Heterogeneous Multi-robot Task Allocation and Scheduling via Reinforcement Learning

Weiheng Dai[1†], Utkarsh Rai[2], Jimmy Chiun[1], Cao Yuhong[1], Guillaume Sartoretti[1]

***Abstract*—Many multi-robot applications require allocating a team of heterogeneous agents to complete a given set of spatially distributed tasks as quickly as possible, such as search and rescue, area inspection/monitoring, and space exploration. We focus on tasks which can only be initiated when all required agents have arrived, such as detection involving sensor fusion or cooperative assembly requiring robots with different tools/skillsets. Robots dynamically form and disband teams based on diverse abilities needed for each task, which, however, can potentially result in extra idle (waiting) times at the task location. Robots (agents) should take into account the schedules of others to maximize their collective efficiency and reduce overall task completion time (makespan). Conventional methods such as mix-integer programming generally require centralized scheduling and long optimization time, which limits their potential for real-world applications. In this work, we propose a decentralized method that fully relies on reinforcement learning (RL) to train a generalized policy applicable to heterogeneous agents. To address the challenge of complex cooperation learning, we further introduce a constrained flashforward mechanism to guide/constrain the agents' exploration and help them make better predictions. Through an attention mechanism that reasons about both short-term cooperation and long-term scheduling dependency, agents learn to reactively choose their next tasks (and subsequent coalitions) to avoid wasting abilities and to shorten the makespan. We compare our method with state-of-the-art heuristic and mixed-integer programming methods, demonstrating its generalization ability and showing it closely matches or outperforms these baselines while remaining at least two orders of magnitude faster.**

I. Introduction

Multi-robot systems (MRS) are increasingly preferred over single-robot systems in many applications due to their ability to collaborate, resist robot failures, and tackle complex tasks that would be difficult or impossible for a single robot to accomplish. Deployments of MRS in applications like search and rescue [1], [2], space exploration [3] and healthcare [4] may require a team of heterogeneous robots with various motion, function, and perception abilities to tightly collaborate. For example, search and rescue [5] may involve ground and aerial robots working together to locate survivors through sensor fusion (e.g., camera and infrared) to avoid false positives, while collaborative assembly [6] may require robots with different manipulator/tools, e.g., mechanism to raise, gripper to hold, or tool to fasten. In this work, we focus on such scenarios, where robots need to sequentially execute a set of spatially distributed tasks requiring *tight cooperation*. That is,

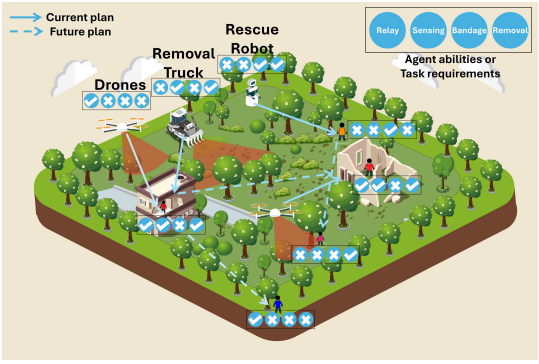

Fig. 1. Cooperative search and rescue in various environments. Vectors represent robots' abilities and rescue tasks' requirements, with each element indicating possession of a unique skill from relaying, sensing, bandaging, or obstacle removal, respectively. Schedules of robots are shown as arrows.

each task may require multiple robots with different *skills* and can only be started when all required robots have arrived. Therefore, to minimize the makespan of the mission, agents must reason about inherent cross-dependencies among different agents/tasks (i.e., correlations between agents' schedules) to form/predict the best *coalition* for each task, synchronize their arrivals to tasks to minimize waiting times, and carefully schedule the sequential/parallel execution of tasks.

Such problems are known as Multi-Robot Task Allocation (MRTA) problems, where the complexity exponentially grows with the number of robots, tasks, and required skills. Existing MRTA solvers can be mainly categorized into heuristic methods [7]–[9], mixed-integer programming (MIP) methods [10]–[13], and learning-based methods [14], [15]. State-of-art MIP-based methods [10], [12] typically rely on centralized optimization, which can derive exact solutions for each agent's route through one-shot assignment. Recently, learning-based methods [14], [15] relying on sequential decision-making have shown great potential to balance runtime and solution quality, particularly in large-scale scenarios involving time-critical situations or frequent replanning. However, all these methods suffer from the impact of *deadlocks* where all agents cannot complete their current tasks because they are waiting for each other. For optimization-based methods, deadlocks can make them struggle to find a good or even any solution in larger instances within reasonable time. For learning-based methods, deadlocks can significantly hinder training efficiency as agents can hardly learn any useful strategies from incomplete experiences (failed episodes).

In this work, we propose a novel reinforcement learning (RL) approach to obtain a decentralized cooperative policy enacted by all agents, which can generalize to arbitrary scenarios (with a single constraint on the maximum number of unique

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

[2] Author is with International Institute of Information Technology Hyderabad (IIIT)

skills among agents). By relying on attention mechanisms, agents learn to reason about the cross-dependencies among task locations, agents skills, and current status of other agents, as well as the status of all tasks, to make reactive decisions on which task to execute next. By doing so, agents naturally form/disband coalitions by converging on or diverging from tasks, iteratively creating cooperative schedules. To address the challenges posed by cooperation learning and complex cross-dependencies, we propose a *constrained flashforward mechanism* (CFM) to constraint tasks that agents can select and manipulate the order in which they make decisions during training, while ensuring the policy remains decentralized during inference time. This mechanism helps agents search for a suitable decision order and learn to make better predictions of each others' intent through sometimes acquiring their future actions, thereby mitigating the risk of deadlocks and improving the overall cooperation. As a result, our approach enables agents to iteratively and cooperatively build high-quality schedules much faster than existing optimization methods, making our approach particularly suitable for deployments in time-critical situations or requiring frequent replanning and complex coalitions. We empirically show that our policy can generalize to different scenarios without further tuning and scale up to 150 agents and 500 tasks, with up to 5 skills. Our results indicate that, in simple problems, our method can match or outperform an MIP-based exact solver [12] and a heuristic method [7] while being at least two orders of magnitude faster. In complex scenarios, we can solve instances where an exact solver cannot find any solution even when given hours of computing time.

