Dynamic Coalition Formation and Routing for Multirobot Task Allocation via Reinforcement Learning

Weiheng Dai[1†], Aditya Bidwai[1], Guillaume Sartoretti[1]

***Abstract*— Many multi-robot deployments, such as automated construction of buildings, distributed search, or cooperative mapping, often require agents to intelligently coordinate their trajectories and form coalition over a large domain, to complete spatially distributed tasks as quickly as possible. We focus on scenarios involving homogeneous robots, but where tasks vary in the number of agents required to start them. For example, construction robots may need to collaboratively air-lift heavy objects at different locations (e.g., prefabricated rooms, crates of material/equipment), where the weight of each payload defines the required coalition size. To balance the total *travel time* of the agents and their *waiting time* (before task initiation), agents need to carefully sequence tasks but also dynamically form/disband coalitions. While simpler problems can be approached using heuristics or optimization, these methods struggle with more complex instances involving large task-to-agent ratios, where frequent coalition changes are needed. In this work, we propose to let agents learn to iteratively build cooperative schedules to solve such problems, by casting the problem in the reinforcement learning framework. Our approach relies on an attention-based neural network, allowing agents to reason about the current state of the system to sequence movement decisions that optimize short-term coalition formation and long-term task scheduling. We further propose a novel leader-follower technique to boost cooperation learning and compare our performance to conventional baselines in a wide variety of scenarios. There, our method closely matches or outperforms the baselines; in particular, it yields higher-quality solutions and is at least 2 orders of magnitude faster than exact solver in cases where frequent coalition updates are required.**

I. Introduction

Due to the often limited capabilities of individual robots, forming teams of robots that are able to work together on tasks provides substantial advantages for many practical applications such as aerial construction of 3D structures [1] (see Fig. 1), cooperative object transport [2], and inspection of industrial machinery [3]. These applications often consist of many spatially-distributed tasks with each of them requiring a specific number of agents to initiate and subsequently work together to complete the task. In such scenarios, agents can distribute themselves into teams (i.e., coalitions) to simultaneously work on different tasks in order to minimize the overall task completion time (i.e., makespan). For example, in scenarios involving aerial robotic construction, a group of UAVs might engage in lifting or building tasks across a large area. Depending on the task requirements (e.g., carrying different weights, bracing complex structures), agents may need to form coalitions of varying sizes. Furthermore, after the completion of a task, agents may need to adapt coalitions to proceed toward the remaining tasks based on their spatial locations and requirements. By forming coalitions dynamically, agents seek to strike a balance between traveling times among different tasks and waiting time before all agents are ready to initiate a task, especially for task-to-agent ratio.

Fig. 1. Multiple drones can dynamically form coalitions to finish spatially-distributed tasks in a cooperative construction scenario.

Allocating tasks to agents while minimizing some cost (the makespan in our case) is a challenging problem, formally known as Multi-Robot Task Allocation (MRTA). According to Gerkey et al.'s [4] taxonomy of MRTA problems, our work falls under the category of ST-MR-TA, where each robot can perform only one task at a time (ST), each task can require the cooperation of multiple robots (MR), and task allocation continuously happens across time (TA). Existing works on ST-MR-TA can be broadly classified into heuristic methods [5], [6], [7], market-based methods [8], and mixed-integer programming (MIP) [9], [10]. Many works like [5], [11] have explored coalition formation among agents in scenarios with high task-to-agent ratios. However, these algorithms often do not consider any minimum coalition size for task execution which is important for real-life applications. Other methods have been proposed to assign the best possible coalition and routes to robots [9], [6], [10], but these methods either primarily focus on one-shot assignment, or do not scale well to large instances as complexity increases exponentially to obtain optimal solutions, or even any solution, under time constraints. In scenarios that involve numerous tasks, time-critical situations, or even dynamically appearing tasks, having a fast solver that can guarantee high-quality solutions becomes of utmost importance for frequent replanning.

In this work, we propose a novel reinforcement learning (RL) approach through which agents (i.e., robots) can ob-

[1] Authors are with the Department of Mechanical Engineering, College of Design and Engineering, National University of Singapore. dai.weiheng@u.nus.edu, {adbidwai, mpegas}@nus.edu.sg

This work was supported by the Singapore Ministry of Education Academic Research Fund Tier 1.

tain decentralized policies to solve the ST-MR-TA problem. Specifically, agents learn to reason about their position, the status of all tasks, as well as the position and short-term intent of other agents, to make reactive movement decisions (i.e., which task to travel to and complete next). By choosing to converge to/diverge from tasks, agents naturally form coalitions and disband during task selection, iteratively constructing cooperative schedules. To train this collaboration in such a large multi-agent state-action space, we further propose a mechanism to reduce the decision complexity during training. Specifically, upon task completion, a random "leader" agent is selected from that coalition; this agent selects the next task, and all other "follower" agents are automatically driven to follow it to that new task (up to task requirements). Any agents left then select their next task in the same manner. In doing so, agent decisions are naturally linked, thus boosting cooperation learning. During policy execution, however, each agent uses its own learned policy to make individual decisions (no leader-follow mechanism), which our results show still yields similar performance. Using our approach, agents are able to collaboratively construct high-quality/near-optimal routes much faster than existing optimization methods relying on pre-training and low inference time, making our approach appealing for deployments in dynamic scenarios where frequent replanning might be necessary. We conduct experiments and compare our approach with state-of-the-art methods on randomly generated instances ranging from 10 to 40 agents and 20 to 200 tasks. Our results show that our approach can match or outperform an exact, MIP-based solver [9] while reducing computation times by at least two orders of magnitude, and outperforms heuristics methods in most cases where frequent coalition updates are required.

