures the problem's gal effectively and trained properly, whereas the traditional non-learning-based methods require tedious reformulation and redesign of some heuristics.

Unlike normal Neural Networks, GNNs have the innate ability to capture both the Euclidean features (such as location information, capacity, and task deadline), along with categorical features such as task type (if it's heterogeneous tasks), and also has the ability to capture structural information of the task neighborhood. A majority of the existing works on GNNs for CO problems have the following drawbacks: **1)** The problems considered are too simple and exclude common real-world factors such as resource and capacity constraints [38, 40, 41, 44]). **2)** Considers problems with smaller number of tasks/locations and robots/agents ($\leq$ 100 tasks and 10 robots) [45, 49], with their scalability remaining unclear. **3)** Lack of generalization to larger-sized problems (than used for training) without the need to retrain. The third capability is particularly important for a real-world MRTA problem, such as multi-robot flood response, where it is often impossible to always retrain for scenarios of different sizes as compared to the training scenarios, due to the large-training time as well as the computational expense. To address these above-mentioned gaps, we propose a new learning-based framework with the ability to generalize to solve for large-sized MRTA problems (SR-ST) with commonly considered constraints – involving up to 1000+ tasks and 200+ robots – and generalize across even problem scenarios larger than those encountered in training without the need to re-train. Notable related recent developments are discussed below.

The use of sequence-to-sequence models, e.g., pointer networks and attention mechanism, to learn policies for combinatorial optimization problems in graph space using experiences over episodes [38, 41] has become popular over the last few years. Kool et al. [38] implemented an attention mechanism encoder-decoder policy and REINFORCE algorithm for solving a wide variety of combinatorial optimization problems as graphs, with the primary contribution being the approach's flexibility across multiple problems using the same hyperparameters. Wang et al. [50] demonstrated that learning can produce faster solutions than standard exact methods for multi-robot scheduling problems. However, the problem sizes examined in that work and similar studies [45] were limited to 5 robots and 100 tasks, considering only temporal constraints. In this paper, we investigate larger problems, encompassing up to 1000 tasks and 200 robots, and address additional complexities such as task deadlines, robot ferry range constraints, payload capacity constraints, and multiple routes.

We implement a reinforcement learning (RL) approach to generate MRTA solutions, with the main computing effort being on the offline learning part, and the learned policy being deployed in real-time during operation. In order to enable generalizability and scalability to the RL approach, we introduce a new GNN-based policy architecture that combines *attention mechanism* with an enhanced encoding network (embedding layers), where the latter is particularly designed to capture local structural features of the graph in an equivariant manner. The embedding layer is a variation of *Covariant Compositional Networks* (CCN), introduced by [51]. CCN was initially developed to predict molecular properties by learning the local structural information of molecules. The choice of this embedding is inspired because of the following properties: **i)** operates on an undirected graph; **ii)** uses receptive field and aggregation functions based on tensor product and contraction operations, which by virtue of being equivariant leads to a permutation- and rotation-invariant embedding; and **iii)** provides an extendable representation, i.e., an n-th order tensor representation that can extend to multi-level networks (e.g., combining function, motion, and communication of robots or vehicles). Compared to other promising GNN encoders that have been used to promote generalizability in MRTA [47] or related combinatorial optimization problems [52], CCN differs in how it ensures covariance to permutations [51], e.g., in the ordering of nodes in a task graph. We implement a simpler variation of the CCN architecture since the exact original network was found to be computationally burdensome for learning with sequential decision-making. The new proposed policy called Covariant Attention Mechanism (CAM) consists of a CCN-based encoder and an Attention Mechanism-based decoder. We implement the attention-based decoder for CAM as proposed in [38, 53].

1.3 Contributions. The primary contributions of this paper can be summarized as: **1)** Formulating the SR-ST class of MRTA problems with a dynamic task space and robots with range and payload constraints as a Markov Decision Process or MDP over graphs with the multi-robot state information embedded as the *context* portion of the policy model, such that the policy for selecting tasks can be learned using an RL approach; this is motivated by our prior work in [47, 54]. **2)** Design a GNN-based policy network with an *encoder-decoder* architecture, with the *encoder* being based on covariant compositional networks (CCN), whose embedding capabilities significantly improve generalizability and scalability to larger task graphs and multi-robot teams. **3)** An attention-based decoder (inspired by [38, 47]) for sequential decision-making, and specifically extend it to a multi-agent CO setting.

The proposed CAM architecture is evaluated on a representative MRTA problem that involves coordinating a team of unmanned aerial vehicles (UAVs) to time-efficiently deliver flood relief. *In this regard, the capability to handle dynamics tasks, where new tasks that appear while the operation is ongoing, thus leading to a dynamic task graph, is a major new milestone compared to other related architectures in our prior work [47].* The results of this case study demonstrate how CAM outperforms the state-of-the-art attention-based method AM [38], in terms of scalability and convergence, thereby emphasizing the encoder's effectiveness.

The remainder of the paper is organized as follows: Section 2 defines the MRTA problem and its formulation as an MDP over graphs. Section 3 presents our proposed new GNN architecture. Section 4 describes simulation settings and different case studies. Results are discussed in Section 5.

2 MRTA: Problem Definition and Formulations

The MRTA problem is defined as the allocation of tasks and resources among several robots that act together without conflict in the same environment to accomplish a common mission. The optimum solution (decision) of the MRTA problem is a sequence of tasks for each robot that do not conflict with each other and maximizes the mission outcome (e.g., the fraction of tasks completed) or minimizes the mission cost (e.g., total distance traveled), subject to the robots' range and capacity constraints. Multiple trips are allowed for each robot, with intervening visits to the depot where they fully regain capacity (payloads) and range (battery). In our prototypical application, every task has a location and requires a relief package to be delivered by a robot. We consider the packages to be of the same size and weight, and every robot has a maximum payload capacity, i.e., the number of packages that can be carried.

Here, the following assumptions are made: 1) All robots are identical and start/end at the same depot; 2) There are no environmental uncertainties; 3) The location (x_i, y_i) of task-i and its time deadline τ_i are known to all robots; 4) Each robot can share its state and its world view with other robots; and 5) There is a depot (Task-0), where each robot starts from and visits if no other tasks are feasible to be undertaken due to the lack of available range or lack of payloads, or when there are no active tasks remaining; at the depot, robots recharge instantaneously via battery swap and

reload packages to full capacity. Each tour is defined as departing from the depot, undertaking at least one task, and returning to the depot; 6) Motivated by the multi-UAV relief delivery problem, tasks are considered to be instantaneous, which means that reaching the waypoint associated with a task completes that task; 7) The shared workspace (environment) used by the robots is deterministic and fairly large compared to the robot body length, hence lower-level motion planning (including collision avoidance) is straight-forward – the cost of travel between any two locations implicitly accounts for this, without using an explicit motion planning effort in this paper; moreover, time of arrival at any location becomes a simple exact computation.

Subject to these assumptions, the MRTA problem is considered to be a type of combinatorial optimization problem that can be modeled in graph space. The optimization formulation of MRTA is discussed in Section 2.1. In order to learn policies that yield solutions to this CO problem, we express the MRTA problem as a Markov Decision Process (MDP) over a graph in section 2.2.

2.1 MRTA as Optimization Problem. This MRTA problem is adopted from [3]. The exact solution to the MRTA problem can be obtained by formulating it as an integer nonlinear programming (INLP) problem. For a problem with N tasks and N_r robots, the objective of this INLP can be summarily expressed as:

$$\min\ f_{\text{cost}} = \psi - u(\psi)e^{-d_\psi} \tag{1}$$

$$N_{\text{success}} = \sum_{i\in[1,N]} \eta_i \begin{cases} \eta_i = 1, & \text{if } \tau_i^{\text{completed}} \le \tau_i \\ \eta_i = 0, & \text{otherwise} \end{cases}$$

$$\psi = (N - N_{\text{success}})\,/N$$

$$u(\psi) = \begin{cases} 1 & \text{if } \psi = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$d_\psi = \sum_{i=1}^{N_r} d_i^{\text{total}}/(\sqrt{2}\,N)$$

Here, τ_i is the time deadline of task i, $\tau_i^{\text{completed}}$ is the time at which task i is completed, and they are used to compute the number of tasks completed in a mission, namely N_{success}. The term d_i^{total} represents the total traveled distance by robot-i during the entire mission (could involve multiple trips by the robot).

The term d_ψ is the scaled averaged distance traveled by the robots in the team over the mission, where the division by $\sqrt{2}N$ serves as the scaling and averaging factor; here $\sqrt{2}$ represents the maximum possible distance between any two points in a normalized 1×1 area where tasks and depot are located. Here, we craft the objective function to be minimized, f_{cost} (Eq. 1), such that it emphasizes maximizing the task completion %, i.e., 100× the number of tasks completed divided by the total number of tasks in that scenario. This objective function varies in the range $(-1, 1]$; the function $f_{\text{cost}} < 0$, if all tasks are completed, and $f_{\text{cost}} > 0$, if the completion rate is lower than 100%. The second term in f_{cost} is active only in the former case, and accounts for the scaled travel cost of the robots.

The primary constraints considered in this INLP formulation of the concerned class of MRTA problems include: 1) conflict resolution, where no two robots can choose the same task; 2) the total distance traveled by any robot during a trip should be less than its maximum range subject to battery capacity; and 3) the number of tasks selected/completed by a robot during a trip should be less than or equal to it maximum payload capacity $C_{\max}$. A detailed mathematical formulation of these ILP constraints and related others can be found in [3]. Note that, here we use a slightly different objective/cost function (one that is nonlinear) compared to that in [3] to better reflect the generalized setting for the class of MRTA problems with ferry range, payload capacity, and task-deadline constraints.

2.2 MDP over a Graph. Building on the formulation in [47], the MRTA problem is considered to involve a set of nodes/vertices (V) and a set of edges (E) that connect the vertices to each other, which can be represented as a complete graph $\mathcal{G} = (V, E)$. Each node represents a task, and each edge connects a pair of nodes. Let Ω be a weight matrix where the weight of the edge ($\omega_{ij} \in \Omega$) represents the cost (e.g., distance traveled) incurred by a robot to take task-j after completing task-i. For MRTA with N tasks, the number of vertices and the number of edges are respectively equal to N and $N(N-1)/2$. Node i is assigned a 3-dimensional feature vector denoting the task location (x_i, y_i) and time deadline, i.e., $\delta_i = [x_i, y_i, \tau_i]$ where $i \in [1, N]$. Here, ω_{ij} can be computed as $\omega_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, where $i, j \in [1, N]$.

