Learning to Solve Multi-Robot Task Allocation with a Covariant-Attention based Neural Architecture

Anonymous Author(s)

Abstract: This paper demonstrates how time-constrained multi-robot task allocation (MRTA) problems can be modeled as a Markov Decision Process (MDP) over graphs, such that approximate solutions can be modeled as a policy using Reinforcement Learning (RL) methods. Inspired by emerging approaches for learning to solve related combinatorial optimization (CO) problems such as multi-traveling salesman (mTSP) problems, a graph neural architecture is conceived in this paper to model the MRTA policy. The generalizability and scalability needs of the arguably more complex CO problem presented by MRTA (compared to say mTSP) are addressed by innovatively using the concept of Covariant Compositional Networks (CCN) to learn the local and global structures of graphs. The resulting learning architecture is called Covariant Attention-based Mechanism or CAM, which comprises three main components: 1) an encoder: CCN-based embedding model to represent the task space as learnable feature vectors, 2) a decoder: an attention-based model to facilitate sequential decision outputs, and 3) context: to represent the state of the mission and the robots. To learn the feature vectors, a policy-gradient method is used. The CAM architecture is found to generally outperform a state-of-the-art encoder-decoder method that is purely based on Multi-head Attention (MHA) mechanism (which has been shown to solve a wide variety of CO problems) in terms of task completion and cost function, when applied to a class of MRTA problems with time deadlines, robot ferry range constraints, and multi-tour allowance. CAM also demonstrated remarkably better scalability (albeit with increased variance in performance), i.e., in terms of cost function over unseen scenarios with larger task/robot spaces than those used for training. Lastly, the 2-3 orders of magnitude smaller computing time of the learned models (along with better cost values) compared to a metaheuristic MRTA approach (implemented via Google OR tools) provide further evidence regarding the unique potential of learning approaches in delivering exceptionally time-efficient solutions even for large problems involving up to 1000’s of tasks.

Keywords: MRTA, Reinforcement learning, graph learning

1 Introduction

In multi-robot task allocation (MRTA) problems, we study how to coordinate tasks among a team of cooperative robotic systems such that the decisions are free of conflict and optimize a quantity of interest [1]. The potential real-world applications of MRTA are immense, considering that multi-robotics is one of the most important emerging directions of robotics research and development, and task allocation is fundamental to most multi-robotic or swarm-robotic operations. Example applications include disaster response [2], last-mile delivery [3], environment monitoring [4], and reconnaissance [5]). Although various approaches (e.g., graph-based methods [6], integer-linear programming (ILP) approaches [7, 8], and auction-based methods [9, 10]) have been proposed to solve the combinatorial optimization problem underlying MRTA operations, they usually do not scale well with number of robots and/or tasks, and do not readily adapt to complex problem characteristics without tedious hand-crafting of the underlying heuristics. In the recent years, a rich body of work has emerged on using learning-based techniques to model solutions or intelligent heuristics for combinatorial optimization (CO) problems over graphs. The existing methods are mostly limited to classical CO problems, such as multi-traveling salesman (mTSP), vehicle routing (VRP), and max-cut type of problems. In this paper, we are instead interested in learning policies for an important class of MRTA problems [11] that include characteristics such as tasks with time deadlines, robots with constrained range, and ability to conduct multiple-tours.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.
In this paper, we show how such MRTA problems can be modeled as a Markov Decision Process over graphs, allowing us to learn task allocation policies by performing reinforcement learning (RL) over graphs. We specifically focus on a class of MRTA problems that falls into the Single-task Robots, and Single-robot Tasks (SR-ST) class defined in [1, 12]. Here, a feasible and conflict-free task allocation is defined as assigning any task to only one robot [6]. Subsequently, we propose a new covariant attention-based model (aka CAM), a neural architecture for learning over graphs to construct the MRTA policies. This architecture builds upon the attention mechanism concept and innovatively integrates an equivariant embedding of the graph to capture graph structure while remaining agnostic to node ordering.

1.1 Multi-Robot Task Allocation
The MRTA problem can be formulated as an Integer Linear Programming (ILP) or mixed ILP. When tasks are defined in terms of location, the MRTA problem becomes analogous to the Multi-Traveling Salesmen Problem (mTSP) [13] and its generalized version, the Vehicle Route Planning (VRP) problem [14]. Existing solutions to mTSP and VRP problems in the literature [15, 16] have addressed analogical problem characteristics of interest to MRTA, albeit in a disparate manner; these characteristics include tasks with time deadlines, and multiple tours per vehicle, with applications in the operations research and logistics communities [17, 18]. ILP-based mTSP-type formulations and solution methods have also been extended to task allocation problems in the multi-robotic domain [19]. Although the ILP-based approaches can in theory provide optimal solutions, they are characterized by exploding computational effort as the number of robots and tasks increases [8, 20].