II. Related Work

A. Task allocation and task scheduling

Korsah et al. [12] expanded on the taxonomy introduced in [4] by incorporating interrelated utilities and constraints of robots and tasks in MRTA problems. This work falls under the category of ST-MR-TA with cross-schedule dependencies (XD) referred to as XD[ST-MR-TA], which indicates that the suitability of an agent for a particular task is not solely determined by its own schedule but also relies on the schedules of other agents within the system. Existing methods to solve XD[ST-MR-TA] problems can be generally categorized into integer programming [10], [13], [9], heuristics [7], [6], and auction-based methods [14], [8]. For example, Korsah et al. [10] first proposed a MIP-based anytime task allocation and scheduling planner for problems with cross-schedule dependencies, where different tasks may have synchronization constraints for initiation and time-window requirements for completion. Recently, Fu et al. proposed a scheduling framework utilizing MIP for heterogeneous robot teams [9], which addresses task decomposition, assignment, and scheduling problems while accounting for uncertainty in agents' abilities and task requirements. However, such MIP-based approaches do not scale well to large problems, as their computing times can reach/exceed tens of minutes on realistic instances, also limiting their application in dynamic scenarios. Different from exact solvers, Ferreira et al. [7] proposed a heuristic method in which they first decompose the task into robot actions and solve the problem as a Vehicle Routing Problem (VRP), thus existing VRP algorithms [15], [16], [17] could be applied. In addition, auction-based methods [14], [8] are often decentralized. However, agents tend to be competitive or greedy when placing their bids, which can result in degraded performance. For more complex cases beyond task decomposition and uncertainty, some studies address constraints like task precedence (i.e., order or priority of task execution) for ST-MR-TA problems [18], [19], [20], [21]. While beyond the scope of our current work, it is worth noting that our agents make reactive decisions while considering task executability. By dynamically changing task availability, agents have the potential to naturally address additional constraints such as task precedence.

B. Learning-based routing and scheduling

Traditional optimization methods, while efficient for solving small-scale problems and yielding exact/near-optimal solutions, encounter significant challenges when applied to large-scale instances as the computation time tends to experience exponential growth with problem size. Therefore, many recent approaches have proposed to rely on deep reinforcement learning (dRL) to solve such problems in a data-driven way for rapidly finding high-quality solutions. Narzri et al. first proposed an end-to-end VRP framework [22], which relies on learned attention mechanisms to solve such optimization problems. Cao et al. [23] further proposed an RL-based decentralized sequential decision-making method for the multiple traveling salesman problem (mTSP). These approaches exhibit great scalability and are able to generate near-optimal solutions through a Transformer-style network with very short computation time, particularly for large instances. Recently, Agrawal et al. [24] addressed task assignment in warehouse using an attention dRL approach, but only focused on single-agent tasks without cooperation. Similarly, some works [25], [19] proposed an RL-based scheduler to tackle scheduling problems under precedence constraints, which showed great generalization to different routing and scheduling problems. However, these works only consider agents who work as individuals, and challenges in coalition formation have not been tackled yet.

III. Problem formulation

We consider a team of n homogeneous agents $A = \{a_1, a_2, \ldots, a_n\}$ starting from a depot k_0. These agents need to complete m tasks $\{k_1, k_2, ..., k_m\}$ spatially distributed within a given 2D domain, and finally return to the depot k_0. Without loss of generality, the domain is normalized to the $[0, 1] \times [0, 1] \subset \mathbb{R}^2$ area. The depot and tasks are represented as a node set $K = \{k_0, k_1, k_2, \ldots, k_m\}$, each node $k_j \in K$ located at position (x_{k_j}, y_{k_j}). Each task k_j is associated a requirement c_{k_j} and duration w_{k_j}, where $c_{k_j} \in \mathbb{N}^+$ indicates the number of agents required to start it, and $w_{k_j} \in \mathbb{R}^+$

the time needed for such a coalition (once assembled) to complete the task. Agents are always in one of three states: 1) waiting for other agents to initiate a task, 2) executing a task, or 3) traveling from one task to another, which is performed through a straight line (Euclidean distance between tasks) at constant speed v. A solution is a set of individual agent tours $\Phi = \{\phi_{a_1}, \ldots, \phi_{a_n}\}$ that allows agents to complete all tasks based on their requirements and return to the depot, where each tour $\phi_{a_i} = (k_0, k_{i_1}, k_{i_2}, \ldots, k_0)$ is a set of non-repeating node (except for the depot). The makespan of a solution is the longest tour duration among all agents, where individual tour duration is calculated as the sum of the agent's task execution times, travel times, but also waiting times (between its arrival to a task and that of its last taskmate, which depend on the tours of other agents). Our objective is to find a solution with minimal makespan.