The MDP is defined in a decentralized manner for each individual robot, to capture its task selection process. This can be expressed as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}_a, \mathcal{R} >$. The components of the MDP can be defined as **State Space ($\mathcal{S}$):** A robot at its decision-making instance uses a state $s \in \mathcal{S}$, which contains the following information: 1) the current mission time, 2) its current location, 3) its remaining ferry-range (battery state), 4) the number of packages it is carrying, 5) the planned (allocated) tasks of its peers, 6) the remaining ferry-range of its peers, 7) the number of packages each peer robot is carrying, and 8) the state of the task space. A task is considered active if its deadline has not passed and if the task is not completed. The state of tasks contain the location, the time deadline, and the task status – *active* (not selected yet and deadline hasn't passed), *completed* (already selected or completed by a robot), and *missed* (i.e., the deadline has passed, without ever being selected). Here we assume that each robot can broadcast its information to its peers without the need for a centralized system for communication, as aligned with modern communication capabilities [45]. **Action Space ($\mathcal{A}$):** The set of actions is represented as $\mathcal{A}$, where each action a is defined as the index of the selected task, $\{0, \ldots, N\}$ with the index of the depot as 0. The task 0 (the depot) can be selected by multiple robots, but the other tasks are allowed to be chosen once if they are active (not completed or missed tasks). **$\mathcal{P}_a(s'|s, a)$:** A robot by taking action a at state s reaches the next state s' in a deterministic manner (i.e., deterministic transition model is defined). **Reward ($\mathcal{R}$):** The reward function is defined as $-f_{\text{cost}}$ and is calculated when there are no more active tasks, meaning all tasks have either been completed or their deadline has passed. **Transition:** The transition is an event-based trigger conditioned on a robot reaching its selected task or the depot (if selected). Unlike the scenarios with static tasks, the state transition in scenarios with dynamically generated tasks is stochastic – this is due to the possibility of new tasks to randomly appear during the mission. If $\mathcal{T}_{\text{active}}^t$ is the set of active tasks at time step t and $\mathcal{T}_{\text{active}}^{t+1}$ is the set of active tasks at time step $t+1$, then $\mathcal{T}_{\text{active}}^{t+1} \not\subset \mathcal{T}_{\text{active}}^t$.

3 Covariant Attention-based Neural Architecture

To operate on the MDP defined over graphs in Section 2.2, the policy architecture needs to represent each node as a continuous vector, preserving its properties as well as the structural information of the neighborhood of that node.

Before describing the technical components of our proposed Covariant Attention Mechanism, the so-called CAM neural architecture, we provide an illustration (Fig. 1) and a summary description here of how this policy architecture is used by robots or agents during an SR-ST operation. The CAM model for task allocation is executed by each robot just when it reaches its current destination (task location or depot), in order to decide its next task or destination. Since full observability is assumed across the multi-robot team and the policy-model execution time is almost negligible, the current setup is agnostic to whether the online CAM model is executed

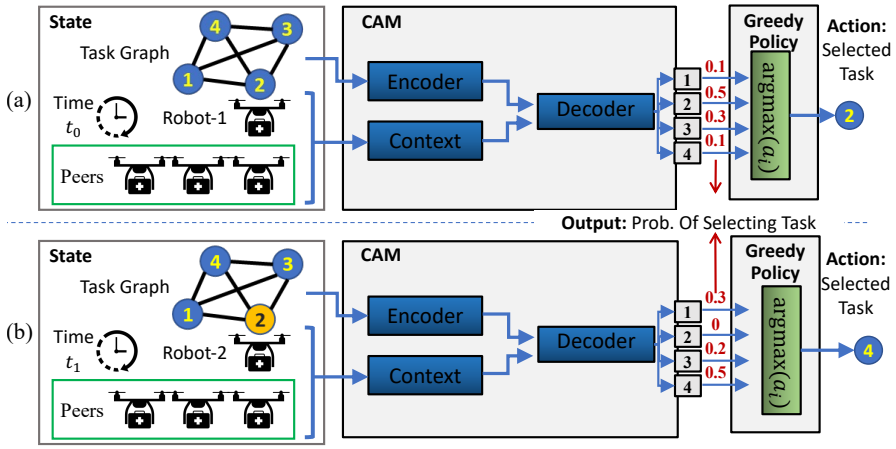


Fig. 1 Deployment of an MRTA policy using CAM architecture. a) Robot-1 at t_0. b) Robot-2 at t_1; here, the CAM output for the previously selected task (task 2 in (b)) is set as 0.

centrally off-board or on-board each robot. As an example, Figure 1 illustrates how a robot-1 and robot-2 use the CAM policy model to choose a task at two different decision-making instances ($t = t_0$ and $t = t_1$). Here, the inputs to the CAM model include **1)** the task graph information, i.e., the properties of all the tasks/nodes d_i and the computed weight matrix Ω, **2)** the current mission time, **3)** the state of robot-r, and **4)** the states of robot-r's peers. The CAM model then generates the probability of selecting each available task as its output. A greedy strategy of choosing the task with the highest probability is used here, which thus provides the next destination to be visited by that robot. It should be noted that the probability values for completed tasks and missed tasks (i.e., whose deadlines pass before being selected by any robots) are set at 0.

Figure 2 shows the detailed architecture of CAM. As shown in this figure, the CAM model consists of three key components, namely: ***Context***, ***Encoder***, and ***Decoder***. The context includes the current mission time, the states of robot-r, and the states of robot-r's peers. The state of a robot consists of its destination x, y coordinates and the available range ρ. The encoder and decoder components are further described below.

3.1 CCN-inspired Node Encoder. For learning over graphs, the performance of the trained model depends mostly on the ability of the Graph Neural Network (GNN) to transform all the required node information into a feature vector or tensor. For our case, apart from the node properties, some of the features that are essential include a node's local neighborhood information and permutation invariance. The latter ensures insensitivity to the indexing of tasks or nodes. Using a node's local neighborhood information which consists of its association with its local neighbors during training is more beneficial than considering the association with the entire graph, for generalizing to unseen nodes as demonstrated by frameworks such as GraphSAGE [55], thus enabling the GNN to generalize for problems with a larger number of nodes without the need to re-train. The encoder represents the properties of each graph node (preserving its structural information) into a continuous feature vector of dimension d_{embed}, which is fed to the decoder. Each node i here has three properties which are the x-coordinate (x), y-coordinate (y), and the time deadline (τ) of the task. The proposed formulation and architecture can however be readily applied to problem scenarios with a greater number of task node properties. Note that our encoding mechanism can also be extended to a probabilistic scenario, for example where an estimated deadline τ follows a probability distribution, which is likely in a disaster response type operation. The encoding for each node should include its properties and its positional association with its neighboring nodes.

We implement a simpler variation of CCN [51]. We determine the nearest k neighbors of a node i, expressed as $\mathcal{N}_i$ – this is also called the receptive field of node i based on the positional coordinates (x and y). The first step is to compute a feature vector by linear transformation for each node i. To encode the node properties, we do a linear transformation of d_i to get a feature vector F_{d_i} for all $i \in [1, N]$, i.e., $F_{d_i} = W^d d_i^T + b_d$.

Here $W^d \in \mathbb{R}^{d_{\text{embed}} \times 3}$, is the weight matrix, $b_d \in \mathbb{R}^{d_{\text{embed}} \times 1}$ is the bias vector, and $\delta_i = [x_i, y_i, \tau_i]$. For effective decision-making, we also need to preserve the structural information. Therefore we define a matrix $F_{d_i}^{\mathcal{N}}$ as given below.

$$F_{d_i}^{\mathcal{N}} = \text{Concat}(F_{d_j}), \quad j \in \mathcal{N}_i \tag{2}$$

where the Concat operation concatenates all the F_{d_j} vectors ($\forall j \in \mathcal{N}_i$) into a matrix of size $|\mathcal{N}_i| \times d_{\text{embed}}$, with $||$ being the cardinality operator. We compute a matrix $F_i^{\mathcal{N}}$ (as shown in Eq. 3), which we believe captures the association of a node with its local neighbors (one-hop neighbors) in terms of the node properties. This is equivalent to the message passing operation in [51].

$$F_i^{\mathcal{N}} = W^{\mathcal{N}}(F_{d_i}^{\mathcal{N}} - F_{d_i}) + b^{\mathcal{N}} \tag{3}$$

Here $W^{\mathcal{N}} \in \mathbb{R}^{d_{\text{embed}} \times d_{\text{embed}}}$, $b^{\mathcal{N}} \in \mathbb{R}^{d_{\text{embed}} \times 1}$. $F_i^{\mathcal{N}}$ captures the similarity of node properties of neighboring nodes ($\mathcal{N}$) to that of node i. For every node, its initial embedding and the neighborhood node embeddings are concatenated, similar to the tensor stacking operation in [51]. This concatenation is followed by an aggregation operation, which results in the final node embeddings, as given by

$$F_i = \text{Aggregate}(W_f(\text{Concat}(F_i^d, F_i^{\mathcal{N}})) + b_f) \tag{4}$$

Here, $W_f \in \mathbb{R}^{d_{\text{embed}} \times d_{\text{embed}}}$, $b_f \in \mathbb{R}^{d_{\text{embed}} \times 1}$. Thus finally we get an embedding F_i for each node, where $F_i \in \mathbb{R}^{d_{\text{embed}} \times 1}$. Here, W^d, b_d, $W^{\mathcal{N}}$, $b^{\mathcal{N}}$, W_f, and b_f are learnable weights and biases. The Aggregate function is the summation across all the columns of a matrix. This summation along with the relative difference in node properties, as in Eq. 3, preserves permutation-invariance and the structural properties of the graph. Structural properties refer to node similarities (or property closeness) in neighborhoods of the graph. Note that, these operations make the encoded state w.r.t. a given node insensitive to the order of the neighboring nodes, and thus the overall state space becomes independent of the indexing of tasks or to graph rotations (the latter promotes generalizability). Equations 2, 3, and 4 represent a single layer of encoding. Multiple layers of encoding can be performed with the output of the previous layer being the inputs to equations 2 and 3

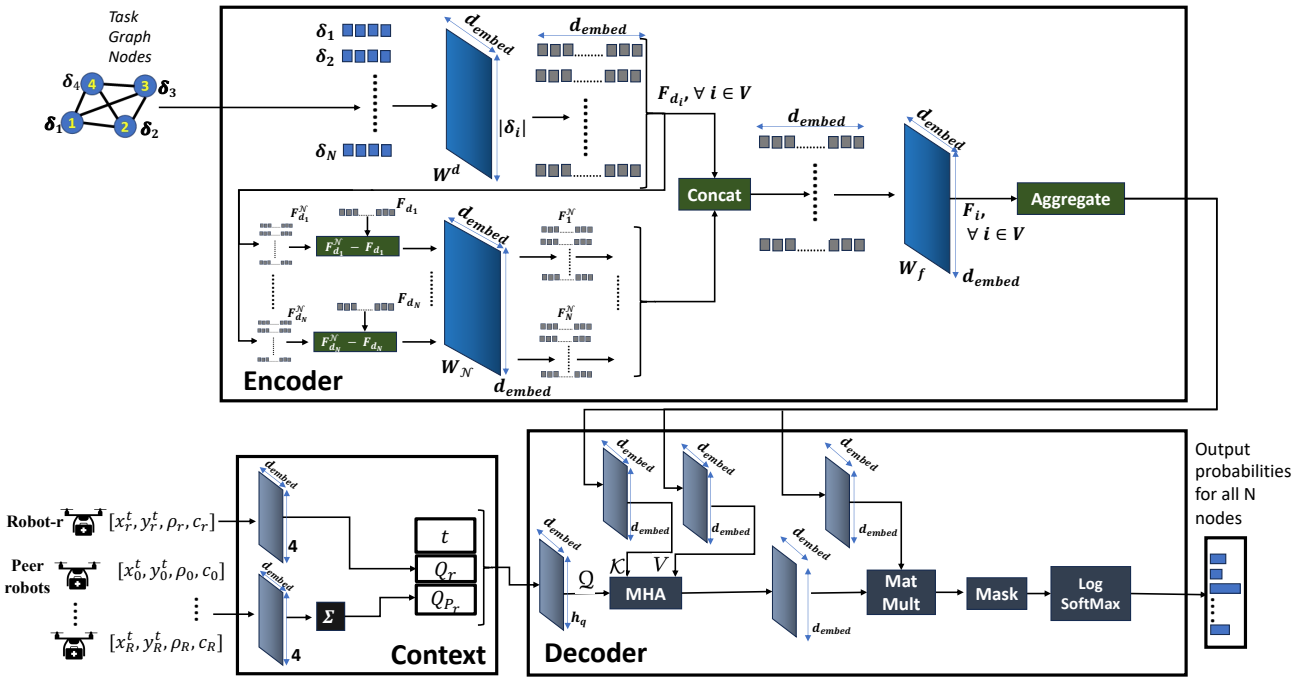


Fig. 2 CAM architecture with the information flow along the context, encoder & decoder. The robot r is the robot taking decision. The biases are omitted for ease of representation.

in the next layer. More layers can be beneficial when learning the structure of larger task graphs; however, this comes at the cost of a greater number of training parameters often leading to greater training costs and memory requirements.