For example, for the studied SR-ST problem, the cost of solving the exact ILP formulation of the problem, even with a linear cost function (thus an ILP), scales with $O(n^3m^2h^2)$, where $n$, $m$, and $h$ represent the number of tasks, the number of robots, and the maximum number of tours per robot, respectively [6]. As a result, most practical online MRTA methods, e.g., auction-based methods [9] and bi-graph matching methods [2], use some sort of heuristics, and often report the optimality gap at least for smaller test cases compared to the exact ILP solutions. Recently, it has been shown that Graph Neural Networks (GNNs) can provide an alternative method with a computational efficient run-time [21].

1.2 Contributions of this Paper
The primary contributions of this paper can be stated as follows: 1) Formulating the general SR-ST class of MRTA problems as a Markov Decision Process or MDP over graphs, such that the optimal (task allocation) policy can be learned using an RL approach; 2) A Graph Neural Network architecture for multi-agent task/resource allocation problems, inspired by a state-of-the-art attention mechanism based method [21] (which is exceptionally effective in solving CO problems like TSP and VRP); 3) Improving the state-of-the-art AM method to generalize better and scale better without the need to retrain the model for larger sized (higher number of nodes) problems; 4) Incorporating a covariant compositional mechanism for node-based embedding (aka encoder) of the graph such that local structural information along with permutation invariance, and associated task properties are preserved; 5) Extending the attention-based mechanism (AM) [21], serving as a decoder, and associated context to a multi-agent combinatorial optimization setting. The proposed CAM architecture is evaluated on a large representative problems of MRTA (involves coordinating a team of multiple unmanned aerial vehicles (UAVs) to respond to flood victims by dropping survival packages) and the capacitated VRP. Further investigation is provided to show the important evidence of the promising convergence of the CAM method, and its advantage over the standard attention mechanism approach, in terms of the cost function and scalability of the learnt policy with number of tasks. The remaining portion of the paper is organized as follows. Section 2 describes the MRTA problem definition and its formulation as a MDP over graphs. Section 3 presents our proposed new graph neural network for the MRTA problem. Section 5 describes simulation settings and different case studies. Results, encapsulating the performance of these methods on different-sized problems and various analyses of the proposed method, are presented in Section 6. The paper ends with concluding remarks.

2 MRTA: Problem Definition and Formulations
The multi-robot task allocation (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 (conflict-free allocation) that maximizes the mission outcome (e.g., fraction of tasks completed) or minimize the mission cost (e.g., total distance travelled) subject to the robots’ range
constraints. 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 \(t_i\) 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. Each tour is defined as departing from the depot, undertaking at least one task, and returning to the depot. This MRTA problem is a class of combinatorial optimization problems, which can be modeled in graph space. 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, described next. The optimization formulation of this MRTA is then given in Section 2.2.

### 2.1 MDP over a Graph

The MRTA problem can be represented as a complete graph \(G = (V, E)\), which contains a set of nodes/vertices \(V\) and a set of edges \(E\) that connect the vertices to each other. Each node is a task, and each edge connects a pair of nodes. The weight of the edge \((\omega_{ij})\) represents the cost (e.g., distance) incurred by a robot to take task-\(j\) after achieving task-\(i\). For MRTA with \(N\) tasks, the number of vertices and the number of edges are \(N\) and \(N(N - 1)/2\), respectively. Node \(i\) is assigned a 3-dimensional feature vector denoting the task location and time deadline, i.e., \(d_i = [x_i, y_i, t_i]\).