IV. Methodology

In this section, we frame our multi-robot task allocation as an RL problem and provide network and training specifics.

A. Multi-Robot Scheduling as an RL problem

We frame our problem as a decentralized sequential decision-making problem. Each agent a_i sequentially chooses its next task upon completion of its current one, thus iteratively constructing its own tour ϕ_{a_i} until all tasks have been completed and all agents return to the depot. Agents select their next tasks in a sequentially-conditional manner, considering the actions already taken by all other agents. To do so, after an agent makes a decision and travels to the next task, we directly update its location to that of the selected task, which informs the entire team of its decision. In our problem, agents often make decisions asynchronously, as the task completion time is rarely identical among parallelly running tasks. However, all agents in the same coalition will make decisions at the same time after completing their common task. In this case, agents make decisions in the order of their arrivals at that task.

To speed up the training and simplify decision-making, we introduce a leader-follower decision framework. In this framework, leaders are iteratively chosen to make decisions for a (sub)group of fellow agents. For instance, if a coalition of five agents completes a task, a leader will be randomly selected to decide the next task which requires a particular number of agents (e.g., three). Subsequently, a subset of agents in the original coalition (here, the leader and two other randomly picked agents) will move on to this next task, while the remaining agents (here, two) will choose their next task in the same leader-follower way, conditioned on the decisions made by the previous agents. By reducing the number of decisions during training, agents learn policies with lower variance and greater efficiency. In addition, learned policies can be utilized in a decentralized manner where each agent makes individual decisions.

B. RL formulation

1) Observation: The observation of agent a_i at time t is $o_i^t = \{G_i^t, A_i^t, M^t\}$, and includes three components: i) an augmented graph G_i^t of all tasks associated with a_i, ii) the current state of all agents A_i^t, and iii) a binary mask M^t. In our problem, we define tasks on a complete planar graph $G = (K, E)$, where the vertices K is the set of tasks (and depot) nodes and the edges are denoted as $(k_i, k_j) \in E, \forall k_i \neq k_j$, where $k_i, k_j \in K$. An augmented graph $G_i^t = (K_i^t, E)$ is used to describe the state of all tasks with respect to agent a_i. Each vertex in G_i^t is denoted as $k_{ji}^t = (x_{k_j} - x_{a_i}, y_{k_j} - y_{a_i}, c_{k_j}, w_{k_j}, c_{k_j} - g_{k_j})$, where $j \in \{0, 1, 2, \ldots, m\}$, (x_{k_j}, y_{k_j}) and (x_{a_i}, y_{a_i}) are the coordinates, c_{k_j} is the task requirement, w_{k_j} the task duration, and g_{k_j} the number of agents that currently have been assigned to this task. The depot is a special case, for which $k_{0i}^t = (x_{k_0} - x_{a_i}, y_{k_0} - y_{a_i}, 0, 0, 0)$. A_i^t contains information that pertains to the state of each agent with respect to the ego agent a_i with $a_{ji}^t = (x_{a_j} - x_{a_i}, y_{a_j} - y_{a_i}, b_{a_j}, d_{a_j}, e_{a_j}, n_{a_j}) \in A_i^t$, where $j \in \{1, 2, \ldots, n\}$, b_{a_j} is the time required for agent a_j to reach its next task (or 0 if the agent is currently at a task), d_{a_j} is the time agent a_j has spent waiting on the current task so far (0 if the agent is traveling/executing a task), e_{a_j} is the remaining time to complete agent a_j's current task (0 if traveling/waiting), while $n_{a_j} \in \{0, 1\}$ indicates whether the agent has already selected a task. The binary mask $M^t \in \mathbb{R}^{m+1}$ is shared among all agents, where each component represents whether the corresponding task has already been completed (1) or not (0).

2) Action: At each decision step t of agent a_i, given its current observation o_i^t, our decentralized neural network parameterized by θ outputs a stochastic policy $p_\theta(a \mid o_i^t) = p_\theta(\tau_t = j \in E \mid o_i^t)$ over all tasks, where finished tasks are filtered using the mask M_t. We sample agent a_i's action from $p_\theta(a \mid o_i^t)$ following a multinomial probability distribution during training, and select action greedily at inference time.

3) Reward: In our problem, the objective is to minimize the time needed for the team to complete all tasks and return to the depot (makespan). We simply define a sparse reward calculated at the end of the training episode as $R(\Phi) = -T$, where T is the makespan. In addition, we incorporate a timeout for each episode to prevent deadlocks, where deadlocks may occur as agents cannot finish an episode due to poor coordination (which happens earlier on during training).