Compared to the CapAM method introduced in our prior work on MRTA [47], a key difference in the policy architecture is the choice of encoder in this paper. In CapAM, the encoder is based on Graph Capsule convolutional Networks (GCAPCN) [56], while in CAM here, the encoder is based on Covariant Compositional Networks (CCN) [51]. Both types of encoder networks perform the two fundamental GNN operations which are message passing and aggregation. In GCAPCN, the message passing is performed by matrix multiplication of the graph Laplacian with the feature matrix, and aggregation is based on a convolution operation. In contrast, with CCN, the message passing is based on the relative feature difference of the higher dimensional projections of a node and its neighboring nodes, which is premised to improve the steerability of node activation [51]; here aggregation is a simple summation.

3.2 Attention-based Decoder. The main objective of the decoder is to use the information from the encoder and the current state as context or query, and thereof choose the best task by calculating the probability value of getting selected for each (task) node. In this case, the first step is to feed the embedding for each node computed by the encoder as **key-values** $(\mathcal{K}, \mathcal{V})$. The key $\mathcal{K}$ and value $\mathcal{V}$ for each node are computed by two separate linear transformations of the node embedding obtained from the encoder. The next step is to compute a vector representing the current state, also known as the **context** (as shown in the bottom left of Fig. 2). The context for the multi-head attention (MHA) layer in this experiment consists of the following seven features: 1) Time elapsed; 2) Available range of the robot taking decision; 3) Current location of robot taking decision; 4) Current destination of other robots; and 5) Available range for other robots. These context parameters are all concatenated to a single vector of length h_q, which then undergoes a linear transformation to get a vector of length d_{embed} also called the query $\mathcal{Q}$. Figure 2 illustrates the structure of the decoder.

Now the attention mechanism can be described as mapping the query $(\mathcal{Q})$ to a set of key-value $(\mathcal{K}, \mathcal{V})$ pairs. The inputs, which are the query $(\mathcal{Q})$ is a vector, while $\mathcal{K}$ and $\mathcal{V}$ are matrices of size $d_{\text{embed}} \times N$ (since there are N nodes). The output is a weighted sum of the values $\mathcal{V}$, with the weight vector computed using the compatibility function expressed as:

$$\text{Attention}(\mathcal{K}, \mathcal{V}, \mathcal{Q}) = \text{softmax}(\mathcal{Q}^T \mathcal{K} / \sqrt{d_{\text{embed}}}) \mathcal{V}^T \tag{5}$$

Here h_l is the dimension of the key of any node i ($k_i \in \mathcal{K}$). In this work, we implement a multi-head attention (MHA) layer in order to determine the compatibility of $\mathcal{Q}$ with $\mathcal{K}$ and $\mathcal{V}$. The MHA implemented in this work is similar to the decoder implemented in [38, 47, 53]. As given in [53], the MHA layer can be defined as:

$$\text{MHA}(\mathcal{K}, \mathcal{V}, \mathcal{Q}) = W_{\text{MHA}} \times (\text{Concat}(\text{head}_1 \ldots \text{head}_{h_e})) + b_{\text{MHA}} \tag{6}$$

where the Concat operator concatenates the attention heads into a single vector. Here, W_{MHA} and b_{MHA} are respectively the learnable weights and biases for linear transformation of the concatenated attention heads.

Here $\text{head}_i = \text{Attention}(\mathcal{K}, \mathcal{V}, \mathcal{Q})$ and h_e is the number of heads, which is set as 8 in this paper.

The output from MHA undergoes a linear transformation (Fig. 2) which is then multiplied with another linear transformation of the node embeddings, resulting in the *logits* for the nodes. Based on the *logits*, the final `logsoftmax` layer outputs the probability values $(p_i,\ i = 1, 2, \ldots, N)$ for all the nodes. Here, the next task to be done by the given robot is then chosen based on a greedy approach – the node with the highest probability is chosen, i.e., select j, where $p_j \geq p_i\ \forall\ i \neq j$. During the decision-making instance of a given robot, out of all the available tasks, we mask the infeasible tasks by setting their probability as 0, i.e., $p_i = 0$, if $i \in \mathbf{T}_{\text{infeas}}$; this is achieved by setting logits of the masked nodes as $-\infty$. Particular to the given robot taking decision, the set of infeasible tasks ($\mathbf{T}_{\text{infeas}}$) includes: 1) any task that has already been selected and/or completed by a robot in the team, aka inactive tasks; 2) any task that was never selected by any robot but whose deadline has passed, aka missed tasks; 3) any active task that has

not been selected/completed but whose deadline will pass before the given robot can reach that task; and 4) any task whose distance from the current location of the given robot and distance from the depot sums up to a value greater than the remaining range of the given robot (related to its remaining battery). Note that only in the case of a deciding robot that is currently at the depot, we also mask the depot by setting its probability to zero; in all other cases, the probability of choosing the depot is > 0.

3.3 Context. As discussed in Section 3.2, for a robot r deciding its next task, the context portion consists of 1) Time elapsed t; 2) Available range of the robot taking decision ρ_r; 3) The current number of packages carried by the robot c_i; 5) Current location of robot taking decision (x_r, y_r); 6) Current destination of other robots $(x_p, y_p, \forall\ p \in P_r)$; 7) Available range for other robots $(\rho_p, \forall\ p \in P_r)$; 8) The current number of packages carried by each of the peers $(c_p, \forall\ p \in P_r)$, where P_r represents the peers of robot r. Here, features 2, 3, and 4 represent the current state of the robot taking the decision, while features 5, 6, and 7 represent the states of the peer robots. The context feature vector can be computed as

$$\mathcal{Q} = W_{\mathcal{Q}} \times \text{Concat}(t, Q_r, Q_{P_r}) + b_{\mathcal{Q}} \tag{7}$$

where,

$$Q_r = W_{Q_r} \times ([x_r, y_r, \rho_r]) + b_{Q_r} \tag{8}$$

and

$$Q_{P_r} = \Sigma_{p \in P_r} W_{P_r} \times ([x_p, y_p, \rho_p]) + b_{P_r} \tag{9}$$

In the above equations, Q_r, and Q_{P_r} are learnable vectors respectively representing the state of the robot taking decision and the state of the peer robots. The 'Concat' operation in Eq. 7 concatenates t, Q_r and Q_{P_r} into a single vector. Here, $W_{\mathcal{Q}}$, W_{Q_r} and W_{P_r} are the respective learnable weights of the linear transformations, while $b_{\mathcal{Q}}$, b_{Q_r}, and b_{P_r} are the respective biases. The dimensions of Q_r and Q_{P_r} is d_{embed}, and the length of the final feature vector $\mathcal{Q}$ is also defined as d_{embed}. The summation aggregation operation in Eq. 9, makes the context vector agnostic to the number of robots.

3.4 Learning Framework. The CCN-inspired encoder and attention-based decoder, detailed in Sections 3.1 and 3.2, utilize learnable weight matrices. Supervised learning methods for matrix learning are impractical due to the high computational complexity of the exact I(N)LP solution process, scaling with $O(n^3m^2h^2)$ for the ILP formulation of the MRTA problem. To address this, we employ a reinforcement learning algorithm. This work utilizes a simple policy gradient method (REINFORCE) with a greedy rollout baseline, facilitating a comparison with [38]. Each epoch involves a training set and a validation set. The training set used for training the model θ_{CAM} and the validation set used for updating the baseline model θ^{BL}_{CAM} are described in Section 4.1. Samples from these sets consist of graphs as defined in Section 2.2.

Algorithm 1 outlines the training algorithm. In algorithm 1, the function GenerateScenarios$(N_{\text{tr}}, N_{\text{vl}})$ generates a set of N_{tr} training samples (denoted as $\mathcal{D}_{\text{tr}}$) and a set of N_{vl} validation samples (denoted as $\mathcal{D}_{\text{vl}}$). Each sample is randomized in terms of the properties of the tasks, namely location and time deadline. The location and deadlines are randomly sampled following a uniform distribution within bounds that are defined in Section 4.1). For scenarios with dynamically generated tasks, we consider a birth time for the tasks that are sampled following a distribution defined later in Section 5.3. The function SampleBatch$(\mathcal{D}_{\text{tr}}, B)$ picks a random selection of B samples out of the set $\mathcal{D}_{\text{tr}}$. The function Simulation$(\theta, \mathcal{D}_i)$ simulates the generic i-th task scenario $\mathcal{D}_i$ with the robots using the policy θ, and outputs the total cost of the scenario and the actions taken by the robots.

The policy, defined in accordance with Eq. 1, guides the behavior of robot r – returning to the depot ($a = 0$) if constraints aren't met, or selecting the task with the highest probability using the CAM network in a greedy approach, as depicted in Fig. 1.

Algorithm 1 Training Algorithm

Input: N_{E}: Number of epochs, N_b: Number of batches, B: Batch size, N_{tr}: Training data size, N_{vl}: Validation data size.

