IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning Scheduling Policies for Multi-Robot Coordination with Graph Attention Networks

Zheyuan Wang[1] and Matthew Gombolay[1]

***Abstract*— Increasing interest in integrating advanced robotics within manufacturing has spurred a renewed concentration in developing real-time scheduling solutions to coordinate human-robot collaboration in this environment. Traditionally, the problem of scheduling agents to complete tasks with temporal and spatial constraints has been approached either with exact algorithms, which are computationally intractable for large-scale, dynamic coordination, or approximate methods that require domain experts to craft heuristics for each application. We seek to overcome the limitations of these conventional methods by developing a novel graph attention network-based scheduler to automatically learn features of scheduling problems towards generating high-quality solutions. To learn effective policies for combinatorial optimization problems, we combine imitation learning, which makes use of expert demonstration on small problems, with graph neural networks, in a non-parametric framework, to allow for fast, near-optimal scheduling of robot teams with various sizes, while generalizing to large, unseen problems. Experimental results showed that our network-based policy was able to find high-quality solutions for $\sim$90% of the testing problems involving scheduling 2–5 robots and up to 100 tasks, which significantly outperforms prior state-of-the-art, approximate methods. Those results were achieved with affordable computation cost and up to $100\times$ less computation time compared to exact solvers.**

I. INTRODUCTION

Advances in robotic technology are enabling the introduction of mobile robots into manufacturing environments alongside human workers. By removing the cage around traditional robot platforms and integrating dynamic, final assembly operations with human-robot teams, manufacturers can see improvements in reducing a factory's footprint and environmental costs, as well as increased productivity [1]. For human workspaces associated with final assembly, tasks need to be quickly allocated and sequenced (i.e., scheduled) among a set of robotic agents to achieve a high-quality schedule with respect to the application-specific objective function while satisfying the temporal constraints (i.e., upper and lower bound deadline, wait, and task duration constraints), as well as spatial constraints on agent proximity for safe and efficient collaboration with human workers. The problem of resource optimization is made difficult by the inter-coupled constraints requiring a joint schedule rather than allowing each agent to compute their work plans independently. Furthermore, scheduling decisions must be generated quickly and effectively in response to dynamic disturbances.

*This work was supported by the Office of Naval Research under grant GR10006659 and Lockheed Martin Corporation under grant GR00000509.

[1]Zheyuan Wang and Matthew Gombolay are with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA {pjohnwang,mgombolay3}@gatech.edu

Conventional approaches to scheduling typically involve formulating the problem as a mathematical program and leveraging commercial solvers or developing custom-made approximate and meta-heuristic techniques. Exact algorithms aim to find the optimal schedule based on enumeration or branch-and-bound, making them computationally expensive and unable to scale to large, real-time scheduling. Exact methods often rely on hand-crafted, "warm-start" heuristics unique to each application. Alternatively, heuristic approaches are lightweight and often effective; however, designing application-specific heuristics requires extracting and encoding domain-expert knowledge through interviews and trial-and-error-based research, a process which leaves much to be desired. Furthermore, accurately and efficiently extracting this knowledge remains an open problem [2].

To overcome the limitations of prior work, we build on promising developments in deep-learning-based architectures (i.e., graph neural networks) to learn heuristics for combinatorial problems. Analogous to the convolutional neural networks for feature-learning in images, graph neural networks are able to hierarchically learn high-level representations of graph structures through convolutions and backpropagation. Yet, these approaches have only been developed for simpler scheduling problems, e.g. the traveling salesman problem (TSP) [3], [4], in which the graph is fully apparent and edges are undirected. Conversely, multi-robot scheduling is a fundamentally different problem in which the graphical structure is a directed, acyclic graph with latent, disjointed temporal and spatial constraints that must be inferred.

In this paper, we develop a novel model, called RoboGNN scheduler, which is based on graph attention network (GAT) [5], to learn scheduling policies that reason about the underlying simple temporal network (STN) structure [6] and auxiliary constraints for multi-robot allocation and sequencing. We formulate scheduling as a sequential decision-making problem, in which individual robots' schedules are collectively, sequentially constructed in a rollout fashion. Our RoboGNN scheduler is non-parametric in both the number of tasks and the number of robots, meaning that the model can learn a policy from problem formulations of one size while still being able to construct schedules for task sets much larger than those seen during training. This non-parametricity is relatively unique in machine learning but is fundamental to scheduling problems as the needs of the manufacturer evolve minute by minute. A valuable benefit is that our approach can leverage imitation learning from small-scale problems in which supervised examples can be generated with exact solution methods, without the need for application-specific