II. Related work

A. Multi-robot Task Allocation and Scheduling

Multi-robot task allocation problems are categorized along three axes according to Gerkey's taxonomy [16]: single-task (ST) vs. multi-task (MT) robots, single-robot (SR) vs. multi-robot (MR) tasks, and instantaneous assignment (IA) vs. time-extended assignment (TA). Many well-studied problems fall into the ST-SR-TA category such as traveling salesman problem (TSP) [17] and vehicle routing problem (VRP) [18]. However, these problems only address scenarios where each task only requires one agent. Recently, research has increasingly shifted towards the ST-MR-TA category, which is the focus of our work. Methods like [19] used decentralized auction algorithms to address automated construction tasks with cross-dependencies, but they only focused on small teams and lacked replanning abilities. Other centralized approaches use mixed integer programming (MIP) [10], [12], [13], heuristics [9], [20], and evolutionary algorithms [7], [8]. Many of these works [9], [13], [20] do not consider tight cooperation, allowing agents to complete their portion of each task independently from others, thus neglecting the time-synchronization of agents at tasks. Recently, Fu et al. [12] introduced an optimization framework for task decomposition, scheduling, and coalition formation based on MIP, which also accounts for the risk of task completion and uncertainties in agent abilities. However, MIP-based methods often require long optimization times, particularly for large-scale problems. Other evolutionary methods rely on Ant Colony Optimization (ACO) [7] to solve Collab-MTSP instances that considers tight cooperation in TSP, generating solutions with quality comparable to MIP. Liu et al. [8] applied Particle Swarm Optimization (PSO) and local search to find near-optimal solutions considering multiple objectives and precedence. However, these methods focus on tasks requiring coalition of only a few agents and cannot handle difficult tasks with complex dependencies.

B. Deep Learning-based Task Allocation and Scheduling

Many recent approaches have focused on deep learning-based methods, which have proven effective for large-scale scenarios and long-term objectives. By shifting the computational cost from inference to training, these methods can find near-optimal solutions more quickly when deployed, making them suitable for real-world applications. Works such as [21], [22] utilize deep RL with Transformer-style networks to solve job shop scheduling and TSP in a data-driven manner and exhibit great scalability. To address spatial-temporal constraints and coordination of agents, [23], [24] framed the problem as a sequential decision-making problem and iteratively added new edges to the solution graph through attention-based networks. Despite the discussed strengths, these methods primarily focus on single-agent tasks without explicit cooperation among agents, thus they cannot organize agents into sub-teams/coalitions, which is crucial for handling tasks with complex requirements. Recent works such as [14], [15] tackle these limitations by using Graph Attention Networks (GATs) to learn a scalable decision policy for teams of cooperative agents. However, methods like [15] require a large amount of expert demonstration data to do imitation learning. RL methods like [14] only consider a homogeneous team where all the tasks require the same skill. In this paper, we propose a general decentralized decision-making framework for heterogeneous multi-robot task allocation problems that does not require any demonstration data.

III. Problem formulation

We use a *species-traits* model as defined in [12], [25], [26], where an MRS is described as a community of agents. In this model, each agent belongs to a species with a unique set of traits encoding the agent's skills. We consider a set of species $S = \{s_1, s_2, \ldots, s_{k_s}\}$ and a set of agents $N = \{a_1, a_2, \ldots, a_{k_n}\}$, where each species s_i contains n_{s_i} agents such that $k_n = \sum_{i=1}^{k_s} n_{s_i}$. Agents in the same species are identical, and each agent a_j in species s_i possesses a trait vector $\boldsymbol{c}_{a_j} = \boldsymbol{c}_{s_i} = [c_1^j, c_2^j \ldots, c_{k_b}^j]$, where $k_b \in \mathbb{Z}^+$ is the number of unique skills, $c^j \in \mathbb{N}$ represents the capability for each skill of agent a_j. When several agents form a coalition L, the trait vector of this coalition $\boldsymbol{c}_L$ will be the element-wise sum of individual agents' trait vectors.

Agents of the same species start at a common species depot. For convenience, S also represents the set of (different) species depots. These agents must complete all tasks $M = \{m_1, m_2, ..., m_{k_m}\}$ spatially distributed within a given 2D domain, and then return to their depot(s). Without loss of generality, the domain is normalized to a $[0, 1] \times [0, 1] \subset \mathbb{R}^2$

area, and the location of each task (and depot) is denoted as (x_i, y_i), where $i \in S \cup M$. Each task m_j is associated with a trait requirement $\boldsymbol{q}_{m_j} = [q_1^j, q_2^j, \ldots, q_{k_b}^j]$ and a execution duration t_{m_j}. Here, $q^j \in \mathbb{Z}^+$ suggests the minimum required capability of this skill to start the task, and $t_{m_j} \in \mathbb{R}^+$ represents the execution time needed for agents to complete the task once assembled at task location. To start a task, trait requirements must be satisfied by the assigned coalition, denoted as $\boldsymbol{c}_L \succeq \boldsymbol{q}_{m_j}$, where $\succeq$ is an element-wise greater-than-or-equal-to operator, and all assigned agents must be present at the task location for the entire execution duration. Though agents may arrive at the task location at different times, they must wait until all collaborating agents are present.

We then define all tasks and depots on a complete graph $\mathcal{G} = (V, E)$, where $V = S \cup M$ is the set of vertices, and E is the set of edges connecting all vertices denoted as $(v_i, v_j), \forall v_i \neq v_j$, where $v_i, v_j \in V$. Each vertex v_i represents the status (e.g., requirements, location, etc.) of a task (or a depot) and each edge contains a weight of Euclidean distance between two connected vertices. Similarly, we define an agent graph $\mathcal{G}_\mathcal{A} = (N, E^A)$, where each vertex represents agent's status.

Agents are always in one of three states during reactive scheduling: 1) waiting for other agents at a task location to initiate it, 2) executing a task, 3) traveling from one task (or depot) to another. Each agent's total working time is the sum of the time spent on each state from the start to the return to its depot. Our objective is to minimize the makespan, which is the maximum total working time among all agents. Formally, we define the solution as a set of routes for each individual agent as $\Phi = \{\phi_{a_1}, \ldots, \phi_{a_n}\}$, where $\phi_{a_i} = (v_s, v_{i_1}, \ldots, v_s)$ is a sequence of tasks that agent a_i visits or executes, v_s is the agent species depot, and each v_i is a task it (co-)executed.

IV. Methodology

In this section, we cast MRTA problem into a RL problem and illustrate details of our network and training.