$\theta_{\text{CAM-RL}}$ - CAM-RL
$\theta^{BL}_{\text{CAM-RL}}$ - Baseline CAM-RL
for epoch = 1..N_{epoch} **do**
  $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{vl}} \leftarrow$ GenerateScenarios$(N_{\text{tr}}, N_{\text{vl}})$
  $N_b \leftarrow \lfloor N_{\text{tr}}/B \rfloor$
  for step = 1..N_b **do**
    $\mathcal{D}_{\text{tr,b}} \leftarrow$ SampleBatch$(\mathcal{D}_{\text{tr}}, B)$ {$\mathcal{D}_{\text{tr,b}}$: Batch Training Dataset}
    $\mathbf{a}^{\text{BL}}$, $f^{\text{BL}}_{\text{cost}} \leftarrow$ CalculateCost$(\theta^{BL}_{\text{CAM-RL}}, \mathcal{D}_{\text{tr,b}})$
    $\mathbf{a}$, $f_{\text{cost}} \leftarrow$ CalculateCost$(\theta_{\text{CAM-RL}}, \mathcal{D}_{\text{tr,b}})$
    $\nabla\mathcal{L} \leftarrow \frac{1}{B}\sum_{i=1}^{B}(f_{\text{cost,i}} - f^{\text{BL}}_{\text{cost,i}}) \log \text{softmax}(\mathbf{a}_i)$
    $\theta_{\text{CAM-RL}} \leftarrow$ Adam$(\nabla\mathcal{L}, \theta_{\text{CAM-RL}})$
  end for
  $\mathbf{a}^{\text{BL}}_{\text{vl}}$, $f^{\text{BL}}_{\text{cost,vl}} \leftarrow$ CalculateCost$(\theta^{BL}_{\text{CAM-RL}}, \mathcal{D}_{\text{vl}})$
  $\mathbf{a}$, $f_{\text{cost}} \leftarrow$ CalculateCost$(\theta_{\text{CAM-RL}}, \mathcal{D}_{\text{vl}})$
  if $(\sum_{i=1}^{N_{\text{vl}}} f^{\text{BL}}_{\text{cost,i}} > \sum_{i=1}^{N_{\text{vl}}} f_{\text{cost,i}}) \wedge$ (T-Test$(\mathbf{a}_{\text{vl}}, \mathbf{a}^{\text{BL}}_{\text{vl}}) > \epsilon$) **then**
    $\theta^{BL}_{\text{CAM-RL}} \leftarrow \theta_{\text{CAM-RL}}$
  end if
end for
CalcuateCost Procedure:
for i = 1..$|\mathcal{D}|$ **do**
  $\mathbf{a}_i, f_{\text{cost,i}} \leftarrow$ Simulation$(\theta, \mathcal{D}_i)$
  $\mathbf{a} \leftarrow \mathbf{a} \cup \mathbf{a}_i$
  $f_{\text{cost}} \leftarrow f_{\text{cost}} \cup f_{\text{cost,i}}$
end for
return $\mathbf{a}$, f_{cost}

3.4.1 Simulation and Framework Settings. The *"Python"* 3.7 and the 64-bit distribution of *"Anaconda 2020.02"* are used to implement the MRTA approaches. The environment, training algorithm, and the evaluation of the trained model, are all implemented in *Pytorch-1.5* for CAM and baselines. The training based on Pytorch is deployed on two GPUs (NVIDIA Tesla V100) with 16GB RAM.

4 Case Studies and Competing Approaches

We design and execute a set of numerical experiments, which is further described in sub-section 4.1, to investigate the performance of our proposed learning-based algorithm over graph space (*CAM*) and compare it against four different approaches: **1)** an extended version of a state-of-the-art graph learning-based algorithm proposed by [38], so-called *attention-based mechanism* (*AM*) approach; **2)** a recent bipartite graph matching approach called BiG-MRTA [3], which has been shown to outperform other online methods for solving the concerned class of MRTA problems; **3)** a myopic baseline called Feasibility-preserving Random-Walk (Feas-RND) that takes randomized but feasible actions, i.e., avoiding conflicts and satisfying other problem constraints [3]; and **4)** a Integer Non-linear Programming solver that is suitable for offline application, and used here to compute the optimality gaps over the smaller problems (involving 50 tasks). The Feas-RND method, on the other hand, provides a baseline that AM and CAM should clearly surpass in performance (cost function), in order to demonstrate that meaningful task selection policies are being learned, as opposed to simply mapping random feasible actions. The comparative approaches are described in more detail later in this section. The codes for this paper as well as the supplementary materials (Appendices A, B, C, and D) can be found in [57].

4.1 Design of Experiments & Learning Procedures. To train the proposed CAM method, we define an MRTA case study with varying numbers of UAVs and 200 tasks, namely flood victim locations to be served. A synthetic 2D environment with 1 sq. km area is used for this purpose, with the time deadline of tasks varied from 0.1 to 1 hour. The UAVs are assumed to have a range of $\Delta_k = 4$ km, a payload capacity of 5 packages, i.e., $C_{\text{max=5}}$, and a nominal speed of 10 km/h. Results obtained on this problem settings would readily scale to larger areas encountered in real-world settings, as long as UAV's nominal speed and ranges are proportionately scaled (e.g., most Group 1, $<$ 20 lb, UAVs can typically fly above 100 km/hr [58]). We assume an instantaneous battery swap is provided at the depot location, which is used when UAVs return to the depot since they are running low on battery.

It is important to note that the flood victim application is used here merely for motivation, and the CAM architecture is in no way restricted to this application, but can rather solve problems in the broad (important) class of capacity/range-constrained and timed task-constrained SR-ST problems. Moreover, even the policies learned here for CAM demonstration on the described case settings can generalize to related SR-ST problems with up to 1,000 tasks, which represents a fairly large MRTA problem domain in reference to the related literature in the multi-robotics domain.

To perform training and testing of the learned model, we proceed as follows: *Training Phase:* We use a policy gradient reinforcement learning algorithm REINFORCE with rollout baselines for learning the optimal policy. The learnable parameters in this architecture include all the weights in the encoder and the decoder. The training is carried out for a total of 100 epochs. Each epoch consists of 10,000 random training samples, which are evaluated and trained in batches of 100 samples.

Testing Phase: In order to provide a statistically insightful evaluation and comparison, the methods are tested on different cases involving varying numbers of tasks and robots, with each case having 100 random test scenarios. The randomization of the task properties across samples or scenarios is drawn from the same uniform probability distribution for both the training and testing phases. The training and test settings and the modifications to AM for MRTA are given in Section 5.1. We further implement the trained model in scenarios where tasks are dynamically generated. By this, we demonstrate the ability of our method to be trained on simple static scenarios/tasks, with the capability to be implemented on a more complex mission where scenarios/tasks are dynamically generated during the mission.

4.2 Baselines. BiG-MRTA: The BiG-MRTA algorithm [3] is an online method based on the construction and maximum weighted matching of a bipartite graph. BiG-MRTA [3] uses a novel combination of bipartite graph construction, an incentive model to assign edge weights in the bigraph, and maximum weighted matching (based on the Karp algorithm [59]) over the bigraph to allocate tasks to robots. This method has been developed as an online solver for SR-ST type MRTA problems, where tasks have deadlines, new tasks could appear during the mission, and robots are subject to range and payload constraints.

AM: The attention-based mechanism (AM) reported by [38] has been shown to solve a few different classes of single-agent/robot combinatorial optimization problems. To be able to implement the AM method for the MRTA problems studied here, the AM method is adapted as i) The node properties that are defined in Section 2.2 are used in AM; ii) The context for the attention mechanism is modified to be the same as that used for CAM; and iii) The cost function used for training is changed to that in Eq. 1. The number of attention heads for the encoder is set at 8, with 3 layers of encoding. The node embedding length is set at 128.

Integer Non-Linear Programming: The MRTA problem is formulated as an Integer Non-Linear Programming (INLP), based on [3]. Unlike the other baseline methods, the INLP solution (exact optimization) will consist of the sequence of tasks assigned to each robot at the start of the mission, given a scenario, without including any dynamic tasks (since the optimal assignment then needs to be re-done). The complete formulation consisting of the objective function and the constraints are discussed in the Appendix A in [57]. We use the Gurobi solver in Python to run the optimization, which uses the Branch and Bound method for solving INLP. The INLP solutions are generated only for scenarios with the number of tasks $N = 50$, since larger problems become computationally intractable for the exact INLP process. To generate the solution for every scenario in the $N = 50$ case, a run time of 25 mins is allowed to INLP. It is possible that in some cases, the INLP does not converge to an optimal solution, or even fails to find a feasible solution, within this allowed run-time.

5 Results and Discussion

In this section, we first analyze the training convergence for CAM and AM, followed by the generalizability analyses, scalability analyses, and computing time analysis. We then perform an ablation study on the encoder and the decoder of CAM to highlight their contributions. These stated analyses are all performed on problems with only static tasks fixed at the start of the mission. Subsequently, we implement the trained CAM and AM models in scenarios with dynamically generated tasks. All of the above analyses are backed by statistical tests to establish (at least 5%) statistical significance in comparisons of the performance of CAM to the baseline methods.

5.1 Learning Curve. In order to compare the convergence of the proposed CAM method with that of the AM approach, we run both methods with similar settings as given in Table 1, and plot their learning curve (convergence history), which is shown in Fig. 3. As seen from this figure, the AM method's cost function stagnates after 14 epochs. On the other hand, the CAM method experienced a steady improvement in its cost function value throughout the training period until epoch 91. The CAM method converges to a cost function value of -0.330 compared to a much poorer cost function value of 0.108 achieved by AM. This could be attributed to the direct implementation of the transformer network [53], which was designed for machine translation and thus consists of multiple layers of Multi-head attention. In contrast, our CAM model uses simple linear transformations of the node properties and their relative differences in local neighborhoods to better capture structural information.

Table 1 Training algorithm settings for CAM and AM.

Details	Vale
Algorithm	*REINFORCE*
Baseline	*Rollout*
Epochs	100
# of tasks	200
Training samples	10000
Baseline samples	1000
Optimizer	*Adam*
Learning step size	0.0001
Training frequency	100 samples

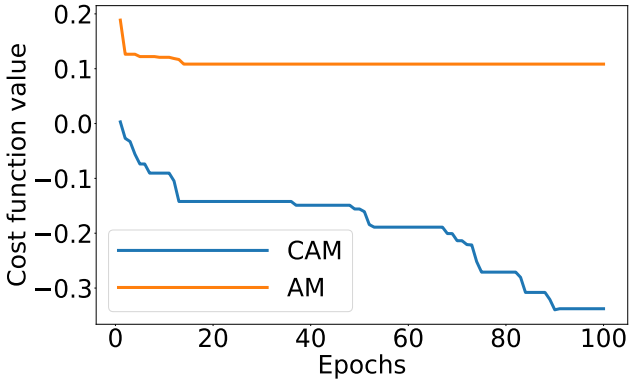


Fig. 3 Training history of CAM and AM for 200 tasks.

5.2 Generalizability and Scalability analysis of CAM. In this paper, *generalizability* refers to the performance of the trained model on unseen test scenarios that involve the same (or lower) number of tasks as in the scenarios used for training; and where the test and training scenarios are drawn from the same probability distribution over task locations and deadlines. Here, generalizability is analyzed on test scenarios with the number of tasks fixed at 50 and 200, drawn from the same distribution over the 2D space, and the number of robots fixed at 5 and 40. In this paper, *scalability* refers to the performance of the trained model over test scenarios

with higher (and increasing) numbers of tasks and robots than those encountered in the scenarios used for training. Here, we analyze scalability by evaluating the CAM and AM models on test scenarios with the number of tasks fixed at 500 and 1000, and the number of robots fixed at 50 and 100. The task-to-robot ratio is however kept the same across the generalizability and scalability analysis cases, in order not to introduce another control factor affecting the numerical experiments.