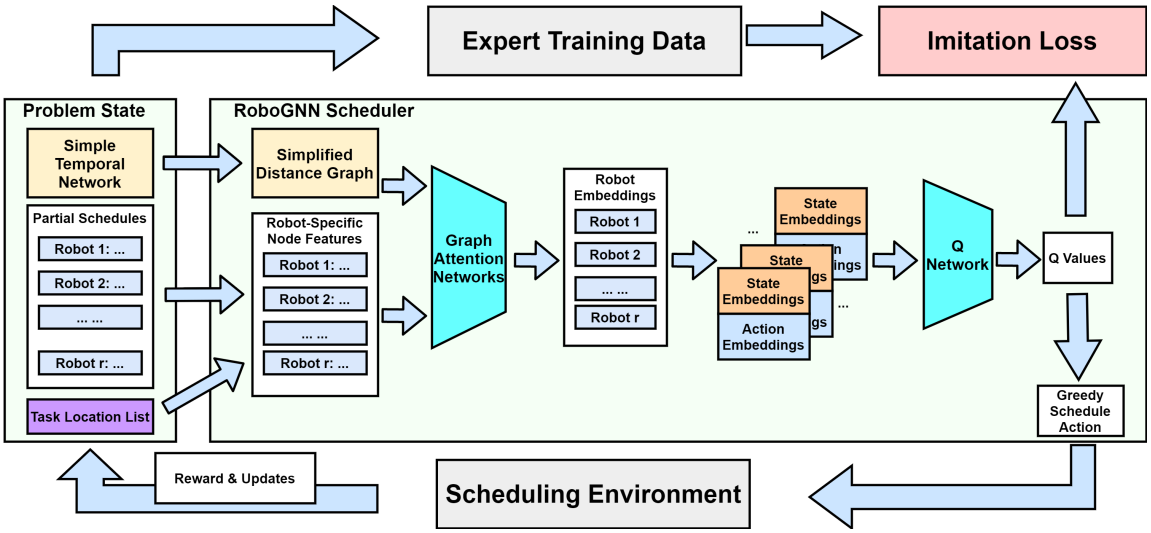


Fig. 1. The figure depicts the proposed framework, which incorporates graph attention networks and imitation learning for multi-robot scheduling. The RoboGNN scheduler uses a graph attention network, with robot-specific input node features constructed from partial schedules, to extract high level robot embeddings, and a separate Q network to evaluate discounted future rewards of state-action pairs for greedy schedule generation. The scheduler is trained with transitions generated from expert schedules using an imitation loss.

warm-starts, and still be applied on large-scale problems that are computationally intractable for exact approaches. We combine imitation learning with graph neural networks to learn a heuristic policy for scheduling, allowing for fast, near-optimal scheduling of robot teams. The combined framework is illustrated in Fig. 1. We demonstrate that our approach is able to find high-quality solutions for $\sim$90% of the testing problems involving scheduling two to five robots and up to 100 tasks with proximity constraints, which significantly outperforms prior state-of-the-art method. Moreover, those results are achieved with affordable computation cost and up to $100\times$ faster computation time versus exact solvers.

II. RELATED WORK

Task assignment and scheduling for multi-robot teams is an important class of problems with applications to manufacturing, warehouse automation, and pickup-and-delivery [7]. Our focus is on the single-task robots (ST), single-robot tasks (SR), time-extended assignment (TA) category with cross-schedule dependencies [XD] under the iTax taxonomy [8], modeling multi-robot construction of a large workpiece, e.g. fuselage. Formulating the problem into a mixed-integer linear program (MILP) yields MILP-based solution techniques with exponential complexity, leading to intractability for factory operations [9]. One popular way to accelerate the computation is to combine MILP and constraint programming (CP) methods into a hybrid algorithm using decomposition [10], but the performance may be limited by the decomposition quality. And it does not scale beyond a few agents and dozens of tasks.

Other hybrid approaches integrate heuristic schedulers within the MILP solver to achieve better scalability characteristics. Chen et al. incorporated depth-first search (DFS) with heuristic scheduling [11]. Additional approaches perform cooperative scheduling by incorporating Tabu search within an MILP solver [12] or by applying heuristics to abstract the problem to groupings of agents [13]. Researchers have also sought to apply metaheuristic techniques, including simulated annealing (SA) [14] and genetic algorithms (GAs) [15].

Some have pursued heuristic-learning for solving scheduling problems with approaches using policy [16] and Q-learning [17], [18]. Yet, the common limiting factor of these methods is that they are either not multi-agent or they do not handle the robust set of temporal and spatial constraints that we consider (i.e., cross-schedule dependences [XD]). Moreover, these methods depend on customized features to achieve satisfying results.

To address these limitations, we consider recent advances in graph neural networks (GNN) that extend deep neural networks to handle arbitrarily-structured data [21]. Recently, GNNs have been used to solve combinatorial optimization problems, including the traveling salesman problem (TSP) [3], [4], and other complex applications [19], [20]. In these prior works, the node embeddings obtained from GNNs are combined with machine learning algorithms to construct solutions. Table I summarizes recent work in this direction. Ours is the only that considers graphs with directed, weighted edges, which enables us to consider multi-robot coordination problems under temporal and spatial constraints, which use inherently directed, acyclic graphs often modeled as simple temporal networks (STNs) [22], [23], [24], [25]. To the best of our knowledge, we are the first to leverage GNNs in solving STN-based scheduling problems.