The MDP defined in a decentralized manner for each individual robot (to capture its task selection process) can be expressed as a tuple \(< S, A, P, \omega, R >\). The components of the MDP can be defined as:

- **State Space** \((S)\): A robot at its decision-making instance uses a state \(s \in S\), which contains the following information: 1) the current mission time, 2) its current location, 3) its remaining ferry-range (battery state), 4) the planned (allocated) tasks of its peers, 5) the remaining ferry-range of its peers, and 6) the states of tasks. The states of tasks contain the location, the time deadline, and the task status – active, completed, and missed (i.e., deadline is passed). **Action Space** \((A)\): The set of actions is represented as \(A\), 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). \(P_a(s'|s, a)\): 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** \((R)\): The reward function is defined as \(-f_{\text{cost}}\), and is calculated when there is no more active tasks (all tasks has been visited once irrespective of it being completed or missed). **Transition**: The transition is an event-based trigger. An event is defined as the condition that a robot reaches its selected task or visits the depot location.

### 2.2 MRTA as Optimization Problem

This MRTA problem is adopted from [22] with the following modification – payload constraints are not imposed on the robot. The exact solution of the MRTA problem can be obtained by formulating it as an integer nonlinear programming problem, which can be summarily expressed as:

\[
\begin{align*}
\min & \quad f_{\text{cost}} = r - u(r)e^{-d^2} \\
\text{where} \quad r & \in [0, 1] \quad \text{and} \quad u(r) = \begin{cases} 
1 & \text{if } r = 0 \\
0 & \text{otherwise}
\end{cases} \\
\text{subject to} & \quad t_i^f < t_i \\
& \quad d_{ij} \leq \Delta_k, k \in [1, N_r] \quad i, j \in [0, N]
\end{align*}
\]  

Here \(t_i^f\) is the time at which task-\(i\) was completed, \(\Delta_k\) is the available range for robot-\(k\) at any point of time, and \(d_{ij}\) is the distance between nodes-\(i\) and \(j\). A detailed formulation of the exact ILP constraints that describe the MRTA problem with range restrictions, multi-tours per robot and tasks with deadlines, can be found [22]. Note that, here we use a slightly different objective/cost function to better reflect the generalized setting for the class of MRTA problems with ferry-range and task-deadline constraints (unlike the slightly more application-related setting used in [22]). Here, we craft the objective function (Eq. (1)) such that it emphasises maximizing the completion rate (i.e., the number of completed tasks divided by the total number of tasks); and if perfect completion rate (100\%) is feasible, then the travelled cost is also considered. The term of \(1 - r\) is defined as task completion rate; i.e., the number of completed tasks (\(N_{\text{success}}\)) divided by the total tasks (\(N\)) or \(r = \frac{N - N_{\text{success}}}{N}\). \(d_r\) is a normalized value of the total distance travelled by all robots in the team. The
term \(d_r\) is the average travelled distance over all robots (i.e., \(d_r = \frac{\sum_{i=1}^{N_r} d_i^{\text{total}}}{\sqrt{2}N}\)). The terms \(N_r\) and \(d_i^{\text{total}}\) represent respectively the number of robots and the total traveled distance by robot \(i\) during the entire mission. The above objective function (Eq. (1)) gives a positive value if the completion rate is lower than 100\%, otherwise it gives a negative value.

3 Covariant Attention-based Neural Architecture

For learning to work on the MDP defined over graphs in Section 2.1, we need to represent each node as a continuous vector, preserving its properties as well as the 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 and 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 called whether the online CAM model is executed centrally off-board or on-board each robot. As an example, Figure 1 illustrates how robot-1 and robot-2 uses 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 includes 1) the task graph information (i.e., the location of each task and its associated time deadline), 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 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., missed deadline) 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 encoder and decoder components are 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 is essential include a node’s local neighborhood information, and permutation invariance. 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 like GraphSAGE [23], thus enabling the GNN to generalize for problems with larger number of nodes without the need to retrain. The encoder represent the properties of each graph node (preserving its structural information) into a continuous feature vector of dimension \(d_{\text{embed}}\), which is fed to the decoder. Each node \(i\), has three properties which are the \(x\)-coordinate \((x)\), \(y\)-coordinate \((y)\), and the time deadline \((t)\) of the task. The encoding for each node should include its properties and the its positional association with its
neighboring nodes. We implement a variation of CCN [24]. We determine the nearest $k$ neighbors of a node ($Nb_i$) 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 $F_{id}$ for all $i \in [1, N]$, i.e., $F_{id} = W^d d_i^T + b_d$. Here $W^d \in \mathbb{R}^{d_{emb} \times 3}$, $b_d \in \mathbb{R}^{d_{emb} \times 1}$, $d_i = [x_i, y_i, t_i]$. For effective decision making, we also need to preserve the structural information. Therefore we define a matrix $d_{iNb}$ as defined in Eq. (5).