To measure and compare performance, we use two metrics: 1) Average cost function (Eq. 1): This metric accounts for the completion rate of tasks and the total traveled distance, averaged over the set of test scenarios; and 2) Average computing time: This measures how long each method takes to compute the entire solution during runtime, averaged over the test set. From a statistical perspective, the number of tasks, the number of robots, and the method are considered as the independent variables, while the task completion rate and the cost are considered as the dependent variables. Therefore we first perform a Multivariate Analysis of Variance (MANOVA) to determine if the independent variables have an effect on the dependent variables. Test values are reported in Table 5 in Appendix B [57]. This process is followed by the Analysis of Variance (ANOVA) to determine if the independent variables have an effect on the cost. The corresponding test values are reported in Table 8 in Appendix C [57]). The ANOVA test suggests that the above factors have an influence. Hence, we performed a pairwise T-test (table 11 in Appendix D [57]) to compare the baseline methods to CAM on the different scenarios characterized by the number of tasks and the number of robots. The p-values from these tests suggest that there is a significant difference between the cost obtained by CAM compared to baseline methods for all scenarios.

Generalizability: Figures 4(*a*), 4(*c*), and 4(*e*) show the cost function (the lower the better) obtained by each method for the unseen test with 50, 100 and 200 tasks. The corresponding task completion rates are shown in Figs. 4(*b*), 4(*d*), and 4(*f*). A cost function value of less than 0 indicates that 100% task completion has been achieved and only the distance contributes to the cost, as seen from the objective function in Eq. 1. Note that, among the tested scenarios in the 50-task-5-robot and 50-task-10-robot cases, for 5/100 scenarios and 53/100 scenarios, respectively, the INLP solver could not find feasible solutions within the allowed maximum run time. Results reported for INLP in Figs. 4(*a*) and 4(*b*) reflect only those scenarios for which at least a feasible solution was found by the solver within the allowed maximum run time. From Figs. 4(*a*) and 4(*b*), CAM demonstrates 100% task completion in almost all scenarios and outperforms all the other methods. For 50-task-10-robots, INLP demonstrates a comparable performance to CAM, however, INLP has a larger variance compared to CAM and also takes significantly more computing time – 25 mins for INLP as compared to .16 seconds by CAM (Table 2). Note that since the INLP process is clipped at 25 mins of maximum run time, it is not necessarily achieving converged optimal solutions, which is responsible for the poorer-than-expected median performance (in both cases) and variance (in 10 robot case) across the 50-task scenarios for INLP. Theoretically, given enough time and computing resources, exact solutions to the INLP should outperform all other methods.

In light of the above observation, to provide an empirical understanding of the worst-case optimality gap of CAM, we compute the average performance difference between CAM and INLP in the worst 5% of scenarios for the 50-task-5-robot and 50-task-10-robot cases – i.e., the average of 5 scenarios that have the highest value of Cost(CAM) - Cost(INLP). It was found that CAM has an averaged optimality gap of 0.16 and 0.36 in the 5% worst-case scenarios for the 50-task-5-robot and 50-task-10-robot settings, respectively. However, even in these worst-case scenarios, CAM reported the same task completion % as the INLP, thus showing that the optimality gap is mainly attributed to a relative larger distance travelled by robots when using the CAM policy.

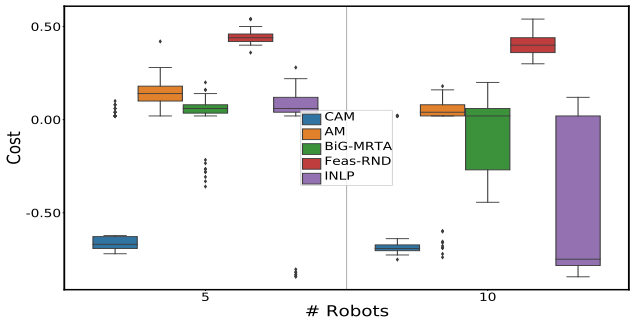
For 100-tasks-20-robots scenarios, CAM clearly outperforms all the competing methods in terms of median cost and median task completion %. The relative performance for all the methods for scenarios with 200 tasks, as shown in Fig. 4(*e*) and 4(*f*), is similar to that of 100 tasks. Only in one case (20-robot-200-task), Big-MRTA performs comparable to that of CAM. Overall, across scenarios with different numbers of tasks and robots, we therefore observe from Fig. 4 that the relative performance of the methods follows roughly the same order (from best to worst) – CAM, Big-MRTA, AM and FeasRND. The ability of CAM to clearly outperform the state-of-the-art graph learning baseline, AM (Fig. 4), demonstrates that encoding choices in the CAM architecture provide a better representation of task space.

For scenarios with 100-tasks-10-robots and 200-tasks-20-robots, (Figs. 4(*c*), 4(*d*), and 4(*e*), 4(*f*), respectively), CAM experiences a noticeably higher variance in cost performance even though only a marginal increase in variance is observed with respect to task completion %. This is because there are several scenarios (42 out of 100 in the 100-tasks-10-robots case, and 61 out of 100 in the 200-tasks-20-robots case) where CAM is not able to achieve 100% task completion; and the noted variance is an artifact of the objective function (f_{cost} in Eq. 1) being discontinuous about the point where $\psi = 0$.

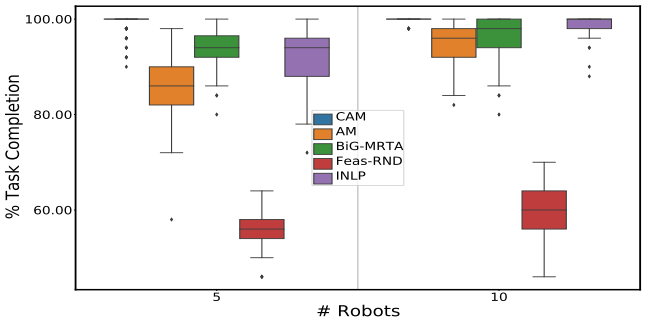
Scalability: To investigate the scalability of the learned model, we use a new set of unseen test scenarios with greater numbers of tasks and robots compared to that encountered during the training of CAM and AM; training used only 200-tasks scenarios. Figures 5(*a*) and 5(*c*) show the performance of the trained model of CAM and the competing methods in terms of the cost function (lower the better) for four large case studies, involving 500-tasks-50-robots, 500-tasks-100-robots, 1000-tasks-100-robots, and 1000-tasks-200-robots. For each case, 100 randomly generated scenarios are used for statistical analyses. The corresponding task completion rate performances are shown in Figs. 5(*b*) and 5(*d*). It can be observed from these figures that CAM clearly outperforms AM and the feasible random baseline. Its comparison to BiG-MRTA is mixed, which is however not surprising given that BiG-MRTA with its incentive function tuned for this flood response case study is arguably one of the strongest methods in the literature for this particular MRTA problem. As seen from Fig. 5, BiG-MRTA slightly outperforms CAM for the 500-tasks-50-robots case, while for the other cases, CAM and BiG-MRTA have comparable performance, with CAM having a higher standard deviation compared to BiG-MRTA. However, when computing time is considered, the advantages of learned policies such as in CAM, compared to graph matching approaches such as BiG-MRTA, become apparent and are discussed later.

Lastly, similar to the 100-tasks-10-robots scenarios in the generalizability analyses (Fig. 4(*c*)), we observe a high variance for 500-tasks-100 robots and 1000-tasks-200 robots cases here. This is again because there is a significant number of scenarios with 100% task completion and those without (27 out of 100), that considered together leads to larger performance variance due to the discontinuous nature of the cost function, as explained earlier.

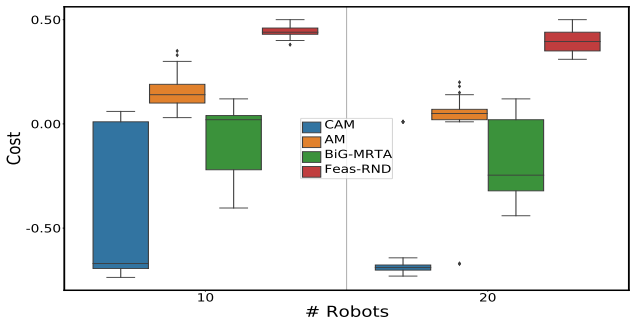
Computing time (Training and Execution) Based on the epoch information in Section 4.1, the average time to complete a training epoch for the learning-based methods CAM and AM is found to be 19.50 minutes, i.e., ~11.7 seconds per sample scenario on average. Tables 2 and 3 provide the scenario-averaged measurements of computing time needed by each method for generating solutions during testing. It can be observed from these tables that the learned policies (CAM and AM) take 0.14 s to 20 s to compute the entire MRTA solution (sequence of tasks assigned to each robot), as the number of tasks grows from 50 to 1000. In comparison, while BiG-MRTA's computing time is comparable for small cases (with 50-100 tasks), for 200-task cases it is 4× slower, and for the largest case of 1000-tasks, BiG-MRTA is over 70× slower than CAM and AM. This run-time performance advantage of CAM can be critical to success in time-sensitive multi-robot operations, such as disaster response and warehouse operations.



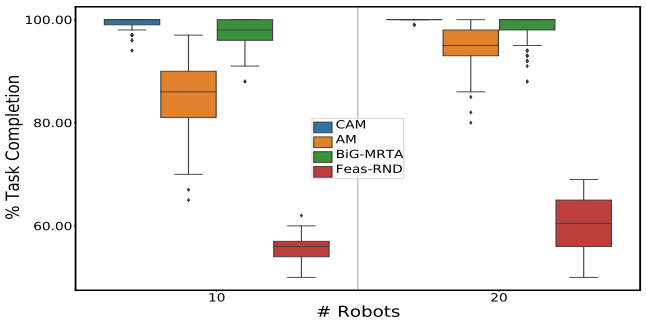
(*a*) Cost for scenarios with 50 tasks. Lower the better.



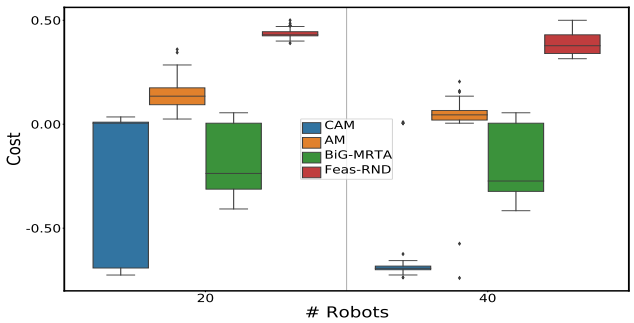
(*b*) % Task completion for scenarios with 50 tasks. Higher the better.



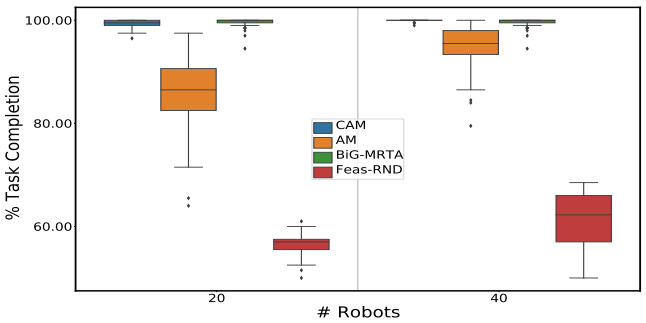
(*c*) Cost for scenarios with 100 tasks. Lower the better.



(*d*) % Task completion for scenarios with 100 tasks. Higher the better.



(*e*) Cost for scenarios with 200 tasks. Lower the better.



(*f*) % Task completion for scenarios with 200 tasks. Higher the better.