III. PROBLEM STATEMENT

We consider the problem of coordinating a multi-robot team in the same space, with both temporal and resource/location constraints. We describe its components, under the XD (ST-SR-TA) category of the widely accepted taxonomy proposed in [8], as a six-tuple $<\boldsymbol{r}, \boldsymbol{\tau}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{Loc}, z>$. $\boldsymbol{r}$ is the set of robot agents that we assume are homogeneous

TABLE I

RECENT WORK ON GRAPH NEURAL NETWORKS FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

Article	Graph Structure (edge type)	Attention Mechanism	Learning Method	Appication Domain
Khalil et al., 2017 [3]	Undirected, weighted	No	Q-learning	Minimum Vertex Cover, Maximum Cut and TSP
Kool et al., 2019 [4]	Undirected, unweighted	Yes	REINFORCE	TSP, Vehicle Routing Problem, Orienteering Problem, Prize Collecting TSP
Mao et al., 2019 [19]	Directed, unweighted	No	REINFORCE	Data processing cluster scheduling
Gasse et al., 2019 [20]	Undirected, weighted	No	Behavioral cloning	Four NP-hard problem benchmarks: set covering instances, combinatorial auction instances, capacitated facility location instances and maximum independent set instances
Ours	Directed, weighted	Yes	Imitation learning	Multi-robot coordination under temporal and spatial constraints

in task completion. $\boldsymbol{\tau}$ is the set of tasks to be performed. Each task τ_i takes a certain amount of time dur_i for a robot to complete, and its scheduled start and finish time are denoted as s_i and f_i, respectively (e.g., "task τ_i starts at 00:30, ends at 00:40, requiring 10 minutes" can be denoted as s_i = 30, f_i = 40, dur_i = 10). We introduce s_0 as the time origin and f_0 as the time point when all tasks are completed, so that the schedule has a common start and end point. $\boldsymbol{d}$ is the set of deadline constraints. $d_i \in \boldsymbol{d}$ specifies the time point before which task τ_i has to be completed. $\boldsymbol{w}$ is the set of wait constraints. $w_{i,j} \in \boldsymbol{w}$ specifies the wait time between task τ_i and task τ_j (e.g., "task τ_i should wait at least 25 minutes after task τ_j finishes" means $s_i \geq f_j + 25$). $\boldsymbol{Loc}$ is the set of all task locations. At most, one task can be performed at each location at the same time. Finally, z is an objective function to minimize that includes the makespan and possibly other application-specific terms.

A solution to the problem consists of an assignment of tasks to agents and a schedule for each agent's tasks such that all constraints are satisfied, and the objective function is minimized. We also include the mathematical program (MP) formation of our problem in (1)-(9). We consider a generic objective function, as application-specific goals vary. In Section VI, we consider minimizing the makespan (i.e., overall process duration), which would be $z = \max_i f_i$.

Here we introduce two types of binary decision variables: 1) $A_{r,i}$ = 1 for the assignment of robot r to task τ_i and 2) $X_{i,j}$ = 1 denotes task τ_i finishes before task τ_j starts. $\boldsymbol{L_{same}}$ is the set of task pairs (τ_i, τ_j) that use the same location and is derived from $\boldsymbol{Loc}$. We also have continuous decision variables $s_i, f_i \in [0, \infty)$ corresponding to the start and finish times of task τ_i, respectively. Equation (2) ensures that each task is assigned to only one agent. Equations (3)-(5) ensure that all the temporal constraints are met. Equations (6)-(7) ensure that robots can only perform one task at a time. Equations (8)-(9) account for task locations that can only be occupied by one robot at a time. In Section VI, we employ an exact benchmark (i.e., a mathematical program solver) to solve a linearized, mixed-integer form of these equations on small-scale problems to serve as expert demonstrations.

$$\min(z) \tag{1}$$

$$\sum_{r \in \boldsymbol{r}} A_{r,i} = 1, \forall \tau_i \in \boldsymbol{\tau} \tag{2}$$

$$f_i - s_i = dur_i, \forall \tau_i \in \boldsymbol{\tau} \tag{3}$$

$$f_i - s_0 \leq d_i, \forall d_i \in \boldsymbol{d} \tag{4}$$

$$s_i - f_j \geq w_{i,j}, \forall w_{i,j} \in \boldsymbol{w} \tag{5}$$

$$(s_j - f_i) A_{r,i} A_{r,j} X_{i,j} \geq 0, \forall \tau_i, \tau_j \in \boldsymbol{\tau}, \forall r \in \boldsymbol{r} \tag{6}$$

$$(s_i - f_j) A_{r,i} A_{r,j} (1 - X_{i,j}) \geq 0, \forall \tau_i, \tau_j \in \boldsymbol{\tau}, \forall r \in \boldsymbol{r} \tag{7}$$

$$(s_j - f_i) X_{i,j} \geq 0, \forall (\tau_i, \tau_j) \in \boldsymbol{L_{same}} \tag{8}$$

$$(s_i - f_j)(1 - X_{i,j}) \geq 0, \forall (\tau_i, \tau_j) \in \boldsymbol{L_{same}} \tag{9}$$