A. Task Allocation via Sequential Decision-making

We formulate this problem as a decentralized sequential decision-making problem with global communication, allowing agents to broadcast their information (e.g., task status and agent's working condition). Starting from a depot, each agent independently chooses its next task upon completing the current one. This process iterates until all tasks have been completed and all agents have returned to their species depots, thus each agent can construct a route ϕ_{a_i}. We allow agents to select their next tasks in a sequentially-conditional manner such that they can consider all previous actions taken by the others. That is, each time an agent chooses its next task, we instantly update the observations of all other agents regarding that agent's position and the coalition status of that task (e.g., remaining task requirements) to inform them of the agent's decision. These tasks are rarely completed at the same time, which naturally lets agents at different tasks make decisions asynchronously at different decision steps. However, agents in the same coalition working and finishing simultaneously will select their next tasks at the same decision step. There, we randomize the order in which agents choose to enhance generalization for real deployments.

B. Stabilized Training via Constrained Action Ordering

In our problem, a task can only ever be in one of four states: *empty* (no agent at/en route to the task), *open* (task requirements not met by agent(s) at the task), *in-progress* (task requirements met by agent(s) at the task) or *completed*. Minimizing the overall makespan requires agents to complete as many tasks as possible in parallel. However, we observed that agents often tend to over-simplify this goal, by instead learning to *open* as many parallel tasks as possible, in the (vain) hope of completing all of them, despite not having enough agents to form that many suitable coalitions. This often results in *deadlocks*, where all agents are idly waiting at open tasks that will never be completed. It prevents the completion of the episode and robs the team from any useful experience to improve their policies, thus destabilizing the training process.

To reduce this undesirable (extreme) scattering of the agents, it is vital to guide the agents' exploration and prevent deadlocks by either constraining 1) the number of deciding agents, or 2) the agents' actions space. In homogeneous cases, our recent work proposed a leader-follower framework to reduce deadlocks during training [14]. We let a leader agent from each coalition make the next task decision and force the right amount of agents to follow the leader, to maximally fill the selected task. However, this approach cannot be naturally extended to our heterogeneous setting, where it is difficult to identify the "right" followers among heterogeneous agents. Differently, the approach in [7] employs a deadlock reversal step to release waiting agents and reassign them to the selected waiting task until the deadlock is undone. However, this approach relies heavily on backtracking agent decisions, which would call for frequent and costly jumps backward in time during training to adjust agents' actions, states, and rewards.

In this work, we propose a constrained flashforward mechanism to constrain the agents' action space and occasionally integrate future knowledge about other agents during training. This mechanism helps agents search for suitable decision order and learn to make better predictions of each others' intent through sometimes acquiring their future actions, thereby reducing the likelihood of deadlocks and improving overall cooperation. Our CFM consists of three components:

1) Maximum-open-task strategy: We restrict agents to select empty tasks when the number of open tasks has reached a maximal value. This prevents agents from trying to open too many tasks in parallel, to instead prioritize task completion, and thus reduces the likelihood of deadlocks.

2) Ability mask: This mask $\mathcal{M}$ only allows agents to choose tasks where they can help decrease the remaining requirements. For example, an agent with only an ultrasonic sensor cannot select a task currently only missing a camera.

3) Decision order: Most importantly, we manipulate the order in which agents make decisions, as it can drastically impact the formation of deadlocks. Given the two components mentioned above, an agent may sometimes find itself unable to choose any task based on its current circumstances (i.e., it

cannot contribute to any of the open tasks, and cannot open a new one). In this case, we allow the agent to postpone its decision until it can open a new task. Specifically, we first let all other deciding agents at this time step select their next task. We then cycle back to the problematic agent, in the hope that some of the open tasks are now filled, in which case the agent is now free to open a new empty task. However, if by the end of the decision step, this agent still has no valid task to select, we freeze its state for now and wait until the next decision step. There, we let all new deciding agents select their next task, and then cycle back again to this agent given the future actions selected by these new deciding agents. In doing so, we believe that this agent may learn to better predict the decisions that will be made by deciding agents in its future, thus implicitly learning to predict the intent of other agents in the team. We perform this maneuver until the agent is finally able to open a new empty task. However, to minimize waiting times, we actually record this agent's ultimate task decision as if it had been done at the original decision step, allowing the agent to already start its travel to its new task (as if its future decision had actually been done back when it first tried to make a decision but could not). In doing so, our flashforward process essentially searches for a suitable permutation of the agents' decisions that upholds the action selection constraints used to avoid deadlocks and minimize scattering. This process is used during training, but at inference time, we only keep the ability mask and disable the other two components.

C. RL Formulation

1) Observation: The observation of agent a_i at time t is $o_i^t = \{\mathcal{T}_i^t, \mathcal{N}_i^t, \mathcal{M}_i^t\}$, which consists of two vectors extracted from nodes on task graph and agent graph as $\mathcal{T}_i^t$, $\mathcal{N}_i^t$, respectively, and a decision mask $\mathcal{M}_i^t$.

$\mathcal{T}_i^t \in \mathbb{R}^{(k_m+1)\times k_g}$ is a vector representing the vertices' information of all the tasks and the species depot, where $k_g = 5 + 2k_b$ is the feature dimension. Each row indicates the status of a task (or a depot) as $[\bar{\boldsymbol{q}}_{m_i}^t, \boldsymbol{q}_{m_i}, x_{m_i} - x_{a_i}, y_{m_i} - y_{a_i}, t_{m_i}, d_{m_i}, f_i]$. $\bar{\boldsymbol{q}}_{m_i}^t = \boldsymbol{q}_{m_i} - \boldsymbol{c}_L^t$ that is the remaining trait requirements for task initiation. The position is calculated as relative coordinates w.r.t. the agent. d_{m_i} is the predicted traveling time based on the agent's moving speed and edge cost (Euclidean distance), and $f_i \in \{0, 1\}$ indicates whether the task is completed (1 indicates completed). For depots, we only keep the coordinates while the rest of the values are zero.

$\mathcal{N}_i^t \in \mathbb{R}^{k_n \times k_a}$ is a vector representing the working condition of all the agents, and $k_a = k_b + 6$ is the feature dimension. Each row presents the condition of an agent a_j w.r.t. observing agent a_i as $[\boldsymbol{c}_{a_j}, \boldsymbol{d}, (x_{a_j} - x_{a_i}, y_{a_j} - y_{a_i}), e_j]$, where $\boldsymbol{c}_{a_j}$ is the trait vector, $\boldsymbol{d} \in \mathbb{R}^{1\times 3}$ is a vector of agent's remaining execution time to complete its current task, remaining travel time to arrive at its next task, and waiting time from its arrival until the current task is in-progress. The binary value $e_j \in \{0, 1\}$ shows whether the agent's current task is open (0) or in-progress (1).

$\mathcal{M}_i^t$ is a binary mask indicating whether the task is open for the agent, obeying the decision constraint.