C. Policy network

We use an attention-based network with a encoder-decoder architecture to learn the agents' policies $\pi_\theta(a \mid o_i^t)$, as depicted in Fig 2. The network parameters are shared among all the agents. We rely on a group of encoders to extract salient features, allowing agents to build context across the entire system. This shared context enhances communication among agents, enabling them to predict each other's intent and facilitate cooperation. Then, a group of decoders will leverage the learned representations from encoders (i.e., the features of position, task execution status, and travel times, etc.) and reason about potential benefits and costs associated with each task to help agents make informed decisions. Finally, a task selector outputs a probability for an agent to choose from an arbitrary number of tasks. We address scalability issues by leveraging the attention-based networks.

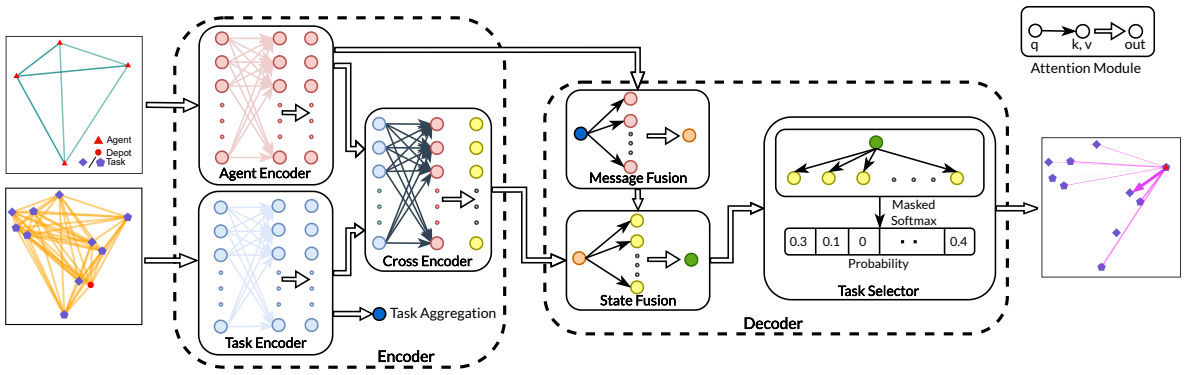

Fig. 2. Network structure. The raw information of tasks and agents is first projected into embeddings and then passed to a group of encoders to build context. In the decoder, this context is then fused with the average of task embedding thus generating a global glimpse of the full system. Finally, we use a task selector to output a probability distribution for an agent to choose the next task.

1) Multi-head attention with gated linear unit: The multi-head attention layer [26] is a vital component of our network. We further integrate a gated mechanism [27] into the attention layer to obtain better representations. In each layer, the z_{th} head takes query $h^q = [h_1^q, h_2^q, \ldots, h_n^q]^T$ and key-value pairs $h^{k,v} = [h_1^{k,v}, h_2^{k,v}, \ldots, h_n^{k,v}]^T$, transformed by three learnable matrices, to calculate an attention vector α_z:

$$Q_z, K_z, V_z = W_z^Q h^q, W_z^K h^{k,v}, W_z^V h^{k,v} \tag{1}$$

$$\begin{aligned} \alpha_z &= \text{Attention}(Q_z, K_z, V_z) \\ &= \text{Softmax}(Q_z K_z^T / \sqrt{d}) V_z, \end{aligned} \tag{2}$$

$$\text{MHA}\left(h^q, h^{k,v}\right) = \text{Concat}\left(\alpha_1, \alpha_2, \ldots, \alpha_Z\right) W^O, \tag{3}$$

where $W^Q, W^K \in \mathbb{R}^{Z \times \dim \times \dim'}, W^V \in \mathbb{R}^{Z \times \dim \times \dim'}$, $dim' = dim/Z$, and Z is the number of heads (in practice, $h = 8$). The output vector will be first passed to layer normalization, followed by a feed-forward layer with a gated linear unit [27]. When h^q and $h^{k,v}$ are the same, it is a self-attention layer as $\text{MHA}(h^q)$.

2) Encoder: In our work, we design a set of encoders including an agent encoder, task encoder, and task-agent cross encoder to integrate information, which allow each agent to build context about the full state of the system. For example, through learned attention mechanisms, the encoders implicitly help agents identify potential collaborators to form coalitions and complete a given task, and/or to predict the long-term impact of choosing a specific task on the other tasks. The raw inputs of the agent observation o_i^t are first passed into linear projection layers, yielding d-dimensional (in practice, $d = 128$) embeddings denoted as h_K and h_R for tasks and agents, respectively. These embeddings are further refined to establish correlations among each agent and each task through agent/task encoder as feature representations, $h'_A = \text{MHA}(h_A)$ and $h'_K = \text{MHA}(h_K)$, where h_A and h_K are queries fed into self-attention layers, respectively. To derive a simplified vector that captures all task information and summarizes the agent's current state, we calculate an aggregated graph embedding, represented as the mean of all task embeddings $\bar{h}_K = \frac{1}{m+1} \sum_1^{m+1} h_{k_i}$. Finally, we use a task-agent cross-attention encoder to build dependencies between tasks and agents as $h'_{KA} = \text{MHA}\left(h'_K, h'_A\right)$.