Fig. 4 Generalizability analysis on unseen test cases: 100 sample scenarios are tested for each #tasks/#robots case.

Table 2 Generalizability: Task size (up to 200) and number of robots (up to 40). Average computation time in seconds. Lower the better.

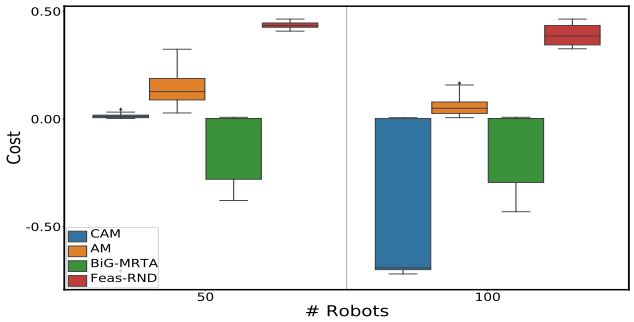
# of Tasks	# of Robots	Avg. Computing Time in seconds				
		BiG-MRTA	**Feas-RND**	**AM**	**CAM**	**INLP**
50	5	0.21	0.5	0.14	0.14	1500
	10	0.20	0.3	0.15	0.16	1500
100	10	0.8	1.2	0.33	0.32	-
	20	1.21	0.8	0.34	0.34	-
200	20	4.72	1.9	1.34	1.21	-
	40	8.37	1.1	1.43	1.40	-

Table 3 Scalability: Task size (up to 1000) and number of robots (up to 200). Average computation time in seconds. Lower the better.

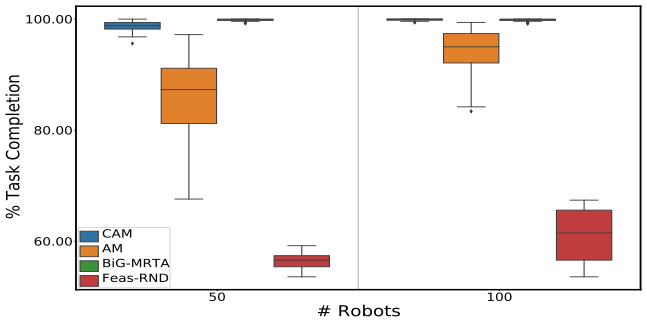
# of Tasks	# of Robots	Avg. Computing Time in seconds			
		BiG-MRTA	**Feas-RND**	**AM**	**CAM**
500	50	69.3	4.20	4.23	4.31
	100	135.3	2.6	4.29	4.71
1000	100	595.4	8.7	19.22	19.05
	200	1420	5.40	20.00	20.03

It is important to point out that decision-making promptness for any online method can however be also affected by imperfect communication, e.g., latency or wireless range constraints, especially if robots are made to wait for the latest information on the state of peer robots or on new tasks appearing during mission. Alternatively, if decisions are taken without waiting, [3] has shown that performance degradation can occur due to conflicting decisions by robots – i.e., a robot i choosing a task that has been recently selected by another peer robot j, but information from robot j has not yet reached robot i when the latter is taking its decision. CAM and other learning-based approaches can be expected to also experience similar performance loss trend. Potential solutions to this issue could include consideration of a partially observable MDP or POMDP formulation of the problem to train CAM. Hence, extending the simulation environment to implement communication latency or range-based constrained communication, and exploring the training of CAM under this changed environment both with the current MDP or a new POMDP formulation are critical directions of future work on CAM-based MRTA.

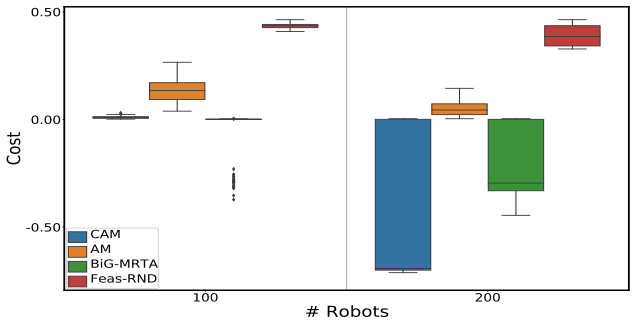
5.2.1 Ablation studies. We performed two ablation studies on CAM to understand the importance of the novel encoder (based on Covariant Compositional Networks or CCN) and the decoder (adopted from the attention mechanism or AM). In the first ablation study, the CCN-based encoding is replaced with simple feedforward layers, with the decoder remaining the same. In the second ablation



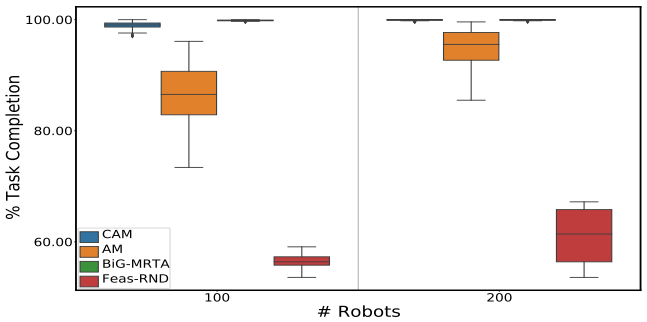
(*a*) Cost for scenarios with 500 tasks. Lower the better.



(*b*) % Task completion for scenarios with 500 tasks. Higher the better.



(*c*) Cost for scenarios with 1000 tasks. Lower the better.



(*d*) % Task completion for scenarios with 1000 tasks. Higher the better.

Fig. 5 Scalability analysis on unseen test cases involving greater numbers of tasks and robots than in training: 100 sample scenarios are tested for each #tasks/#robots case.

study, the AM-based decoder is replaced by a simple feedforward network. It should be noted that the node embedding length (d_{embed}), is the same for all the cases.

Encoder ablation: The policy model with the encoder ablated is called $\mathbf{CAM}_{E_{FF}}$. Here the node encoding is performed using:

$$F_{d_i} = W^d d_i^T + b_d \tag{10}$$

$$F_i = W^F F_{d_i} + b_F \tag{11}$$

where, $d_i = [x_i, y_i, \tau_i]$, $\forall i \in V$. W^d, W^F, b_d, and b_F are learnable weights and biases, where $W^d \in \mathbb{R}^{d_{\text{embed}} \times 3}$, $b_d \in \mathbb{R}^{d_{\text{embed}} \times 1}$, and $W^F \in \mathbb{R}^{d_{\text{embed}} \times d_{\text{embed}}}$, $b_F \in \mathbb{R}^{d_{\text{embed}} \times 1}$. The decoder for $CAM_{E_{FF}}$ is the same as that of CAM.

Decoder ablation: The policy model with the decoder ablated is called $\mathbf{CAM}_{D_{FF}}$, which takes in the node embeddings and the context information and computes the output probabilities using:

$$P^{\text{Act}} = softmax([P_1^{\text{Act}}, \ldots, P_N^{\text{Act}}]), \tag{12}$$

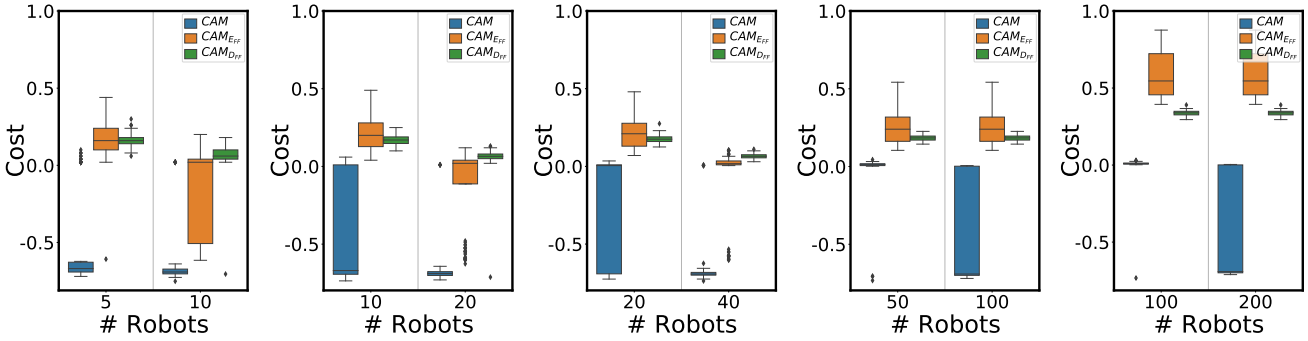
$$P_i^{\text{Act}} = \text{LeakyReLU}(W^{\text{dec}} Concat(F_i, \Omega)^T + b_{\text{dec}}), \text{ where } i \in V$$

Here, W^{dec} and b_{dec} are learnable weights and biases, where $W^{\text{dec}} \in \mathbb{R}^{N \times 2d_{\text{embed}}}$ and $b_{\text{dec}} \in \mathbb{R}^{N \times 1}$. $F_i \; \forall \; i \in V$ are the node embeddings from the encoder, and $\Omega \in \mathbb{R}^{d_{\text{embed}}}$ is the context vector. The encoder for $CAM_{D_{FF}}$ is the same as that of CAM.

Both the ablated versions of the CAM model are trained on the MRTA-Multi-UAV flood response problem in Section 4, using the parameters in Table 1. They are tested on the same test scenarios as that used for CAM, AM, and $BiG\text{-}MRTA$ earlier, involving varying numbers of tasks and robots. Test results are shown in Fig. 6). We performed statistical tests similar to the ones in section 5.2. We first performed a MANOVA (Table 6 in Appendix B [57]), then an ANOVA with cost being the dependent variable (Table 9 in Appendix C [57]), followed by pairwise T-test (Table 12 in Appendix D [57]). The p-value of the pairwise T-test suggests that there is a significant difference between the mean of CAM compared to the ablated $CAM_{E_{FF}}$ and $CAM_{D_{FF}}$ models, for all the studied cases.

As can be seen from Fig. 6, the performance of both $CAM_{E_{FF}}$ and $CAM_{D_{FF}}$ is significantly poorer compared to CAM, in terms of the cost function. The performance drop is more for scenarios with a relatively lower number of robots. We found that with the encoder ablated, $CAM_{E_{FF}}$ suffered up to 35% drop in task completion rate compared to CAM. With the decoder ablated, $CAM_{D_{FF}}$ suffered up to 17% drop in task completion rate compared to CAM. These results support the choice of the special CCN-based encoder and MHA-AM decoder portions of the proposed graph neural network for MRTA. It is also observed that particularly for the larger task scenarios, the performance drop is greater with encoder ablation than with decoder ablation, which shows that the encoder contributes more significantly to the scalability to unseen test scenarios with a larger number of tasks.

5.3 MRTA with dynamically generated tasks. One of our hypotheses in this work is that using local neighborhood information for a node instead of merely considering the entire graph, can lead to more meaningful node representations shared by tasks graphs of different sizes. This capability is posited to be also beneficial when dealing with MRTA operations where new tasks get generated as the mission is ongoing since even though the task graph keeps changing in this case, they are expected to be composed of shared local neighborhood structures. To further test this hypothesis we implement the CAM architecture on MRTA problems where new tasks are introduced during the mission based on the same probability distribution from which the pre-mission task graph is drawn in any scenario. The deadline for the new tasks is such that it is greater than the time elapsed at which the task is introduced during the mission. The mission starts with T number of tasks and R robots and as the mission progresses, D number of tasks are introduced and dynamically added at different times before the end of the mission. When a new task is introduced, its node representation is computed on the go using the encoder (Section 3.1),



(*a*) 50 tasks. (*b*) 100 tasks. (*c*) 200 tasks. (*d*) 500 tasks. (*e*) 1000 tasks.