2) Action: Each time an agent completes its current task, our decentralized neural network, parameterized by θ, outputs a stochastic policy over all tasks based on the agent's observation as $p_\theta(a \mid o_i^t) = p_\theta(\tau_t = j \mid o_i^t)$, where $j \in \{i\}_0^{k_m}$ represents the indices of tasks $(1, \ldots, k_m)$ and the agent's depot (0). Completed/in-progress tasks and non-open tasks due to our CFM are filtered by the mask M_i^t. During training, we sample agent action from $p_\theta(a \mid o_i^t)$ following a multinomial probability distribution. For inference, we both tried to select actions greedily (argmax over this probability distribution), or at weighted-random (Boltzmann).

3) Reward: Our objective is to minimize the makespan, ensuring that agents cooperatively complete all tasks and return to their species depots as quickly as possible. We define a sparse reward calculated at the end of the training episode as $R(\Phi) = -T - W$, where T is the makespan and W represents the average ability redundancy for tasks as

$$W = \frac{1}{k_m} \sum_{i=1}^{k_n} w_i, \tag{1}$$

$$w_i = \begin{cases} \frac{\sum_{j=1}^{k_b} \mathrm{abs}(\boldsymbol{c}_L^{(j)} - \boldsymbol{q}_{m_i}^{(j)})}{\sum_{j=1}^{k_b} \boldsymbol{q}_{m_i}^{(j)}}, & \text{if } \boldsymbol{c}_L \succeq \boldsymbol{q}_{m_i} \\ \eta, & \text{otherwise,} \end{cases} \tag{2}$$

where η is user-defined value, in practice, we set $\eta = 10$ to keep T and W on the same scale. During training, a time-out T_{max} is set to end deadlock scenarios.

D. Policy Network

We design an attention-based network to build the context about the entire instance from a global perspective, enabling agents to make informed decisions and learn a policy $\pi_\theta(a \mid o_i^t)$ shared among all heterogeneous agents. The network follows an encoder-decoder structure as shown in Fig 2. The encoder builds contexts by aggregating the working conditions (e.g., positions, coalitions) of all agents and the status of all tasks (e.g., remaining requirements, distance). In this way, agents can implicitly exchange their intents and reason about dependencies among different tasks (and coalitions). The decoder then fuses the learned representations from the encoder to distill salient features and evaluate the future impacts and benefits of choosing each task. Finally, it outputs a probability distribution over all tasks. Our attention-based network could handle any number of agents and tasks during inference, providing flexibility and scalability in real-life deployments or even dynamic environments.

1) Multi-head attention with gated unit: Here we introduce the fundamental component of our network, the attention layer [27] with a gated mechanism [28] to learn better representations. We take a set of input queries h^q and key-value pairs $h^{k,v}$, then calculate an attention vector α with three learnable matrices W^Q, W^K, W^V as

$$Q, K, V = W^Q h^q, W^K h^{k,v}, W^V h^{k,v}, \tag{3}$$

$$\alpha_z = \mathrm{Attention}(Q_z, K_z, V_z) \tag{4}$$

$$= \mathrm{Softmax}(Q_z K_z^T / \sqrt{d}) V_z, \tag{5}$$

$$\mathrm{MHA}\left(h^q, h^{k,v}\right) = \mathrm{Concat}\left(\alpha_1, \alpha_2, \ldots, \alpha_Z\right) W^O, \tag{6}$$

In multi-head attention, each head computes its own attention vector, then the outputs from all heads are concatenated and

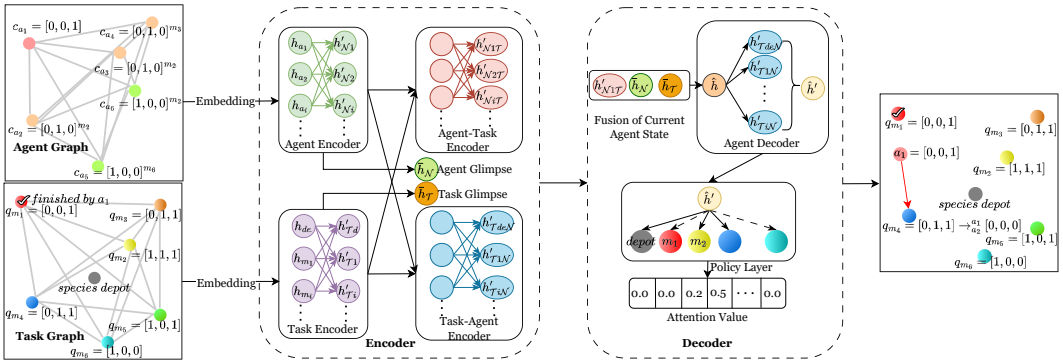

Fig. 2. Network structure used in this work. It follows an encoder-decoder structure where the status of agents and tasks such as trait vectors and trait requirements are first processed by a group of agent encoder, task encoder and cross encoder to build context and generate glimpses of the whole system. These features are then used by the decoder to output a policy (i.e., a probability distribution over all tasks).

multiplied with another learnable matrix, W_O, to calculate the final representation. Z is the number of heads (in practice, $Z = 8$). Finally, the output vector is passed to layer normalization and a forward layer with a gated linear unit.

2) Encoder: Our encoder consists of a task encoder, an agent encoder, and cross encoders. In the task encoder, node features $\mathcal{T}_i^t$ are first passed to a linear projection layer that maps them to d-dimension embeddings $h_\mathcal{T}$ (in practice, $d = 128$). A multi-head self-attention layer then encodes these embeddings $h'_\mathcal{T} = \mathrm{MHA}(\mathrm{h}_\mathcal{T}, \mathrm{h}_\mathcal{T})$ to build the context of all the tasks and learn the dependencies among them. In addition, we obtain a global glimpse $\bar{h}_\mathcal{T}$ to capture all the information by taking the average of node embeddings. Similarly, for the agent encoder, the vector features of agents $\mathcal{N}_i^t$ are processed the same way as task encoder to generate the context of all agents $h'_\mathcal{N}$ and a global agent glimpse $\bar{h}_\mathcal{N}$. These encoding contexts are then fed into two encoders to build enhanced agent-task context $h'_{\mathcal{NT}} = \mathrm{MHA}\left(h'_\mathcal{N}, h'_\mathcal{T}\right)$ and task-agent context $h'_{\mathcal{TN}} = \mathrm{MHA}\left(h'_\mathcal{T}, h'_\mathcal{N}\right)$, where task/agent encodings serve as queries and key-value pairs respectively.