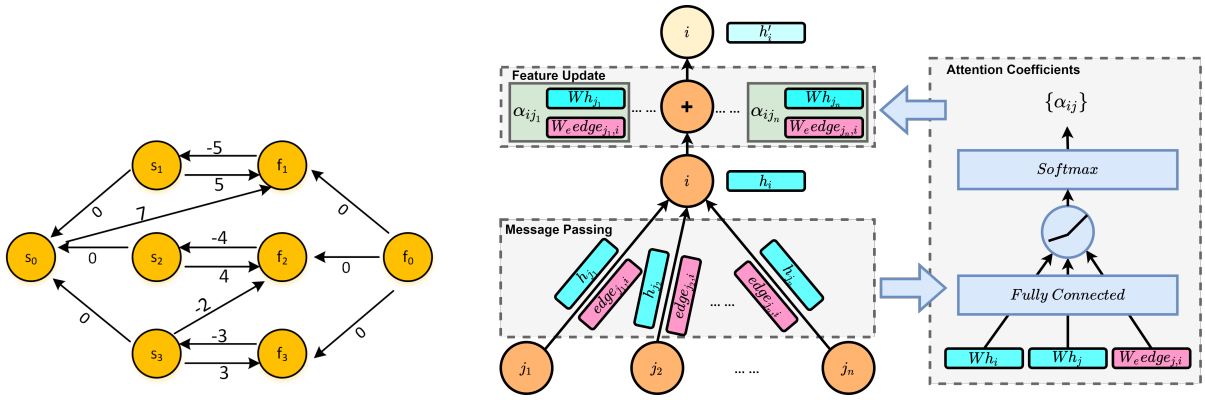
$$A_{r,i} \in \{0, 1\}, X_{i,j} \in \{0, 1\}, s_i, f_i \in [0, \infty)$$

IV. REPRESENTATION: GRAPH NETWORKS

Multi-robot task allocation and scheduling problems have been commonly modeled as STNs, because the consistency of the upper and lower bound temporal constraints can be efficiently verified in polynomial time. However, as we develop multiple agents, physical constraints, etc., we also have latent disjunctive variables that augment the graph to account for each agent being able to perform only one task at a time and for only one robot to occupy a work location at a time. This scheduling scenario is known as the Disjunctive Temporal Problem [26]. GNNs are an ideal choice for reasoning about STNs given their graphical nature. However we must expand on prior work to handle both the directed nature of these graphs, as well as the disjunctive component from multi-robot coordination in time and space. These extensions are a key contribution of this paper.

Modern GNNs capture the dependence of graphs via message-passing between the nodes, in which each node aggregates feature vectors of its neighbors from previous layers to compute its new feature vector. After k layers of aggregation, a node v's representation captures the structural information within the nodes that are reachable from v in k hops or fewer. Systems based on GNNs have demonstrated ground-breaking performance on tasks such as node classification, link prediction, and clustering [21]. Here, we make use of the graph attention layer (GAT) proposed in [5], which is a variant of a graph convolutional layer that introduces an attention mechanism to improve generalizability and modify its structure to make it suitable for representing an STN.

STN Preprocessing – In an STN, each task τ_i is represented by two event nodes: its start time node s_i and finish time node f_i. An example of an STN consisting of



(a) (b)

Fig. 2. Fig. 2(a) depicts an STN with start and finish nodes for three tasks, as well as placeholder start and finish nodes, s_0 and f_0. Task 1 has a deadline constraint and there is a wait constraint between task 3 and task 2. Fig. 2(b) depicts the forward pass of the adapted graph attention layer (left-hand side), which consists of two phases: 1) Message passing: each node receives features of its neighbor nodes and the corresponding edge weights; 2) Feature update: neighbor features are aggregated using attention coefficients; the right-hand side illustrates how attention coefficients are calculated.

3 tasks is shown in Fig. 2(a). For preprocessing purpose, we run Floyd Warshall's all-pairs-shortest-paths algorithm on the original STN to find the minimum distance graph [27]. Because in our problems, task duration is deterministic, it is possible to further remove the finish nodes f_i (except f_0) from the distance graph without losing information on the temporal constraints describing relations between each task. The resulted distance graph, which consists of only half the nodes of the original STN, is used by the graph attention network to learn high level robot embeddings.