Fig. 6 Ablation analysis comparing the cost of CAM model with two other versions of CAM model with the encoder and decoder respectively ablated ($CAM_{E_{FF}}$ and $CAM_{D_{FF}}$). Lower the better.

and the embedding of all the other nodes is re-computed to account for the changed local neighborhoods.

5.3.1 Test Scenarios with Dynamic Tasks. Note that the same CAM and AM models that are trained previously over static task spaces are used here, to demonstrate the unique generalizability of these GNN-based policies without the need for retraining over dynamic task spaces. Testing on dynamic task spaces is performed for MRTA problems with the number of tasks N varying from 100 to 500 and the number of robots varying from 10 to 50. In each of these # tasks/# robots cases, $\phi_N = 25\%$ of the N tasks are made available at the start of the mission and the remaining 75% tasks are added dynamically between the elapsed times of 0.1 to 0.3 hours.

5.3.2 Performance Analysis on Dynamic Tasks:. In these dynamic task cases, Fig. 7 respectively show the box plots of the cost and task completion rate achieved by CAM, AM, and BiG-MRTA over the testing scenarios. To assess the statistical significance of the results, similar to the case studies in Section 5.2 and 5.2.1, we first performed a MANOVA (Table 7 as given in Appendix B [57]), then an ANOVA with cost being the dependent variable (Table 10 as given in Appendix C [57]), followed by pairwise T-test (Table 13 as given in Appendix D [57]).

Figure 7 shows that CAM clearly outperforms AM in all cases with dynamic tasks. With the exception of the 50-robot-500-task case, CAM also provides a better median cost performance compared to BiG-MRTA (which is notable in the literature for its particular ability to handle dynamic tasks). It is however worth highlighting again that for such large (500-task) problem scenarios, BiG-MRTA is noticeably slower in computing the task decisions. The task completion rates of CAM and BiG-MRTA are mostly comparable across all the cases. These observations thus provide strong evidence for the suitability of CAM for MRTA problems with dynamic tasks. Compared to the other methods, CAM illustrates a higher variance in cost performance across about half of the cases reported in Fig. 7, which is again attributed to the discontinuous nature of the cost function, as discussed before. However, the robustness of CAM is still observable when the task completion rate is considered, as seen in Fig. 7.

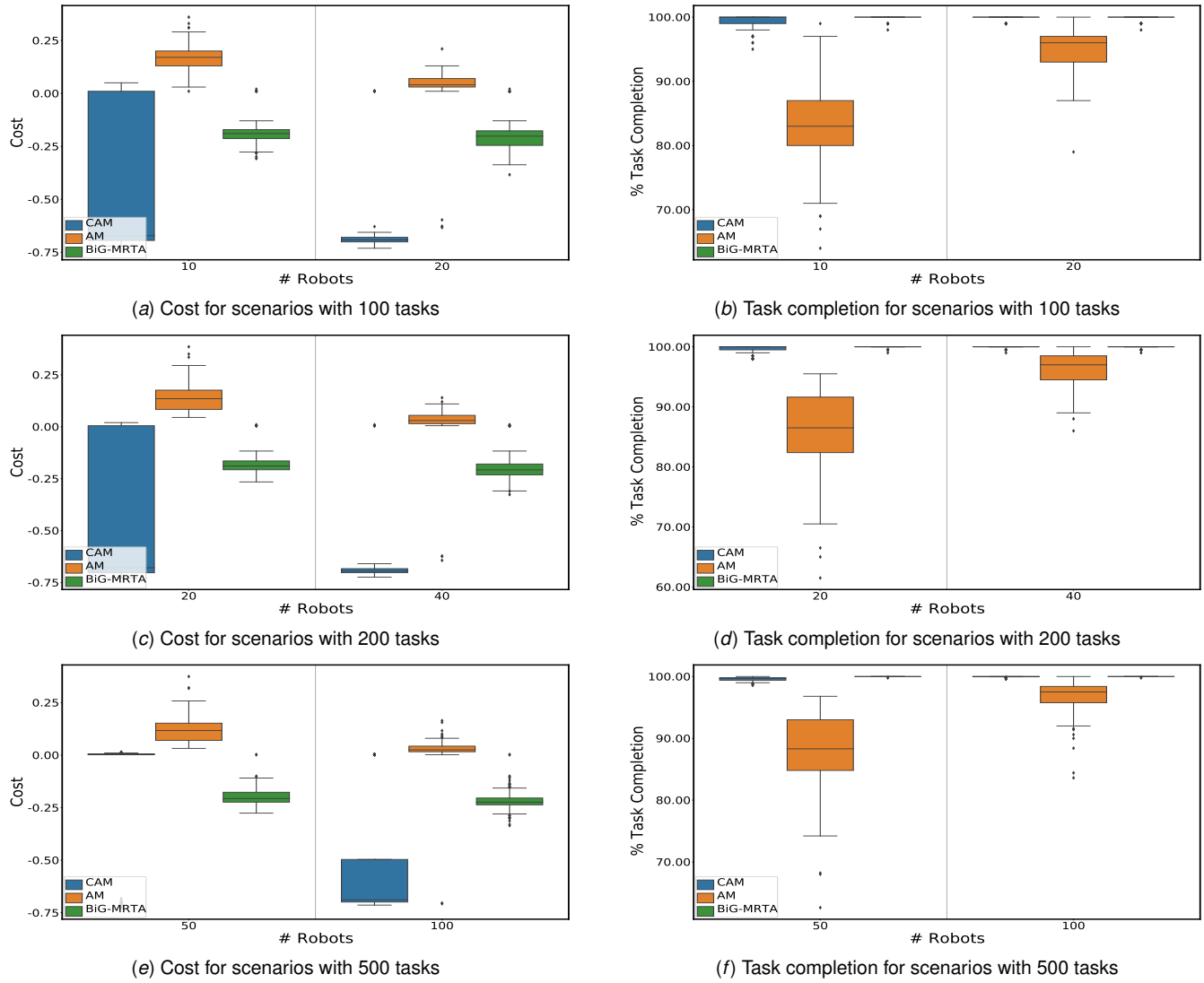
Table 4 shows the computing time of the three methods per decentralized decision taken by a robot, averaged over all decisions taken by all robots during a mission, and further averaged over all mission scenarios under each # tasks/# robots case. Given the presence of dynamic tasks, computing time per task decisions is more reflective of the online or real-time computing burden. In this regard, the computing times of CAM, AM, and BiG-MRTA are comparable for cases with a smaller number of tasks, whereas, for the large 500-task cases, BiG-MRTA takes about $4-5\times$ more computing time compared to CAM.

Table 4 Dynamic task cases: Computing time in seconds per decision by a robot, averaged across all decisions by all robots in all sample scenarios under each # tasks/# robots case. Lower the better.

# of Tasks	# of Robots	Avg. Computing Time in seconds		
		CAM	**BiG-MRTA**	**AM**
100	10	0.004	0.004	0.006
	20	0.006	0.006	0.005
200	20	0.010	0.013	0.010
	40	0.010	0.014	0.007
500	50	0.051	0.230	0.020
	100	0.050	0.330	0.018

6 Conclusion

In this paper, we developed a new graph neural net (GNN) based policy architecture called Covariant Attention Mechanism or CAM, to perform multi-robot task allocation in operations involving tasks with time deadlines, dynamically added tasks, and robots with constrained flight range and payload capacity. This new architecture incorporates an encoder based on covariant compositional network (CCN) embedding and a decoder based on an attention mechanism. The encoder choice was premised on the ability to provide permutation invariant embedding that also captures the local neighborhood structures in large task graphs, while the decoder choice was based on the ability to produce sequential decisions in terms of task selection probabilities. A simple RL algorithm was implemented to train the parameters of this encoder-decoder type CAM architecture. To compare the performance of the proposed CAM method, a state-of-the-art attention mechanism approach (aka AM) was extended to the multi-agent setting. Other comparative methods tested include a recent best-performing (for the concerned problem type) online MRTA method called BiG-MRTA, a myopic random-walk baseline called Feas-RND, and Integer Non-linear Programming with clipped overall computing time. A multi-UAV flood response mission context was considered for testing and performance was analyzed in terms of the task completion rate and a cost value that also includes the total distance traveled by the team of robots over any mission scenario, with 100 unseen scenarios considered under each test case. CAM outperformed BiG-MRTA (with few exceptions), AM, and Feas-RND for most of the test scenarios by achieving a better cost function value. CAM was also able to achieve a high task completion rate for problems that were larger in size than those used during training. The learned CAM policy has shown to be also significantly faster in computing task decisions compared to BiG-MRTA. The ablation study on the encoder along with CAM's performance superiority over AM provided strong evidence regarding the benefits of the CCN, attributed to better capture of local task neighborhoods. The performance of CAM over AM, and its favorable comparison to BiG-MRTA (considering both task completion rate and computing time) readily extended to



(*a*) Cost for scenarios with 100 tasks

(*b*) Task completion for scenarios with 100 tasks

(*c*) Cost for scenarios with 200 tasks

(*d*) Task completion for scenarios with 200 tasks

(*e*) Cost for scenarios with 500 tasks

(*f*) Task completion for scenarios with 500 tasks

Fig. 7 Dynamic task cases: Cost performance analyses of CAM, AM, and BiG-MRTA on unseen MRTA test cases with dynamic tasks (75% of tasks added during operations).

scenarios with dynamic tasks as well.

In its current form, the CAM model makes a greedy choice based on the predicted task selection probabilities. To allow a better balance between exploration/exploitation, an epsilon greedy approach could be explored in the future. In addition, the current design of the task graph and robot context embeddings in CAM assumes that both tasks and robots are homogeneous, communication is perfect (i.e., without latency or dropouts), and the shared workspace is relatively large or not crowded – these do not necessarily fully capture the complexities in applications such as robotic construction and disaster response. Hence, in future work, investigating the ability to handle a heterogeneous and partially observable task graph and agent space, and adopting a joint solution to multi-robot task allocation and motion planning, could provide further insights into the potential for such GNN-based approaches to replace or augment legacy non-learning-based approaches in real-world multi-robot applications.

Acknowledgments

This work was supported by the Office of Naval Research (ONR) award N00014-21-1-2530 and the National Science Foundation (NSF) award CMMI 2048020. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the ONR or the NSF.

References

[1] Gerkey, B. P. and Matarić, M. J., 2004, "A formal analysis and taxonomy of task allocation in multi-robot systems," The International Journal of Robotics Research, **23**(9), pp. 939–954.

[2] Aurambout, J.-P., Gkoumas, K., and Ciuffo, B., 2019, "Last mile delivery by drones: An estimation of viable market potential and access to citizens across European cities," European Transport Research Review, **11**(1), p. 30.

[3] Ghassemi, P. and Chowdhury, S., 2022, "Multi-robot task allocation in disaster response: Addressing dynamic tasks with deadlines and robots with range and payload constraints," Robotics and Autonomous Systems, **147**, p. 103905.