3) Decoder: In the decoder, we first extract the feature of agent that is currently making decision from agent-task context $h'_{\mathcal{NT}}$. We concatenate this feature with the global glimpses of tasks and agents to build an agent current state. It is then passed to a linear layer to remain the same dimension d, as $h_{a_i} = \mathrm{MLP}(\mathrm{Concat}({h'_{\mathcal{NT}}}^{(i)}, \bar{h}_\mathcal{N}, \bar{h}_\mathcal{T}))$. This agent current state then goes through another (two layers of) multi-head attention with $h'_{\mathcal{TN}}$ and a binary mask $\mathcal{M}_i$ to enhance the representation as $h'_{a_i} = \mathrm{MHA}(h_{a_i}, h'_{\mathcal{TN}} \mid \mathcal{M}_i)$. Finally, this enhanced representation is used to calculate an attention score over $h'_{\mathcal{TN}}$, which is the probability of selecting each task.

E. Training

We train our policy using the REINFORCE [29] algorithm with a baseline reward inspired by POMO [30]. In combinatorial optimization problems, the reward is often a joint reward and sparse, making it challenging to stabilize the training. By relying on reinforcement learning with baseline rewards, we enable the agents to learn the policy through self-comparison. Different from POMO, where they choose different first action in traveling salesman problem and calculate an average reward for all the solutions, we here run the same episode multiple times and calculate the average reward. Because we sample actions during training, our policy will generate various trajectories for better exploration when the policy entropy is high, and then stabilize when the policy finally converges. We run an episode β times, and the advantages are calculated as

$$\mathbf{adv_i} = R(\Phi)_i - \frac{1}{\beta}\sum_{i=1}^{\beta} R(\Phi)_i. \tag{7}$$

In practice, we set $\beta = 10$, which we found high enough to help stabilize the training by mitigating the impact of low-quality experiences from episodes with deadlocks. This way, for each run of an episode, agents can learn according to how much their actions performed better or worse than expected from the advantage value. The gradient of the final loss function is defined as:

$$\nabla_\theta L = -\mathbf{E}_{p_\theta(\Phi)}\left[\mathbf{adv} \cdot \nabla_\theta \log p_\theta\left(\Phi\right) \mid o^t\right]. \tag{8}$$

V. Experiments

In this section, we detail our experimental setup for both training and testing. We conduct a series of experiments across different scenarios, ranging from small- to large-scale problems, single- to multi-skill agent, and simple to complex coalition requirements. We compare our method with an MIP-based method and heuristic methods to evaluate the performance in terms of solution quality, generalizability, and computation time. We analyze the performance of different methods and discuss our advantages.

A. Training Setup

We train our policy on randomly generated instances for each episode. For each instance, we sample a number of tasks ranging from 15 to 50 and a number of species ranging from 3 to 5. The positions of these tasks and species depots are uniformly generated in $[0, 1] \times [0, 1]$ domain. Each task has an execution duration sampled from $[0, 5]$ and a 1×5 trait requirement, where we have 5 unique skills and the requirement of each skill is an integer ranging from 0 to 2. For each species, there are 3 to 5 agents with the same trait vector, where each skill is represented by a binary value (0 or 1). This setup can be adapted to different real deployments by normalizing agents' abilities. To simplify the agents' kinematics, we assume all agents move at a constant speed of $v = 0.2$. Although the total number of skills is fixed at 5 due to the network design,

TABLE I
TESTING RESULTS FOR SA-BT SCENARIOS

SA-BT	Method	Success Rate	Makespan	AWT	Computation Time (s)
	TACO	**100%**	35.068 (± 5.857)	9.679 (± 4.905)	66.59
$k_n = 9$	SAS	**100%**	27.408 (± 3.918)	4.286 (± 1.78)	25.60
$k_s = 3$	CTAS-D	**100%**	**23.658 (± 2.918)**	**2.519 (± 1.068)**	600
$k_m = 20$	Greedy	**100%**	32.733 (± 3.371)	5.377 (± 1.572)	**0.11**
$k_b = 3$	RL(g.)	**100%**	29.002 (± 3.073)	4.125 (± 1.414)	0.43
	RL(s.10)	**100%**	27.193 (± 2.715)	3.687 (± 1.333)	4.33
	TACO	**100%**	28.86 (± 5.561)	9.327 (± 4.385)	120.87
$k_n = 25$	SAS	**100%**	27.329 (± 4.686)	5.686 (± 1.982)	40.37
$k_s = 5$	CTAS-D	**100%**	**16.488 (± 1.744)**	**1.988 (± 0.519)**	600
$k_m = 20$	Greedy	**100%**	21.879 (± 2.827)	3.163 (± 1.196)	**0.34**
$k_b = 5$	RL(g.)	**100%**	20.877 (± 2.467)	2.74 (± 0.954)	0.68
	RL(s.10)	**100%**	20.071 (± 2.219)	2.539 (± 0.767)	6.54
	TACO	**100%**	63.923 (± 7.051)	33.727 (± 5.949)	527.26
$k_n = 25$	SAS	**100%**	56.523 (± 7.221)	21.898 (± 4.973)	169.6
$k_s = 5$	CTAS-D	4%	\	\	3600
$k_m = 50$	Greedy	**100%**	43.964 (± 3.749)	8.771 (± 1.573)	**0.402**
$k_b = 5$	RL(g.)	**100%**	36.996 (± 3.1)	6.496 (± 0.952)	1.52
	RL(s.10)	**100%**	**35.477 (± 2.842)**	**6.042 (± 1.053)**	13.45
	TACO	**100%**	38.244 (± 7.029)	15.858 (± 6.704)	909.67
$k_n = 50$	SAS	**100%**	52.682 (± 5.738)	13.469 (± 3.503)	201.41
$k_s = 5$	CTAS-D	96%	**18.649 (± 1.922)**	**2.479 (± 0.54)**	3600
$k_m = 50$	Greedy	**100%**	27.153 (± 2.317)	3.406 (± 0.622)	**0.74**
$k_b = 5$	RL(g.)	**100%**	24.233 (± 2.071)	2.703 (± 0.494)	2.03
	RL(s.10)	**100%**	22.871 (± 1.45)	2.703 (± 0.515)	19.14

Note: the **bold** indicates the best performance and the underline highlights the second best performance. The other tables are showed in the same way.