Robot-Specific Node Features – While the graph attention network uses the same simplified distance graph to calculate the embeddings of each robot given a problem state, the difference lies in the set of input node features each robot uses, which we denote as robot-specific node features. Given all the partial schedules at the current step, we generate the initial input features of each node, with respect to a particular robot, as follows. The first 3 dimensions are the binary encoding denoting whether the corresponding task is scheduled to this robot, to other robots, or not scheduled. For example, [1 0 0] indicates the task is assigned to this robot, and [0 1 0] indicates the task is assgined to one of other robots. We use [1 1 0] as the first 3 features for the placeholder start and finish nodes of the entire schedule, s_o and f_o, respectively. The next dimension is the task duration. The next M dimensions are an one hot encoding of the location the task uses, where M is the number of locations. Thus, the input feature for each node is an (M+4)-dimensional vector. This set of input features is more expressive than that of prior approaches addressing the simpler TSP [3], [4] that only considered the (x,y) position of each node.

Structure Adaptation – The original graph attention network [5] is only able to incorporate undirected, unweighted graphs, yielding that model insufficient for scheduling problems in which temporal constraints are represented by the direction and weight of the edge between the two corresponding event nodes. As such, we make two adaptations for the message passing and feature update phases as shown in Fig. 2(b): 1) The message passing follows the same direction of the edge (i.e., only the incoming neighbors of a node are considered); 2) Edge information is also aggregated when updating the node feature, which is done by adding a fully-connected layer inside each GAT layer that transforms the edge weight $edge$ into the same dimension as the node feature using W_e. The output node feature $\vec{h}'_i$ is updated by (10), where $N(i)$ is the set of neighbors of node i, W is the weight matrix applied to every node, $\vec{h}_j$ is the node feature from the previous layer, and α_{ij} are the attention coefficients. To stabilize the learning process, we utilize multi-head attention [5], consisting of K independent GAT layers computing nodes features in parallel and concatenating those features as the output.

$$\vec{h}'_i = \mathrm{ReLU}\Big(\sum_{j \in N(i)} \alpha_{ij} (W\vec{h}_j + W_e edge_{ji}) \Big) \tag{10}$$

Attention Coefficients – The GAT layer computes the feature embedding for each node by weighting neighbor features from the previous layer with feature-dependent and structure-free normalization, which makes the network non-parametric in the number of tasks. The pair-wise normalized attention coefficients are computed as shown in Fig. 2(b) using (11), where $\vec{a}$ is the learnable weight, $||$ represents concatenation, and $\sigma()$ is the LeakyReLU nonlinearity (with a negative input slope of 0.2). Softmax function is used to normalize the coefficients across all choices of j.

$$\alpha_{ij} = \mathrm{softmax}_j \Big(\sigma \Big(\vec{a}^T \Big[W\vec{h}_i \Big\| W\vec{h}_j \Big\| W_e edge_{ji} \Big] \Big) \Big) \tag{11}$$

Given an STN and a set of robot-specific node features, the graph attention network, constructed by stacking several GAT layers, outputs the embeddings of each node. Then the embedding of the corresponding robot is obtained by averaging over all node embeddings.

V. LEARNING SCHEDULING POLICIES

We first formulate scheduling as a sequential decision-making problem, in which individual robots' schedules are collectively, sequentially constructed in a rollout fashion. At each decision step, the policy picks a robot-unscheduled task pair and assigns that unscheduled task to the end of

that robot's schedule. This step repeats until all tasks are scheduled. Next, we formalize the problem of constructing the schedule as a Markov decision process (MDP) using a five-tuple $<x_t, u, T, R, \gamma>$that includes:

- States: As shown in Fig. 1, the problem state x_t at step t consists of the STN encoding the temporal constraints, all robots' partial schedules constructed so far, and the task location list. As both location information and partial schedules are included as robot-specific node features, we approximate state embedding, h_x, by averaging over all robot embeddings.
- Actions: Action $u = <\tau_i, r_j>$ implies appending task τ_i into the partial schedule of robot r_j, where τ_i is from the set of unscheduled tasks. The action embedding, h_u, is approximated by the node embedding of start time node s_i of τ_i, calculated with robot-specific node features with respect to r_j.
- Transitions T: Transitions correspond to adding the edges associated with the action into the STN and updating the partial schedules. In the MP formulation, when $u = <\tau_i, r_j>$ is taken, besides setting $A_{r,i} = 1$, we add the following two terms before updating the equations: 1) $s_i \leq s_k, \forall \tau_k \in \{unscheduled\}$; 2) $X_{i,m} = 1, \forall \tau_m \in \{unscheduled | (\tau_i, \tau_m) \in \boldsymbol{L_{same}}\}$.
- Rewards R: The immediate reward of a state-action pair is defined as the change in makespan of all the scheduled tasks after taking the action. As such, the cumulative reward of the whole schedule generation process equals the final makespan of the problem (when feasible solutions are found). We divide the change by a discount factor $D > 1$ if the next state is not a termination state. The reward is multiplied by -1.0, as we are minimizing the total makespan. A large negative reward M_{inf} is returned if the action results in an infeasible schedule in the next state. As a result, the goal of the policy is learning to construct the optimal schedule.
- Discount factor γ