3) Decoder: We design our decoders to extract valuable features out of representations while preserving a global consideration of the task and agent dependencies and finally output a probability distribution for selecting the best next task. The aggregated graph embedding $\bar{h}_K$ is passed to message fusion layers with agent representations h'_A as $\bar{h}' = \text{MHA}(\bar{h}_K, h'_A)$. Then the output will go through state fusion layers to revise the current feature $\bar{h}'' = \text{MHA}(\bar{h}', h'_{KA})$. So far, we have computed enhanced features as a global glimpse that leverages the collective information from all tasks and agents. Finally, we use a simple single-head attention layer to calculate the attention score among the global glimpse and task-agent representations with a mask M^t. By removing already-completed tasks, we use the attention score as the probability of selecting each task.

D. Training

We train our policy using the REINFORCE [28] algorithm with greedy rollout baseline [23], [29]. Specifically, our method relies on two neural networks: a policy network, denoted as p_θ, which outputs a probability distribution over the all tasks based on the agent's current observation at each decision time, and a baseline network, denoted as p_b, which calculates a baseline reward by having each agent select its next task greedily (for a copy of the same exact instance/episode). This training algorithm helps reduce the variance of gradients and does not require explicit state value estimation. The gradient of the final loss function reads:

$$\nabla_\theta L = -\mathbf{E}_{p_\theta(\Phi)} \left[(R(\Phi) - R(\Phi_b)) \nabla_\theta \log p_\theta(\Phi) \mid o^{t=0} \right], \tag{4}$$

where the Φ and Φ_b are the group tours generated by the policy and the baseline networks, respectively. Furthermore, we frequently conduct a paired t-test to determine whether the baseline network should be replaced. We compare the performance of the policy network and the baseline network on a randomly generated test set, and then replace the weights of the baseline network with those of the policy network when the latter significantly outperforms the former (based on the condition $\text{T-Test}\left(R(\Phi), R(\Phi_b)\right) < \eta$, where $\eta = 0.05$ in practice). During training, agents rely on our leader-follower based coordination learning techniques; there, only the leader's decisions are used for training, which

we observed helps enhance sample efficiency and stabilize the training.

TABLE I

PARAMETERS OF TEST SCENARIOS

Trained Scenarios			Unseen Scenarios				
Task-to-agent	**m**	**n**	**Task-to-agent**	**m**	**n**	c_k	w_k
			30	100	10		
6	20	10	15	100	20		
15	50	10	10	100	30	[1,5]	[0,5]
7.5	50	20	7.5	100	40		
			15	200	40		

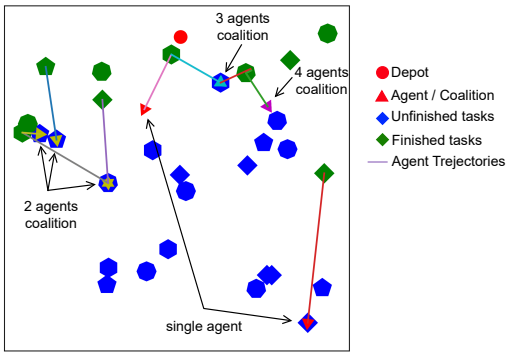

Fig. 3. An example of our task environment. The red circle represents a depot, triangles are agents or coalitions, and polygons with more than three vertices are spatially distributed tasks. The number of vertices n_v in a polygon indicates task requirements c_{k_j}, where $n_v = c_{k_j} + 3$. Agents' trajectories are shown in lines.

V. EXPERIMENTS

In this section, we detail our training/testing setups. We compare our method against the MIP-based approach and heuristic baseline in terms of solution quality, generalization, and computation time across a large range of scenarios with varying sizes. In doing so, we aim to provide an in-depth analysis of the strengths and weaknesses of our method under different scenarios. The code can be found on `https://github.com/marmotlab/DCMRTA.git`

A. Experimental Setting

We train our policy on scenarios with varying numbers of tasks (ranging from 20 to 50) and agents (ranging from 10 to 20). Specifically, we uniformly generate positions of depot and tasks (x_{k_i}, y_{k_i}) within our $[0,1] \times [0,1]$ domain. Each task requirement c_{k_i} is randomly drawn within $\{1, ..., 5\}$, while task duration w_{k_i} is sampled from a uniform distribution $[0, 5]$. All agents move at a constant speed $v = 0.2$. We further set a timeout $t_{max} = 100$ to terminate episodes where deadlocks occur and agents are unable to finish any more tasks. We train our policy using the Adam optimizer with a learning rate $r = 10^{-5}$ which decays by 0.98 every 2000 episodes. The baseline network is compared with the policy network every 1024 episodes.