TABLE II
TESTING RESULTS FOR SA-AT SCENARIOS

SA-AT	Method	Success Rate	Makespan	AWT	Computation Time (s)
$k_n = 9$	CTAS-D	40%	**47.166 (± 11.686)**	**13.414 (± 7.247)**	600
$k_s = 3$	Greedy	**100%**	64.449 (± 11.525)	22.895 (± 7.171)	**0.07**
$k_m = 20$	RL(g.)	**100%**	54.738 (± 10.073)	18.214 (± 5.915)	0.34
$k_b = 3$	RL(s.10)	**100%**	50.648 (± 8.616)	14.532 (± 4.492)	3.37
$k_n = 15$	CTAS-D	34%	58.665 (± 7.82)	21.045 (± 6.687)	1800
$k_s = 5$	Greedy	**100%**	74.179 (± 9.658)	31.493 (± 6.666)	**0.15**
$k_m = 20$	RL(g.)	91%	62.987 (± 11.596)	25.427 (± 12.836)	1.16
$k_b = 5$	RL(s.10)	**100%**	**57.454 (± 6.681)**	**19.689 (± 4.025)**	11.06
$k_n = 25$	CTAS-D	68%	**32.406 (± 8.347)**	8.986 (± 5.59)	600
$k_s = 5$	Greedy	**100%**	41.121 (± 4.421)	12.22 (± 2.627)	**0.218**
$k_m = 20$	RL(g.)	**100%**	36.091 (± 3.343)	9.107 (± 2.187)	1.09
$k_b = 5$	RL(s.10)	**100%**	33.439 (± 3.498)	**7.476 (± 1.686)**	10.2
$k_n = 50$	CTAS-D	0%	\	\	3600
$k_s = 5$	Greedy	**100%**	46.269 (± 3.669)	10.433 (± 1.373)	**1.185**
$k_m = 50$	RL(g.)	**100%**	39.427 (± 3.124)	7.741 (± 1.452)	3.08
$k_b = 5$	RL(s.10)	**100%**	**37.664 (± 2.798)**	**7.253 (± 1.196)**	29.8

TABLE III
TESTING RESULTS FOR MA-AT SCENARIOS

MA-AT	Method	Success Rate	Makespan	AWT	Ability Redundancy	Computation Time (s)
$k_n = 9$	CTAS-D	28%	**42.244 (± 7.541)**	**5.003 (± 3.433)**	**1.842 (± 0.084)**	600
$k_s = 3$	Greedy	**100%**	64.529 (± 13.293)	20.459 (± 8.546)	2.116 (± 0.254)	**0.08**
$k_m = 20$	RL(g.)	**100%**	54.066 (± 11.304)	17.472 (± 6.754)	2.007 (± 0.219)	0.3571
$k_b = 3$	RL(s.10)	**100%**	49.337 (± 11.126)	14.035 (± 6.378)	1.994 (± 0.214)	3.571
$k_n = 15$	CTAS-D	36%	35.916 (± 9.082)	**5.581 (± 2.364)**	**1.855 (± 0.171)**	1800
$k_s = 5$	Greedy	**100%**	45.126 (± 11.124)	12.75 (± 8.713)	2.085 (± 0.26)	**0.11**
$k_m = 20$	RL(g.)	**100%**	38.495 (± 9.985)	10.917 (± 7.225)	1.942 (± 0.238)	0.86
$k_b = 5$	RL(s.10)	**100%**	**35.911 (± 8.985)**	9.021 (± 6.038)	1.919 (± 0.257)	8.4
$k_n = 25$	CTAS-D	90%	**21.116 (± 5.439)**	**2.183 (± 0.98)**	**1.713 (± 0.185)**	600
$k_s = 5$	Greedy	**100%**	29.087 (± 5.828)	5.945 (± 3.523)	2.076 (± 0.25)	**0.173**
$k_m = 20$	RL(g.)	**100%**	25.625 (± 5.006)	4.727 (± 2.894)	1.943 (± 0.25)	0.76
$k_b = 5$	RL(s.10)	**100%**	23.674 (± 4.382)	4.044 (± 2.483)	1.903 (± 0.236)	7.6
$k_n = 25$	CTAS-D	0%	\	\	\	3600
$k_s = 5$	Greedy	**100%**	63.214 (± 14.069)	15.986 (± 9.778)	2.08 (± 0.244)	**0.529**
$k_m = 50$	RL(g.)	**100%**	50.158 (± 11.886)	12.712 (± 7.893)	1.911 (± 0.229)	1.634
$k_b = 5$	RL(s.10)	**100%**	**46.983 (± 10.711)**	**11.555 (± 7.12)**	**1.894 (± 0.232)**	16.34
$k_n = 50$	CTAS-D	14%	39.762 (± 5.752)	**0.614 (± 0.253)**	2.042 (± 0.099)	3600
$k_s = 5$	Greedy	**100%**	35.171 (± 5.701)	5.965 (± 2.827)	2.11 (± 0.254)	**0.983**
$k_m = 50$	RL(g.)	**100%**	29.624 (± 5.011)	4.732 (± 2.426)	1.917 (± 0.234)	2.343
$k_b = 5$	RL(s.10)	**100%**	**27.98 (± 4.581)**	4.788 (± 2.698)	**1.914 (± 0.236)**	23.43

we can still set unused skills as zero, thus generalizing to scenarios where the number of unique skills is fewer than 5. A timeout $T_{max} = 100$ is set to terminate deadlocks. We train our policy using the Adam optimizer with a learning rate $r = 10^{-5}$ that decays by 0.98 every 2000 episodes. To enable batch training, we apply zero-padding to inputs and train it based on maximum input size.

B. Comparison Results

We evaluate our trained model against other baselines on three scenarios: i) MA-AT ii) SA-AT iii) SA-BT. MA indicates the use of multi-skill agents possessing more than one skill, whereas SA involves single-skill agents, i.e., whose trait vector is one-hot. Additive tasks (AT) have cumulative requirements, where trait vectors of all agents in the coalition are element-wise summed to be compared to the task trait requirement, e.g., cooperative heavy payloads carrying, where the individual payload capacity of multiple agents can be summed to confirm they can carry a heavy load together. On the other hand, tasks have binary requirements (BT), meaning they can be successfully completed as long as each required ability is met by at least one agent in the coalition. We train our policy only on MA-AT scenarios, as the other instances could be seen as a subclass of MA-AT problems. In all experiments, we use the same trained model without further constraints or fine-tuning. To evaluate the effectiveness of our method, we compare our approach with CTAS-D [12], SAS [7], TACO [7] and our greedy heuristic.