We aim to learn a policy that schedules tasks and agents following the decision-making process. To enable imitation learning with expert demonstrations, we define an evaluation function, $Q(x_t, u_t)$, that calculates the total discounted reward of taking action u_t at step t. Then, our goal is to approximate the evaluation function with a neural network $\hat{Q}_\theta$ parameterized by weights θ. This function approximator, as show in Fig. 1 under the name "Q network", consists of two fully-connected layers. It takes as input the concatenation of state embedding h_x and action embedding h_u and outputs a score estimating the total rewards of performing action u. As a result, we obtain a greedy policy $\pi := \text{argmax}_u \hat{Q}_\theta(h_x, h_u)$ that selects a task τ_i and a robot r_j at each step to maximize the Q value with corresponding action.

Because we are dealing with homogeneous robots, and the objective is minimizing makespan, we modify the schedule generation process in an opportunistic manner, which uses time-based rollout, to avoid possible delay among different robots' schedules. More specifically, starting from $t = 0$ (here t refers to time points instead of decision steps), at each time step, the policy first collects all the available robots not working on a task into a set $\boldsymbol{r_{avail}} = \{r_j | r_j \text{ is available}\}$. Then, $\forall r_j \in \boldsymbol{r_{avail}}$, the policy tries to assign τ_i using $\tau := \text{argmax}_\tau \hat{Q}_\theta(h_x, h_u)|_{r=r_j}$.

A. Imitation Learning

Although obtaining optimal solutions of large-scale scheduling problems is computationally intractable, it is practical to optimally solve smaller-scale problems with exact methods. Furthermore, we can use these exact methods to automatically generate application-specific examples for training an imitation learning algorithm without the need for the tedious, non-trivial task of developing application-specific heuristics to warm-start the solver. Finally, we typically have access to high-quality, manually-generated schedules from human experts that currently manage the logistics in manufacturing environments. We believe that exploiting such expert data to train the scheduling policy can greatly accelerate the learning process [28].

We aim to leverage such data by training the network on expert dataset D_{ex} that contains schedules either from exact solution methods or the domain experts. For each expert solution, we arrange the scheduled tasks by task start time in ascending order and decompose them into state-action pairs following our schedule generation process. For each transition, we directly calculate the total reward from current step t until termination step n using $R_t^{(n)} = \sum_{k=0}^{n-t} \gamma^k R_{t+k}$ and regress $\hat{Q}_\theta$ towards this value as shown in (12), where the supervised learning loss, L_{ex}, is defined as the Euclidean distance between the $R_t^{(n)}$ and our current estimate based on state embedding h_x and embedding of the action selected by the expert $h_{u,ex}$.

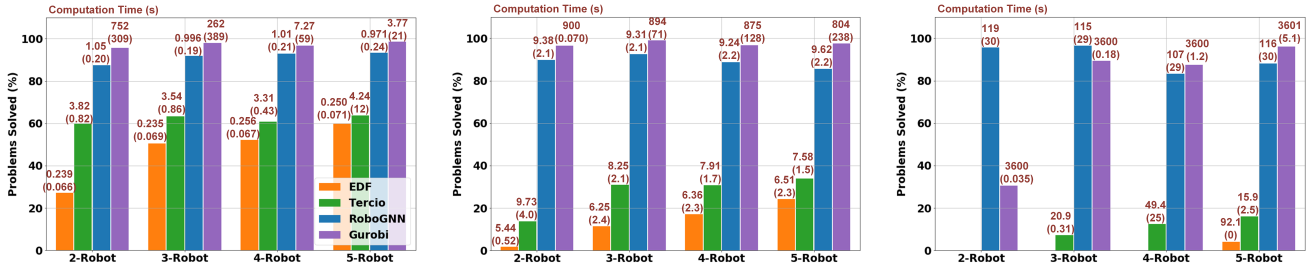
$$L_{ex} = \left\| \hat{Q}_\theta(h_x, h_{u,ex}) - R_t^{(n)} \right\|^2 \tag{12}$$

To fully exploit the expert data, we ground the Q values of actions that are not selected by the expert to a value below $R_t^{(n)}$ using the loss shown in (13), where $h_{u,alt}$ is the action embedding associated with alternate actions not chosen by the expert, q_o is a positive constant used as an offset, and N_{alt} is the number of alternate actions at step t.

$$L_{alt} = \frac{1}{N_{alt}} \sum \Big\| \hat{Q}_\theta(h_x, h_{u,alt}) - \min(\hat{Q}_\theta(h_x, h_{u,alt}), R_t^{(n)} - q_o) \Big\|^2 \tag{13}$$