$$d_{iNb} = \text{Concat}(d_{j}), \quad j \in Nb_i$$

We do a linear transformation of $d_{iNb}$ to get $F_{iNb}$ (as shown in Eq. (6)), which we believe captures the association of a node with its local neighbors in terms of the node properties.

$$F_{iNb} = W^{Nb}(d_{iNb} - d_i^T) + b^{Nb}$$

where $W^{Nb} \in \mathbb{R}^{d_{emb} \times 3}$, $b^{Nb} \in \mathbb{R}^{d_{emb} \times 1}$, $F_{iNb}$ captures the information about how close the node properties of neighbor nodes of node $i$ is to itself, which shows a representation of how important node $i$ is to its neighbors. We compute the final embedding for each node using Eq. (7).

$$F_i = \text{Aggregate}(W_f(\text{Concat}(F_{id}, F_{iNb})) + b_f)$$

Here, $W_f \in \mathbb{R}^{d_{emb} \times d_{emb}}$, $b_f \in \mathbb{R}^{d_{emb} \times 1}$. Thus finally we get an embedding $F_i$ for each node, where $F_i \in \mathbb{R}^{d_{emb} \times 1}$, $W_f$, $b_d$, $W^{Nb}$, $b^{Nb}$, $W_f$, and $b_f$ are learnable weights and biases. The Aggregate function here is the summation across all the columns of a matrix. This summation along with the relative difference in node properties, as in Eq. (6), preserves permutation-invariance and the structural properties (cognition of inter-node distances for example) 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 rotations of the graph.

### 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, choose the best task by calculating a probability value for each node. The first step is feeding the output from the encoder (as key-values) and information from the current state (as context) to a multi-head attention (MHA) layer. The context for the MHA layer in this experiment consist of the following seven features: 1) Current time; 2) Available range of the robot taking decision; 3) Current location of robot taking decision; 4) Current destination of other robots; 5) Available range for other robots. Figure 2 illustrates the structure of the decoder. Attention mechanism can be described as mapping a query ($Q$) to a set of key-value ($K, V$) pairs. The inputs, which are the query ($Q$), key ($K$), and values ($V$), are all vectors. The output is a weighted sum of the values $V$, weight vector is calculated by the compatibility function:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{d^V})V$$

where $d^k$ is the dimension of $K$ or $V$. In this work we implement a multi-head attention (MHA) layer in order to determine the compatibility of $Q$ with $K$ and $V$. The MHA implemented in this work is the same as the one implemented in [21] and [25]. As shown in [25] the MHA layer can be defined as:

$$\text{Multihead}(Q, K, V) = \text{Linear}(\text{Concat}(\text{head}_1 \ldots \text{head}_h))$$

where $\text{head}_i = \text{Attention}(Q, K, V)$ and $h$ is the number of heads. The feed-forward layer is to convert the output from the MHA layer to a dimension that can be taken in for the next process. The final softmax layer outputs the probability values for all the nodes. The next task to be done will be chosen based on a greedy approach, which means the node which has the highest probability will be chosen. The nodes which are already visited will be masked such that it will not be chosen in the future time steps of the simulation.

### 4 Learning Framework

Both the CCN-inspired encoder and the attention-based decoder consist of learnable weight matrices as explained in Sections 3.1 and 3.2. In order to learn these weight matrices, both supervised and unsupervised learning methods can be used. However, supervised learning methods are not tractable since the computational complexity of the exact ILP solution process required to generate labels. The complexity of the ILP formulation of the MRTA problem scales with $O(n^3m^2h^2)$, where $n$, $m$,
and $h$ represent the number of tasks, the number of robots, and the maximum number of tours per robot, respectively. Therefore, we use a reinforcement learning algorithm to conduct the learning.

**Learning Method:** In this work we implement a simple policy gradient method (REINFORCE) as the learning algorithm with greedy rollout baseline, which also enables us to compare the effectiveness of our method with that of [21]. For each epoch, two sets of data are used which are the training set and the validation set. The training data set is used to train the training model ($\theta_{CAM}$) while the validation data set is used to update the baseline model ($\theta_{BL}$). The size of the training data and the validation data used for this paper is mentioned in section 5.1. Each sample data from the training and validation data set consist of a graph as defined in Section 2.1. The pseudo code of the training algorithm for our architecture is shown in Alg. 1. It should be noted that the policy gradient method requires the evaluation of a cost function, which is defined to be same as in Eq. (1). **Policy:** We define the policy such that if the robot $r$ does not satisfy the constraints in Eqs. (3), it returns to depot (i.e., $a = 0$). Otherwise the robot $r$ runs the learnt CAM network and chooses the output (task) based on a greedy approach (selects a task with the highest probability value), as shown in Fig. 1.