[4] Tian, Y.-T., Yang, M., Qi, X.-Y., and Yang, Y.-M., 2009, "Multi-robot task allocation for fire-disaster response based on reinforcement learning," *2009 International Conference on Machine Learning and Cybernetics*, Vol. 4, pp. 2312–2317, doi: 10.1109/ICMLC.2009.5212216.

[5] Thakur, A., Sahoo, S., Mukherjee, A., and Halder, R., 2023, "Making Robotic Swarms Trustful: A Blockchain-Based Perspective," Journal of Computing and Information Science in Engineering, **23**(6), p. 060803.

[6] Ghassemi, P. and Chowdhury, S., 2020, "An extended bayesian optimization approach to decentralized swarm robotic search," Journal of Computing and Information Science in Engineering, **20**(5), p. 051003.

[7] Liu, C. and Kroll, A., 2015, "Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks," Soft Computing, **19**, pp. 567–584.

[8] Behjat, A., Manjunatha, H., Kumar, A., Jani, Collins, L., Ghassemi, P., Distefano, J., Doermann, D., Dantu, K., Esfahani, E., and Chowdhury, S., 2021, "Learning Robot Swarm Tactics over Complex Adversarial Environments," *International*

Symposium on Multi-Robot and Multi-Agent Systems (MRS'21), IEEE, Cambridge, UK, Nov.

[9] Claes, D., Oliehoek, F., Baier, H., and Tuyls, K., 2017, "Decentralised online planning for multi-robot warehouse commissioning," *AAMAS'17: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS*, Vol. 1, ACM, pp. 492–500.

[10] Poudel, L., Zhou, W., and Sha, Z., 2020, "A generative approach for scheduling multi-robot cooperative three-dimensional printing," Journal of Computing and Information Science in Engineering, **20**(6), p. 061011.

[11] Wang, L., Nie, Q., Zhang, Z., Tang, D., and Liu, C., 2024, "Probing an Easy-to-Deploy Multi-Agent Manufacturing System Based on Agent Computing Node: Architecture, Implementation, and Case Study," Journal of Computing and Information Science in Engineering, **24**(4).

[12] Fang, W., Zhang, H., Qian, W., Guo, Y., Li, S., Liu, Z., Liu, C., and Hong, D., 2023, "An adaptive job shop scheduling mechanism for disturbances by running reinforcement learning in digital twin environment," Journal of Computing and Information Science in Engineering, **23**(5), p. 051013.

[13] Yan, Z., Jouandeau, N., and Ali-Chérif, A., 2012, "Multi-robot heuristic goods transportation," *2012 6th IEEE International Conference Intelligent Systems*, IEEE, pp. 409–414.

[14] Maoudj, A., Bouzouia, B., Hentout, A., and Toumi, R., 2015, "Multi-agent approach for task allocation and scheduling in cooperative heterogeneous multi-robot team: Simulation results," *2015 IEEE 13th international conference on industrial informatics (INDIN)*, IEEE, pp. 179–184.

[15] Paul, S., Witter, J., and Chowdhury, S., 2024, "Graph Learning-based Fleet Scheduling for Urban Air Mobility under Operational Constraints, Varying Demand & Uncertainties," *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pp. 638–645.

[16] Nunes, E., Manner, M., Mitiche, H., and Gini, M., 2017, "A taxonomy for task allocation problems with temporal and ordering constraints," Robotics and Autonomous Systems, **90**, pp. 55–70.

[17] Nallusamy, R., Duraiswamy, K., Dhanalaksmi, R., and Parthiban, P., 2009, "Optimization of non-linear multiple traveling salesman problem using k-means clustering, shrink wrap algorithm and meta-heuristics," International Journal of Nonlinear Science, **8**(4), pp. 480–487.

[18] Toth, P. and Vigo, D., 2014, *Vehicle routing: problems, methods, and applications*, SIAM.

[19] Khamis, A., Hussein, A., and Elmogy, A., 2015, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative Robots and Sensor Networks 2015*, Springer, pp. 31–51.

[20] Dantzig, G. B. and Ramser, J. H., 1959, "The truck dispatching problem," Management science, **6**(1), pp. 80–91.

[21] Bektas, T., 2006, "The multiple traveling salesman problem: an overview of formulations and solution procedures," Omega, **34**(3), pp. 209–219.

[22] Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I., 2016, "The vehicle routing problem: State of the art classification and review," Computers & Industrial Engineering, **99**, pp. 300–313.

[23] Azi, N., Gendreau, M., and Potvin, J.-Y., 2010, "An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles," European Journal of Operational Research, **202**(3), pp. 756–763.

[24] Wang, D., Hu, M., and Gao, Y., 2018, "Multi-Criteria Mission Planning for a Solar-Powered Multi-Robot System," *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers Digital Collection.

[25] Jose, K. and Pratihar, D. K., 2016, "Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods," Robotics and Autonomous Systems, **80**, pp. 34–42.

[26] Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E., 2021, "Reinforcement learning for combinatorial optimization: A survey," Computers & Operations Research, **134**, p. 105400.

[27] Archetti, C., Feillet, D., Gendreau, M., and Grazia Speranza, M., 2011, "Complexity of the VRP and SDVRP," Transportation Research Part C: Emerging Technologies.

[28] Cattaruzza, D., Absi, N., and Feillet, D., 2016, "Vehicle routing problems with multiple trips," 4OR, **14**(3), pp. 223–259.

[29] Dias, M. B., Zlot, R., Kalra, N., and Stentz, A., 2006, "Market-based multirobot coordination: A survey and analysis," Proceedings of the IEEE, **94**(7), pp. 1257–1270.

[30] Schneider, E., Sklar, E. I., Parsons, S., and Özgelen, A. T., 2015, "Auction-based task allocation for multi-robot teams in dynamic environments," *Conference Towards Autonomous Robotic Systems*, Springer, pp. 246–257.

[31] Ismail, S. and Sun, L., 2017, "Decentralized hungarian-based approach for fast and scalable task allocation," *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, pp. 23–28.

[32] Ghassemi, P., DePauw, D., and Chowdhury, S., 2019, "Decentralized dynamic task allocation in swarm robotic systems for disaster response," *2019 international symposium on multi-robot and multi-agent systems (mrs)*, IEEE, pp. 83–85.

[33] Mitiche, H., Boughaci, D., and Gini, M., 2019, "Iterated local search for time-extended multi-robot task allocation with spatio-temporal and capacity constraints," Journal of Intelligent Systems, **28**(2), pp. 347–360.

[34] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D., 2009, "Iterated local search for the team orienteering problem with time windows," Computers & Operations Research, **36**(12), pp. 3281–3290, New developments on hub location.

[35] Qian, B. and Cheng, H. H., 2018, "Bio-inspired coalition formation algorithms for multirobot systems," Journal of Computing and Information Science in Engineering, **18**(2), p. 021010.

[36] Choudhury, S., Gupta, J. K., Kochenderfer, M. J., Sadigh, D., and Bohg, J., 2022, "Dynamic multi-robot task allocation under uncertainty and temporal constraints," Autonomous Robots, **46**(1), pp. 231–247.

[37] Wei, C., Hindriks, K. V., and Jonker, C. M., 2016, "Dynamic task allocation for multi-robot search and retrieval tasks," Applied Intelligence, **45**, pp. 383–401.

[38] Kool, W., Van Hoof, H., and Welling, M., 2019, "Attention, learn to solve routing problems!" *7th International Conference on Learning Representations, ICLR 2019*, 1803.08475

[39] Barrett, T., Clements, W., Foerster, J., and Lvovsky, A., 2020, "Exploratory combinatorial optimization with reinforcement learning," *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34, Paper No. 04, pp. 3243–3250.

[40] Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L., 2017, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, pp. 6348–6358.

[41] Kaempfer, Y. and Wolf, L., 2018, "Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Networks," ArXiv, **abs/1803.09621**.

[42] Li, Z., Chen, Q., and Koltun, V., 2018, "Combinatorial optimization with graph convolutional networks and guided tree search," *Advances in Neural Information Processing Systems*, pp. 539–548.

[43] Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J., 2017, "A note on learning algorithms for quadratic assignment with graph neural networks," stat, **1050**, p. 22.

[44] Tolstaya, E., Paulos, J., Kumar, V., and Ribeiro, A., 2021, "Multi-robot coverage and exploration using spatial graph neural networks," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 8944–8950.

[45] Sykora, Q., Ren, M., and Urtasun, R., 2020, "Multi-agent routing value iteration network," *37th International Conference on Machine Learning, ICML 2020*, 2007.05096

[46] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L., 2017, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, 1704.01665

[47] Paul, S., Ghassemi, P., and Chowdhury, S., "Learning Scalable Policies over Graphs for Multi-Robot Task Allocation using Capsule Attention Networks," *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8815–8822, doi: 10.1109/ICRA46639.2022.9812370.

[48] Paul, S. and Chowdhury, S., 2022, "A scalable graph learning approach to capacitated vehicle routing problem using capsule networks and attention mechanism," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 86236, American Society of Mechanical Engineers, p. V03BT03A045.

[49] Strens, M. and Windelinckx, N., 2005, "Combining Planning with Reinforcement Learning for Multi-robot Task Allocation," *Adaptive Agents and Multi-Agent Systems II*, D. Kudenko, D. Kazakov, and E. Alonso, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 260–274.

[50] Wang, Z. and Gombolay, M., 2020, "Learning Scheduling Policies for Multi-Robot Coordination With Graph Attention Networks," IEEE Robotics and Automation Letters, **5**(3), pp. 4509–4516.

[51] Hy, T. S., Trivedi, S., Pan, H., Anderson, B. M., and Kondor, R., 2018, "Predicting molecular properties with covariant compositional networks," The Journal of chemical physics, **148**(24), p. 241745.

[52] Jacob, R. A., Paul, S., Chowdhury, S., Gel, Y. R., and Zhang, J., 2024, "Real-time outage management in active distribution networks using reinforcement learning over graphs," Nature Communications, **15**(1), p. 4766.

[53] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., 2017, "Attention is all you need," Advances in neural information processing systems, **30**.

[54] Paul, S., Li, W., Smyth, B., Chen, Y., Gel, Y., and Chowdhury, S., 2023, "Efficient Planning of Multi-Robot Collective Transport using Graph Reinforcement Learning with Higher Order Topological Abstraction," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5779–5785, doi: 10.1109/ICRA48891.2023.10161517.

[55] Hamilton, W., Ying, Z., and Leskovec, J., 2017, "Inductive representation learning on large graphs," Advances in neural information processing systems, **30**.

[56] Verma, S. and Zhang, Z.-L., 2018, "Graph Capsule Convolutional Neural Networks," stat, **1050**, p. 26.

[57] Paul, S., 2024, "adamslab-ub/CAM_ASME_JCISE: Version1.0," doi: 10.5281/zenodo.11910732, https://doi.org/10.5281/zenodo.11910732

[58] Force, U. T., 2011, "Unmanned aircraft system airspace integration plan," Department of Defense.

[59] Karp, R. M., 1980, "An algorithm to solve the m× n assignment problem in expected time O (mn log n)," Networks, **10**(2), pp. 143–152.