To evaluate the performance of each method, we generate unseen test scenarios in a similar way to training data with parameters detailed in Table I, where each scenario comprises 50 instances. We define the average task-to-agent ratio as $\gamma = \frac{1}{n}\sum_{j=1}^{j=m} c_{k_j}$. We present a simulation of our experimental environment in Fig. 3.

TABLE II

AVERAGE PERFORMANCE COMPARISON ON DIFFERENT SCENARIOS.

	Method	Success Rate	Makespan	AWT	Time
n=10 m=20 $\gamma = 6$	CTAS-D	100%	**29.768 (± 3.338)**	2.586 (± 1.146)	300s
	OR-Tools	100%	40.249 (± 4.493)	2.953 (± 1.264)	20s
	RL-LF (ours)	100%	31.531 (± 3.243)	2.464 (± 0.697)	0.13s
	RL-IA (ours)	100%	33.1 (± 4.329)	3.964 (± 2.356)	0.33s
n=10 m=50 $\gamma = 15$	CTAS-D	70%	70.29 (± 10.182)	10.233 (± 4.131)	3600s
	OR-Tools	100%	77.888 (± 6.718)	6.014 (± 1.298)	20s
	RL-LF	100%	**65.92 (± 6.48)**	6.014 (± 1.298)	0.36s
	RL-IA	100%	67.583 (± 7.515)	8.787 (± 3.299)	0.82s
n=20 m=50 $\gamma = 7.5$	CTAS-D	100%	36.908 (± 3.754)	5.619 (± 1.767)	300s
	CTAS-D	100%	**34.353 (± 3.122)**	4.181 (± 0.989)	1800s
	OR-Tools	100%	41.967 (± 4.405)	0 (± 0)	20s
	RL-LF	100%	34.827 (± 3.251)	2.464 (± 0.697)	0.38s
	RL-IA	100%	35.697 (± 3.262)	2.798 (± 0.799)	0.88s
n=10 m=100 $\gamma = 30$	CTAS-D	0%	/	/	3600s
	OR-Tools	100%	145.364 (± 20.33)	18.133 (± 9.957)	20s
	RL-LF	100%	**124.875 (± 7.909)**	16.067 (± 3.652)	0.80s
	RL-IA	96%	129.712 (± 9.69)	20.201 (± 5.609)	1.93s
n=20 m=100 $\gamma = 15$	CTAS-D	62%	71.573 (± 6.731)	15.136 (± 4.059)	3600s
	OR-Tools	100%	74.096 (± 6.765)	0 (± 0)	20s
	RL-LF	100%	62.782 (± 4.164)	5.618 (± 1.328)	0.85s
	RL-IA	100%	**62.77 (± 4.12)**	5.543 (± 1.241)	1.69s
n=30 m=100 $\gamma = 10$	CTAS-D	98%	55.155 (± 5.828)	13.727 (± 3.938)	900s
	CTAS-D	100%	46.325 (± 3.876)	7.954 (± 1.915)	3600s
	OR-Tools	100%	48.944 (± 5.595)	0 (± 0)	20s
	RL-LF	100%	**44.242 (± 2.756)**	3.542 (± 0.618)	0.91s
	RL-IA	100%	44.582 (± 2.854)	3.692 (± 0.805)	1.97s
n=40 m=100 $\gamma = 7.5$	CTAS-D	98%	42.371 (± 4.566)	9.116 (± 2.963)	900s
	CTAS-D	100%	35.474 (± 2.427)	5.271 (± 1.299)	3600s
	OR-Tools	100%	35.575 (± 3.525)	0 (± 0)	20s
	RL-LF	100%	35.531 (± 2.327)	2.537 (± 0.489)	0.93s
	RL-IA	100%	**35.435 (± 2.252)**	2.634 (± 0.539)	2.01s
n=40 m=200 $\gamma = 15$	CTAS-D	0%	/	/	3600s
	OR-Tools	100%	**63.044 (± 6.988)**	0 (± 0)	20s
	RL-LF	100%	64.328 (± 3.699)	6.199 (± 0.746)	2.35s
	RL-IA	100%	64.742 (± 3.627)	6.302 (± 0.735)	4.94s

B. Comparison Analysis

We use the model trained following Section IV for all the test scenarios, under both decision strategies: leader-follower (RL-LF) and individual actions (RL-IA). In the RL-IA approach, each agent independently selects its next task at each decision step in a fully decentralized manner (no leaders). To evaluate the effectiveness of our method, we compare our approach to two conventional solvers, CTAS-D [9] and OR-Tools [15].