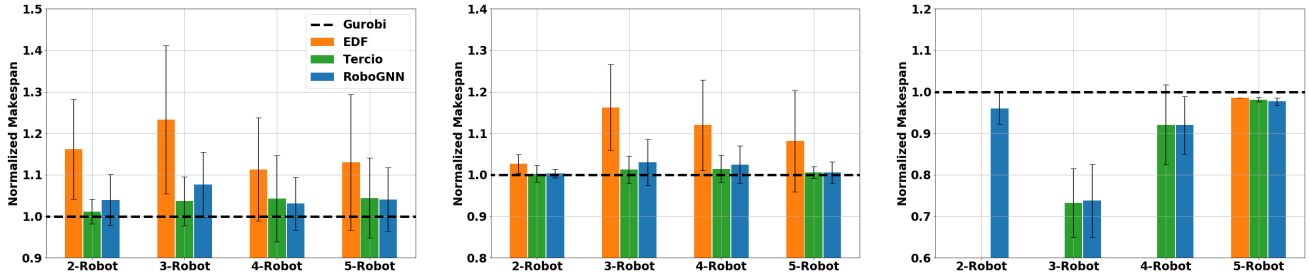
Consequently, the gradient propagates through all the unselected actions that have Q values higher than $R_t^{(n)} - q_o$. We select q_o empirically during training. Note the difference from [28] in that they only train on the unselected action with the max Q value. Combing (12) and (13), we calculate the total loss via (14), where L_2 is the L2 regularization term on the network weights, and λ_1, λ_2 are weighting parameters assigned to different loss terms empirically.

$$L_{sup} = L_{ex} + \lambda_1 L_{alt} + \lambda_2 L_2 \tag{14}$$



(a) (b) (c)

Fig. 3. Proportion of problems solved for multi-robot scheduling: (a) small problems (16–20 tasks); (b) medium problems (40–50 tasks); (c) large problems (80–100 tasks). Results are grouped in number of robots. Mean and standard deviation of computation times (in parenthesis) for each method is shown above each group's bar.



(a) (b) (c)

Fig. 4. Normalized makespan score for multi-robot scheduling: (a) small problems (16–20 tasks); (b) medium problems (40–50 tasks); (c) large problems (80–100 tasks). Results are grouped in number of robots. A smaller (normalized) makespan is better.

VI. EXPERIMENTAL RESULTS

We evaluate the performance of our model on randomly-generated problems simulating multi-agent construction of a large workpiece, e.g. an airplane fuselage. We generate problems involving a team of robots (team size ranging from two to five) in different scales: small (16–20 tasks), medium (40–50 tasks) and large (80–100 tasks), with both temporal constraints and proximity/location constraints (i.e., no two robots can be in the same location at the same time). For each problem, team size is randomly selected from interval $[2, 5]$. Task duration is generated from a uniform distribution in the interval $[1, 10]$. In keeping with distributions typically found in manufacturing environments, approximately 25% of the tasks have absolute deadlines drawn from a uniform distribution in the interval $[1,\ 3T]$, where T is the number of total tasks. Approximately 25% of the tasks have wait constraints; the duration of non-zero wait constraints is drawn from a uniform distribution in the interval $[1, 10]$. We set the number of locations to be 5, and each task's location is picked randomly. For small and medium problems, we generated 1,000 testing problems. For large problems, we generated 100 testing problems. To train the RoboGNN scheduler, we generated another 1,000 small problems. We ran Gurobi, a commercial optimization solver widely used for mixed integer linear programming (v8.1), with a cutoff time of 15 minutes on those problems to serve as exact baselines for testing set and expert demonstrations for training set. This resulted in a total of 17,657 transitions for training. For large problems, Gurobi cutoff time was 1 hour.

Model Details – Our code implementation uses pyTorch [29], and the graph neural networks are built upon Deep Graph Library (https://www.dgl.ai). We apply a three-layer GAT to learn node features. Each layer uses 8 attention heads computing 64 features. The last GAT layer uses averaging while the first two use concatenation to aggregate the features from each head. The Q network uses two fully-connected layers with a hidden dimension of 64. We set γ = 0.99 and use Adam optimizer [30] through training. Imitation learning uses λ_1 = 0.9, and λ_2 = 0.1. We tested learning rates lr from $\{10^{-2}, 10^{-3}, 10^{-4}\}$, q_o from $\{1,\ 3,\ 5\}$, and found the combination of $lr = 10^{-3}$ and $q_o = 3$ achieved the best performance on test set of small problems. Thus we picked them to report the evaluation results. Both training and evaluation were conducted on a Quadro RTX 8000 GPU.

Benchmarks – We benchmark our trained RoboGNN scheduler against the following methods.