5 Case Studies

We design and execute a set of numerical experiments to investigate the performance of our proposed learning-based algorithm over graph space (CAM) and compare it with an extended version of a state-of-the-art graph learning-based algorithm proposed by [21], so called attention-based mechanism (AM) approach. In order to test and validate the approaches, a set of experiments with varying task size and varying number of robots. The results are compared in terms of the cost function (Eq. (1)).

5.1 Design of Experiments & Learning Procedures

To evaluate the proposed CAM method, we define an MRTA case study with varying number of UAVs and 200 task (flood victims) locations. A 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 4 km, and a nominal speed of 10 km/h. We assume instantaneous battery swap is provided at depot location, which is used when UAVs return to depot since they were 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 learnt 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 in reference to the existing literature in the MR domain.

To perform learning and testing of the learned model, we proceed as follows: **Learning Phase:** We use a policy gradient reinforcement learning algorithm (REINFORCE with rollout baselines in this case) for learning the optimal policy. The learnable parameters in this architecture includes 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 models are tested for different cases of varying number of tasks and varying number of robots with each case having 100 random test scenarios from training data distribution.

**Modifications to AM:** The attention-based mechanism (AM) reported by [21] 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 our problem (for comparison with our CAM method), the AM method is adapted to a multi-robot setting. For this purpose, we make the following three changes to the AM method: (i) The node properties that are defined in Section 2.1 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).

6 Results and Discussion

**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 and plot their learning curve (convergence history), as shown in Fig. 3. As seen from this figure, the AM method took 3 epochs to reach its best cost value, but it was not able to find better values after the 3 epochs. On the other hand, the CAM method took 20 epochs to reach the best cost value of AM and after that CAM reached its final converged value in 24 epochs, which is a significantly better optimal cost function value ($f_{\text{cost,CAM}} = -0.266$) compared to AM ($f_{\text{cost,AM}} = -0.009$). Overall, AM has converged to a better
cost function values significantly faster in terms of number of training epochs, with the benefits
being more pronounced in terms of wall time. The higher cost of AM could be in part attributed to
directly implementing the encoding architecture of a transformer network [25]; 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 its relative differences in
local neighborhoods to capture structural information.

Computing time: Training and Execution: Based on the epoch information in section 5.1, the
average time to complete a training epoch was found to be 19.50 minutes (i.e., ~11.7 seconds per
sample) for CAM and AM. To report the execution time of the learnt model, the average cumulative
time for generating the entire sequence for the solution for both CAM and AM for different graph size
are summarized in Table 1. It should be noted that the average execution time per decision making
(assigning one task to a robot) is ~2 milliseconds. In addition, from Table 1, it can be seen that the
cumulative computing time for both AM and CAM is polynomially increasing w.r.t. # of tasks.

6.1 Generalizability analysis of CAM
The model has been trained on scenarios with
200 tasks and varying robot size (randomly a
robot size between 10 and 50 has been selected).
Then, 100 test scenarios have been generated
per robot-task size from the same distribution
of training scenarios. Here, six different combina-
tions have been considered: 50-task-5-robot,
50-task-10-robot, 100-task-10-robot, 100-task-
20-robot, 200-task-20-robot, and 200-task-40-
robot. Figure 4a shows the cost function (the
lower the better) for the unseen test scenarios for both AM and CAM. As it can be seen from Figs. 4a
that the proposed CAM approach outperforms the AM approach in all the test cases by achieving
better mean values of the cost function, respectively. The CAM approach performs significantly
better than AM in terms of the cost function for the lower task-to-robot ratio (here, 5). For the
task-to-robot ratio of 5, the performance of AM shows large variance (hence, lower robustness). The
negative value of the cost function in Fig. 4a shows a perfect completion rate (100%). Based on this
figure, the proposed CAM approach achieved a perfect completion rate for most of scenarios with the
task-to-robot ratio of 5. We posit that the generalizability benefits of the CAM architecture can be at-
tributed to its covariant compositional encoding, which is able to aggregate local node neighborhoods
while remaining agnostic to node ordering. The local structure of the graph is not only important for
effective decision-making, but also expected to be shared across various problems settings drawn
from the same distribution, thereby promoting generalizability of policies when adequately captured.