CTAS-D relies on a state-of-the-art MIP exact solver, Gurobi. It employs a two-step approach: it first optimizes the agent flow to complete all tasks, before rounding and splitting this agent flow into integers to obtain optimal tours. OR-Tools is an optimization tool that uses meta-heuristic algorithms to initially generate a solution, followed by refinement using local search to ultimately achieve a (near-)optimal solution. By partitioning agents into fixed-sized coalitions according to task requirements and durations, we use OR-Tools to minimize the maximum sum of travel and task execution times in each coalition, thereby generating a tour for each agent. CTAS-D is implemented in C++ and uses 8 CPU threads dedicated to optimization, while the OR-Tools and our variants are implemented in Python. We set a maximum computation time of one hour for CTAS-D and 20s for OR-Tools, with the goal of generating solutions of the highest possible quality for each method.

We evaluate all four methods on the same (randomly generated) test scenarios and report the success rate, makespan, average waiting time (AWT), and computation cost (Time) in Table II. The success rate indicates the percentage of instances for which the solver can generate a feasible solution

(higher is better). For makespan, lower is better. Average waiting time represents the average duration that an agent waits before it can start a task (lower is better).

1) Solution quality: For smaller-scale instances with lower task-to-agent ratio (e.g., $m = 20$, $n = 10$, $\gamma = 6$), CTAS-D performs best since these problems do not require high computation costs and can yield near-optimal solutions in reasonable times. However, our RL method achieves similar performance with only a 5.9% performance gap compared to CTAS-D in these scenarios. As the task-to-agent ratio increases, particularly for medium-scale and larger-scale problems (e.g., $m = 200/100/50$, $n = 10/20/40$, $\gamma \geq 15$), we observe a significant drop in performance and an increase in variance resulting from CTAS-D yielding solutions of inconsistent levels of optimality due to the time budget. Specifically, in cases with a large task-to-agent ratio ($\gamma = 30$), CTAS-D cannot even generate a feasible solution within one hour, while our RL method outperforms both OR-Tools and CTAS-D. These large task-to-agent ratio scenarios pose significant challenges because of the combination of a large number of tasks, high task requirements, and a limited number of available agents. In such cases, agents are required to dynamically form coalitions and disband in order to minimize traveling and waiting times, naturally increasing the optimization complexity for these baselines.

On the other hand, we observe that OR-Tools marginally outperforms the other approaches in scenarios with large numbers of agents and tasks (e.g., m=40, n=100/200). However, there again, our method can still generate high-quality solutions within a 2% gap. In these scenarios, dynamic coalition formation cannot significantly improve cooperation, because the number of agents is sufficient to optimally partition them into fixed coalitions. Thus, waiting times can be eliminated without any significant loss in overall performance, and task execution times are optimized.

In addition, our method achieves a considerably lower average waiting time compared with CTAS-D. By allowing agents to reason about the full state of the system to build *context*, they learn to consider not only their own states but also the intentions of other agents and the overall task completion status, enabling agents to sequence informed and reactive decisions. For example, agents can synchronize their arrival to common tasks, and in some cases, even if it entails traveling longer distances, agents can assist other waiting agents to avoid significantly longer waiting times/makespan.

2) Generalization: We train our policy only on small instances, our trained, decentralized policies can generalize to instances with any number of agents and tasks, relying on a transformer-based structure. From our results, we observe a linear relationship between the task-to-agent ratio γ and makespan. This suggests that our method scales linearly with respect to the instance size and achieve good generalization in all scenarios while maintaining the high solution quality.

Second, we further compare our RL-IA and RL-LF variants. These two variants perform very similarly, with only a very marginal edge for RL-LF over RL-IA (single-digit percent improvements at most). There, our results show that agents can make fully independent/decentralized decisions without any retraining or fine-tuning, without any remnant from the leader-follower technique used during training.

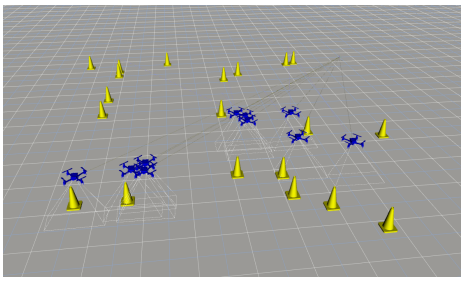

Fig. 4. A ROS Simulation of Crazyflie

Lastly, we evaluate our trained policy in large-scale instances involving up to 300 agents and 1000 tasks. Here, OR-Tools cannot deploy all agents, and CTAS-D cannot yield any solution within the time budget. However, we only observe slight performance decline in our method.

3) Computation time: Our method can generate high-quality solutions at least 2 orders of magnitude faster than the exact solver, and yet achieve similar performance. This significantly strengthens our method's ability to handle dynamic/uncertain environments, where frequent replanning may be necessary, e.g., as new tasks may randomly appear (or as task requirements may be uncertain/changing), or as agents may break down or be added to the team. We believe that this makes our approach more appealing for real-world deployments on robots under realistic settings.

C. Experimental validation

We validate our method in a multi-agent cooperative construction scenario using ROS as shown in Fig. 4, where agents (drones) must navigate between tasks (cones) to complete them. We further conduct experiments in an indoor environment (approximately 5m × 5m), where we deploy a few Crazyflie drones tasked with completing several spatially distributed tasks under onboard motion tracking. The video of our experiments is available as supplemental materials.