CTAS-D relies on Gurobi to solve an MILP instance to find exact solutions. It first optimizes the flow of each species to complete all tasks, and then rounds the species flow to integers and determines the route of each agent. In our tests, we adapt the objective function to minimize the makespan and set an upper bound time for optimization of each instance. SAS and TACO are two heuristic methods based on ACO, where each ant represents a swarm coalition (SAS) or an individual agent (TACO), where pheromone trails are iteratively updated to improve the plan. Importantly, we note that SAS and TACO can only handle BT type problems. In addition, we implement a greedy heuristic algorithm with the same decision constraints as Section IV-B, where agents select the closest task to which they can contribute based on their trait vectors. For our RL method, we develop two variants: RL(g.), where agents greedily choose actions based on the learned policy (yielding a single solution), and RL(s.10), where we run each instance 10 times with Boltzmann action selection, and select the solution with lowest makespan among those.

We conduct experiments on unseen, randomly generated testing sets for each scenarios, with each set containing 50 instances. We report the success rate, average makespan, average waiting time (AWT), computation time, and ability redundancy in Table I-III. The success rate is the percentage of instances where the solver finds a feasible solution within the time constraint. AWT represents the average duration each agent wastes on waiting for others at tasks. Computation cost is the time to solve each instance. Ability redundancy is defined as Equation 1. Except for success rate, for all metrics, lower values are better. The reported average metrics **only considers solvable instances**. We also define a task-to-agent ratio $\frac{k_n}{k_m}$, which indicates the likelihood of deadlocks and difficulty of the problem, as a lower ratio with adequate agents relative to tasks typically reduces the chances of deadlocks.

TABLE IV
LARGE-SCALE MA-AT SCENARIOS

MA-AT	Method	Succese Rate	Makespan	AWT	Computation Time (s)
$k_m = 200, k_n = 50$ $k_s = 5, k_b = 5$	Greedy	**100.00%**	114.98 (± 24.951)	54.075 (± 12.532)	**9.94**
	RL(g.)	**100.00%**	82.698 ± 21.279	36.564 ± 9.727	7.32
	RL(s.10)	**100.00%**	**81.213 (± 20.657)**	**36.11 (± 9.401)**	75.2
$k_m = 500, k_n = 150$ $k_s = 10, k_b = 5$	Greedy	**100.00%**	86.776 (± 9.513)	11.588 (± 2.791)	62.6
	RL(g.)	**100.00%**	56.728 (± 8.694)	**8.694 (± 4.536)**	**55.4**
	RL(s.10)	**100.00%**	**56.093 (± 8.802)**	8.934 (± 4.553)	560.2
$k_m = 500, k_n = 150$ $k_s = 5, k_b = 5$	Greedy	**100.00%**	97.735 (± 20.005)	14.08 (± 4.786)	109.62
	RL(g.)	**100.00%**	67.35 ± 16.745	15.673 ± 10.816	**54.6**
	RL(s.10)	**100.00%**	**66.199 (± 16.475)**	**15.655 (± 10.784)**	560.5

C. Analysis

1) Solution quality: For SA-BT scenario, each task requires 1 to 5 skills, so completing the task only requires small numbers of agents to cooperate (smaller coalitions). The results are presented in Table I. In this scenario, CTAS-D generally performs the best given sufficient optimization time (10 mins to 1 hour). Our method ranks second among all the baselines, even though not being specifically trained for this scenario. We can find solutions with an optimality gap of the makespan less than 23% compared to exact solutions, even in the worst case, while being much faster and upholding a 100% success rate across all testing sets. Although CTAS-D can yield exact/near-optimal solutions, the probability of finding such solutions for large-scale problems with high task-to-agent ratios within the time constraints decreases significantly. On the other hand, both TACO and SAS can only generate low-quality solutions.

Table II shows results for SA-AT instances, scenarios requiring the most complex type of cooperation, which cannot be solved by SAS nor TACO. Here, task requirements are sampled in the same way during training (see Section V-A), but agents are single-skill. Our method overall outperforms CTAS-D in terms of makespan and success rate. Although the results of CTAS-D exclude timed-out instances, naturally giving it an advantage in terms of average performance reported, our method consistently achieves significantly higher success rate and yields high-quality solutions with lower variance.

We present MA-AT results in Table III. These scenarios involve further finding combinations of agents to satisfy task requirements while keeping a low ability redundancy. Our method significantly outperforms the Greedy baseline and is on par with CTAS-D in terms of makespan and ability redundancy. We maintain our 100% success rate over all instances, whereas CTAS-D shows a huge decline in performance as the task-to-agent ratio increases. Delving deeper into these results, we noticed CTAS-D relies heavily on using agents with many abilities and often leaves other agents idle, which explains its lower AWT. However, when such omnipotent robots are unavailable, CTAS-D struggles to find any solution within 1 hour. In contrast, our method, operating within seconds, can efficiently let agents form dynamic coalitions and keep a relatively low ability redundancy.

2) Generalizability: Even though our policy is trained on small-scale instances, once trained, it can generalize to different scenarios with any number of tasks and agents thanks to our network design. We also evaluate its scalability to very large-scale problems with up to 150 agents and 500 tasks, which are difficult, if not impossible, for conventional methods to solve. The results in Table IV and III indicate a near-linear relationship between the task-to-agent ratio and the makespan, indicating that our method can efficiently use the available agents to distribute the workload and maintain near-constant efficiency.

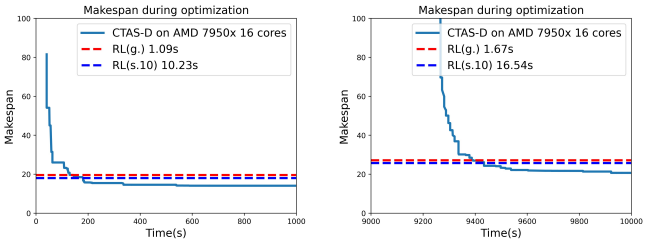

Fig. 3. Makespan of the current solution throughout CTAS-D's optimization process, compared to our final solutions, in MA-AT scenarios with 25 agents (5 species) and 20 tasks (left) and 30 tasks (right). Our approach quickly yields high-performance solutions (see legend), which remain drastically better than early solutions obtained by CTAS-D. A long refinement time is needed for CTAS-D's solution to finally slightly outperform us.

3) Computation Time: From our results, our RL-based approach is the fastest among all the methods (even though CTAS-D is implemented in C++, while our method was written in Python). The time taken by each of our agents to make one decision is determined by the size of inputs, which linearly increases with the problem scale. However, CTAS-D faces an exponential increase in computation time. As a result, our method generates solutions at least two orders of magnitude faster than CTAS-D, and at least ten times faster than heuristic ACO-based methods. Although CTAS-D can finally slightly outperform us given long enough time budgets, we conducted a case study in MA-AT scenarios with 25 agents and 20 or 30 tasks to illustrate how costly it is for CTAS-D to achieve similar performance as ours, as shown in Fig. 3.