6.2 Scalability analysis of CAM
To investigate the scalability of the learnt model,
a new set of unseen test scenarios with larger
task size and robot size than the trained scenar-
arios are conducted. Figure 4b shows the perform-
ance of the trained model of CAM and AM
in terms of the cost function for (for 200 tasks)
four large case studies (i.e., 500-task-50-robot,
500-task-100robot, 1000-task-100-robot, and 1000-task-200-robot) with 100 randomly generated
scenarios for each case study. As shown in Fig. 4b, the proposed CAM method outperforms the AM
method in all cases, with a significant difference in the cost function for the case studies with the
task-to-robot ratio of 5. In the largest case (i.e., 1000-task-200-robot), the learnt model by CAM
achieved a perfect completion rate for most of scenarios (see the supplementary materials).

6.3 Comparative Analysis on CVRP
To demonstrate the versatility of the proposed CAM architecture, we train and test the CAM archi-
tecture and AM architecture (for comparison) on a variation of Vehicle Routing Problem (VRP),
which known as the Capacitated VRP (CVRP). The CVRP benchmark consists of N task locations,
where a vehicle is required to visit the locations and deliver packages such that minimizes a cost
function. Each location consists of a certain demand for the number of packages. The vehicle starts
from a depot, has a maximum capacity for the number of packages, and can have multiple routes to

Table 1: Average time taken (in seconds) to generate full solution for MRTA

<table>
<thead>
<tr>
<th># OF TASKS</th>
<th>AM</th>
<th>CAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.153s</td>
<td>0.150s</td>
</tr>
<tr>
<td>200</td>
<td>0.349s</td>
<td>0.340s</td>
</tr>
<tr>
<td>500</td>
<td>4.240s</td>
<td>4.470s</td>
</tr>
<tr>
<td>1000</td>
<td>19.221s</td>
<td>20.057s</td>
</tr>
</tbody>
</table>
7 Conclusion

In this paper, we proposed a new GNN architecture, called CAM, for a multi-robot task allocation problem with a set of complexities, including tasks with time deadline and robots with constrained range. This new architecture incorporates an encoder based on covariant node-based embedding and a decoder based on attention mechanism. A simple RL algorithm has been implemented for learning the features of the encoder and decoder. In addition, to compare the performance of the proposed CAM method, an attention-based mechanism approach (aka AM) has been extended to be able to handle a multi-agent combinatorial optimization problem (i.e., a multi-robot task allocation problem). In addition, the proposed architecture is operational over varying task size and swarm size (the upper bound is # of tasks and robots that model has been trained on). To evaluate the performance of the proposed architecture, and the extended version of AM are trained for 100 epochs and tested on 1,000 unseen case studies. Performance was analyzed in terms of the cost value and the completion rate. The new proposed architecture showed a better learning than AM by reaching a better cost value. Our primary method, CAM, outperformed the AM approach on both training and test scenarios by achieving better cost function value. The computational cost analysis showed that the proposed CAM model takes a few milliseconds to take a decision; hence, it is an excellent choice for online decision making (here, task allocation). The advantage of using local neighborhood information for node encoding can be seen in the scalability analysis on MRTA and CVRP, where CAM demonstrates superior performance when applying to graphs with larger number of tasks/nodes. As future work, it is necessary to improve the sample efficiency of the learning and incorporating more advanced learning algorithms.

Table 2: Comparison of average cost function VRP with other methods. The value inside the brackets is the average time taken to generate the entire solution.

<table>
<thead>
<tr>
<th># OF TASKS</th>
<th>AVG. COST FUNCTION (AVG. COMPUTING TIME)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GOOGLE OR</td>
</tr>
<tr>
<td>20</td>
<td>6.5 (2s)</td>
</tr>
<tr>
<td>50</td>
<td>11.3 (2s)</td>
</tr>
<tr>
<td>100</td>
<td>17.6 (5s)</td>
</tr>
<tr>
<td>200</td>
<td>21.3 (20s)</td>
</tr>
<tr>
<td>500</td>
<td>54.5 (20s)</td>
</tr>
<tr>
<td>1000</td>
<td>81.8 (200s)</td>
</tr>
<tr>
<td>2000</td>
<td>300.3 (1000s)</td>
</tr>
</tbody>
</table>
References