VI. Conclusion

This work introduces a decentralized reinforcement learning approach for efficiently solving the XD[ST-MR-TA] problems, where each agent iteratively and reactively constructs its tour based on global observations and a learned attention over all tasks and agents. We further stabilize cooperative learning by implementing a leader-follower technique, which reduces decision complexity and help attain the final policy that is fully decentralized and applicable to problems of arbitrary sizes. Based on evaluation results, we show our approach exhibits excellent performance in scenarios with large task-to-agent ratios, whereas in smaller-/larger-scale problems, we are still within a few percent gap of the state-of-the-art baselines.

In future work, we are interested in extending our approach to tackle more complex MRTA problems, such as heterogeneous agent teams and task precedence, which present more challenging scenarios for coalition formation and task decomposition problems.

REFERENCES

[1] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler, "Aerial robotic construction towards a new field of architectural research," *International journal of architectural computing*, vol. 10, no. 3, pp. 439–459, 2012.

[2] E. Tuci, M. H. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," *Frontiers in Robotics and AI*, vol. 5, p. 59, 2018.

[3] N. Correll and A. Martinoli, "Multirobot inspection of industrial machinery," *IEEE Robotics & Automation Magazine*, vol. 16, no. 1, pp. 103–112, 2009.

[4] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[5] L. Capezzuto, D. Tarapore, and S. Ramchurn, "Anytime and efficient coalition formation with spatial and temporal constraints," in *European Conference on Multi-Agent Systems*. Springer, 2020, pp. 589–606.

[6] X.-F. Liu, Y. Fang, Z.-H. Zhan, and J. Zhang, "Strength learning particle swarm optimization for multiobjective multirobot task scheduling," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[7] B. A. Ferreira, T. Petrović, and S. Bogdan, "Distributed mission planning of complex tasks for heterogeneous multi-robot teams," *arXiv preprint arXiv:2109.10106*, 2021.

[8] M. Krizmancic, B. Arbanas, T. Petrovic, F. Petric, and S. Bogdan, "Cooperative aerial-ground multi-robot system for automated construction tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 798–805, 2020.

[9] B. Fu, W. Smith, D. M. Rizzo, M. Castanier, M. Ghaffari, and K. Barton, "Robust task scheduling for heterogeneous robot teams under capability uncertainty," *IEEE Transactions on Robotics*, 2022.

[10] G. A. Korsah, B. Kannan, B. Browning, A. Stentz, and M. B. Dias, "xbots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 115–122.

[11] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings, "Coalition formation with spatial and temporal constraints," in *International Foundation for Autonomous Agents and Multiagent Systems*, 2010, p. 1181–1188.

[12] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[13] M. Koes, I. Nourbakhsh, and K. Sycara, "Constraint optimization coordination architecture for search and rescue robotics," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 3977–3982.

[14] X. Gallud and D. Selva, "Agent-based simulation framework and consensus algorithm for observing systems with adaptive modularity," *Systems Engineering*, vol. 21, no. 5, pp. 432–454, 2018.

[15] L. Perron and V. Furnon. Or-tools. Google. [Online]. Available: https://developers.google.com/optimization/

[16] J. Christiaens and G. Vanden Berghe, "Slack induction by string removals for vehicle routing problems," *Transportation Science*, vol. 54, no. 2, pp. 417–433, 2020.

[17] Y.-H. Lee, S. Leu, and R.-S. Chang, "Improving job scheduling algorithms in a grid environment," *Future generation computer systems*, vol. 27, no. 8, pp. 991–998, 2011.

[18] E. Bischoff, F. Meyer, J. Inga, and S. Hohmann, "Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 3949–3956.

[19] F. Deng, H. Huang, L. Fu, H. Yue, J. Zhang, Z. Wu, and T. L. Lam, "A learning approach to multi-robot task allocation with priority constraints and uncertainty," in *2022 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2022, pp. 1–8.

[20] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[21] H. Aziz, A. Pal, A. Pourmiri, F. Ramezani, and B. Sims, "Task allocation using a team of robots," 2022.

[22] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," *Advances in neural information processing systems*, vol. 31, 2018.

[23] Y. Cao, Z. Sun, and G. Sartoretti, "Dan: Decentralized attention-based neural network for the minmax multiple traveling salesman problem," *arXiv preprint arXiv:2109.04205*, 2021.

[24] A. Agrawal, A. S. Bedi, and D. Manocha, "Rtaw: An attention inspired reinforcement learning method for multi-robot task allocation in warehouse environments," 2023.

[25] J. Park, S. Bakhtiyar, and J. Park, "Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning," *arXiv preprint arXiv:2106.03051*, 2021.

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

[27] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[28] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[29] M. Fan, Y. Wu, T. Liao, Z. Cao, H. Guo, G. Sartoretti, and G. Wu, "Deep reinforcement learning for uav routing in the presence of multiple charging stations," *IEEE Transactions on Vehicular Technology*, 2022.