4) Discussion: Across all these different results, we observe several key advantages of our method. First, although our approach works in a decentralized manner, it yields high-quality solutions comparable to those generated by centralized solvers, while our framework can easily scale to larger-scale instances. Second, complicated cooperation often leads to frequent deadlocks that often prevent conventional methods from finding good solutions, while our agents, using our CFM, can proactively avoid deadlocks by making informed, non-myopic decisions, selecting tasks with long-term benefits, and balancing traveling, working, and waiting times. Finally, our method can generalize to various scenarios and yield comparable results without further tuning. We believe our rapid response and excellent generalization make our method ideal for real-life, time-critical applications where long optimization times are infeasible, and frequent replanning is necessary to address unexpected events.

D. Experimental Validation

We validate our method in an indoor mockup of a distributed search mission (Fig. 4). There, we deploy six Crazyflies drones, separated into two species flying at two different altitudes, to execute six different tasks using the swarm framework from [31]. This demonstration shows our drones can sequentially visit all spatially distributed tasks (cones), dynamically forming coalitions based on task requirements, and finally returning back to their initial depot.

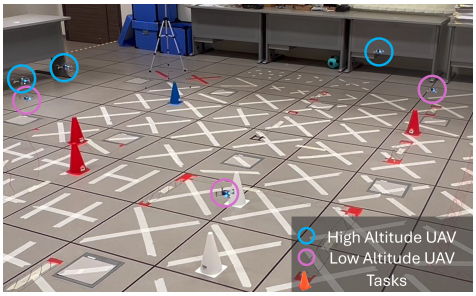

Fig. 4. Aerial robotic validation on a cooperative search mission, where agents of different species (shown as flying at different altitudes) dynamically form/disband coalitions to cooperatively search specific locations (tasks).

VI. Conclusion

In this work, we proposed a decentralized framework that leverages reactive sequential decision-making and distributed multi-agent RL to efficiently solve heterogeneous MRTA problems, with a specific focus on problems requiring complex coalition formation and tight cooperation. To tackle the challenge of frequent deadlock formation during early training, as well as complex cooperation learning, we introduced a novel constrained flashforward mechanism, which enables agents to efficiently explore rich collective behaviors and predict other's intent while keeping the learned policy decentralized, without significant computational overhead. Our evaluation results demonstrate that our method exhibits excellent generalizability across various scales and types of scenarios, without the need for fine-tuning. In particular, it closely matches the performance of exact solutions on smaller-scale, less complex instances, and significantly outperforms heuristic and MIP-based methods on larger-scale problems, while remaining at least 100 times faster!

Our future work will extend this framework to better handle additional constraints, such as tasks with specific time windows for completion, or with precedence constraints. There, we believe a learning-based approach can more effectively integrate such constraints into the agents' decentralized, cooperative decision-making, leading to improved long-term efficiency and easier real-life deployments.

References

[1] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, pp. 189–200, 2021.

[2] J. P. Queralta, J. Taipalmaa, Pullinen *et al.*, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *Ieee Access*, vol. 8, pp. 191 617–191 643, 2020.

[3] M. J. Schuster, M. G. Müller, S. G. Brunner *et al.*, "The arches space-analogue demonstration mission: Towards heterogeneous teams of autonomous robots for collaborative scientific sampling in planetary exploration," *IEEE Robot. Autom. Lett.*, vol. 5, pp. 5315–5322, 2020.

[4] O. Amir, B. J. Grosz, E. L. M. Law, and R. Stern, "Collaborative health care plan support," in *Proceedings of the 12th international conference on Autonomous agents and multiagent systems*. ACM, 2013.

[5] Z. Kashino, G. Nejat, and B. Benhabib, "Aerial wilderness search and rescue with ground support," *Journal of Intelligent & Robotic Systems*, vol. 99, no. 1, pp. 147–163, 2020.

[6] Z. Pan and Y. Yu, "Optimizing heterogeneous multi-robot team composition for long-horizon construction tasks: Time-and utilization-guided simulation," *Automation in Construction*, vol. 165, p. 105520, 2024.

[7] W. Babincsak, A. Aswale, and C. Pinciroli, "Ant colony optimization for heterogeneous coalition formation and scheduling with multi-skilled robots," in *2023 Int. Symp. Multi-Robot Multi-Agent Syst. (MRS)*. IEEE, 2023, pp. 121–127.

[8] 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.

[9] 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.

[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] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta, "Scalable and robust algorithms for task-based coordination from high-level specifications (scratches)," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2516–2535, 2021.

[12] 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.

[13] G. Neville, S. Chernova, and H. Ravichandar, "D-itags: a dynamic interleaved approach to resilient task allocation, scheduling, and motion planning," *IEEE Robot. Autom. Lett.*, vol. 8, no. 2, pp. 1037–1044, 2023.

[14] W. Dai, A. Bidwai, and G. Sartoretti, "Dynamic coalition formation and routing for multirobot task allocation via reinforcement learning," in *IEEE Int. Conf. Robot. Autom. (ICRA).*, 2024, pp. 16 567–16 573.

[15] W. Jose and H. Zhang, "Learning for dynamic subteaming and voluntary waiting in heterogeneous multi-robot collaborative scheduling," in *IEEE Int. Conf. Robot. Autom. (ICRA).*, 2024, pp. 4569–4576.

[16] 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.

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

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

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

[20] B. A. Ferreira, T. Petrović, and S. Bogdan, "Distributed mission planning of complex tasks for heterogeneous multi-robot systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, 2022, pp. 1224–1231.

[21] 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.

[22] 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.

[23] Z. Wang, C. Liu, and M. Gombolay, "Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints," *Autonomous Robots*, vol. 46, no. 1, pp. 249–268, 2022.

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

[25] A. Prorok, M. A. Hsieh, and V. Kumar, "The impact of diversity on optimal control policies for heterogeneous robot swarms," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 346–358, 2017.

[26] O. L. Petchey and K. J. Gaston, "Functional diversity (fd), species richness and community composition," *Ecology letters*, vol. 5, no. 3, pp. 402–411, 2002.

[27] 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.

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

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

[30] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 21 188–21 198, 2020.

[31] J. Chiun, Y. R. Tan, Y. Cao, J. Tan, and G. Sartoretti, "STAR: Swarm Technology for Aerial Robotics research," in *Proc. Int. Conf. Control, Autom. Syst. (ICCAS)*, 2024.