- *Earliest Deadline First (EDF)* – a ubiquitous heuristic algorithm [31] that assigns the available task with the earliest deadline to the first available worker.
- *Tercio* – the state-of-the-art scheduling algorithm for this problem domain [25]. Tercio combines mathematical optimization for task allocation and analytical sequencing test for temporospatial feasibility.
- *Gurobi* – a commercial optimization solver from Gurobi Optimization. Results from Gurobi v8.1 are the exact baseline.

A. Proportion of Problems Solved

The RoboGNN scheduler was trained on small problems and the same model was evaluated on all problem scales. We evaluated our model in terms of proportion of problems solved and compared it with other methods, as shown in Fig. 3. We also reported mean and standard deviation of the computation time for different methods above corresponding

bars, based upon the problems solved by each method.

From this figure, we can see that the RoboGNN scheduler found considerably more feasible solutions than both EDF and Tercio across all team sizes. Our trained policy showed consistently high-performance across different problem sizes (91.5% solved for small problems, 89.3% solved for medium problems, and 91.0% solved for large problems), while the performance of EDF and Tercio decreased precipitously when the number of tasks increased (e.g., proportion of problems solved dropped from 62.0% on small problems to 27.7% on medium problems for Tercio). Moreover, EDF failed to solve any large problems for 2-robot, 3-robot and 4-robot teams, as depicted by zero-height bars in Fig. 3(c). The same is true for Tercio in large 2-robot problems.

As two-robot team imposes a smaller number of robot-related constraints than other team sizes, it took Gurobi longer to find solutions (Fig. 3(a)), and for large-scale problems, this resulted in less feasible solutions within cutoff time (Fig. 4(c)). Overall, for large problems, Gurobi only solved 77.0% problems, and was outperformed by RoboGNN on 2-robot and 3-robot cases. As problem scale increased, the runtime of RoboGNN increased in a faster manner than Tercio, but was still $\sim$ 10x faster than Gurobi, which is a favorable trade-off considering Tercio's poorer performance in the number of problems solved, as shown in Fig. 3.

Considering that we only used expert data on small problems during training, this positive result provides strong evidence that our framework is able to transfer knowledge learned on small problems to help solve larger problems.

B. Normalized Makespan

To compare the quality of solutions found by different methods, we reported results evaluated on another metric: normalized makespan, where the makespan was normalized to the one found by the exact method, Gurobi.

Fig. 4 showed the average makespan score, normalized to the value found by Gurobi, of our approach and other baseline methods. Error bar denoted standard deviation. To make fair comparison, we only counted problems for which all four methods found solutions in Fig. 4(a) and 4(b). In Fig. 4(c), EDF and Tercio were excluded for the problem groups where they found zero feasible solutions. Overall, RoboGNN and Tercio achieved similar makespan score, with EDF being the worst. For large problems, both RoboGNN and Tercio were able to find better solutions than Gurobi.

C. Ablation Study

To show the necessity and benefit of incorporating edge information into the GAT layer, we also trained and evaluated a similar policy based on the original GAT models, using small problems involving 2-robot teams. As a result, the trained policy only solved 5.7% of the testing problems. This showed the effectiveness of our adaptation in order to leverage graph attention networks to automatically learn to coordinate robot teams in complex scheduling environments.

VII. ROBOT DEMONSTRATION

We demonstrate our trained RoboGNN scheduler to coordinate the work of a five-robot team in a simulated environment for airplane fuselage construction, as shown in Fig. 5. The problem consists of eighteen tasks located randomly among 5 locations. Besides respecting the temporal constraints that exist among the tasks, the scheduler has to make sure that the same physical location can only be occupied by at most one robot at any time to prevent collisions. The execution makes use of the Robotarium, a remotely accessible swarm robotics research testbed with GRITSBot X robots [32]. A video with a detailed breakdown of the demonstration can be found at http://tiny.cc/y3vgkz.

VIII. DISCUSSION

Combing the results presented in previous section, we showed that the RoboGNN scheduler not only found significantly more solutions than other heuristics but also achieved high solution quality. Impressively, our network-based scheduler outperformed all baselines in terms of the proportion of instances solved and the solution quality when problem size scales up to 100 tasks for two- and three-robot teams. Our method also outperforms all approximate solution techniques for four- and five-robot teams at this scale while yielding a 100$\times$ speedup over our exact baseline. Even though our method constructs schedules under a deterministic setting, this speedup allows us to re-schedule in a timely manner in response to unexpected disturbance during execution. Furthermore, we can leverage the method from [25], which uses the output schedule's ordering constraints back into the original STN—rather than using the output schedule itself—to preserve a high degree of flexibility in dispatching the robots via the modified STN. We also demonstrated our method on a multi-robot testbed (Fig. 5). We summarize our contributions as follows:

1) To our best knowledge, ours is the first to leverage graph neural networks in solving STN-based scheduling problems with spatial constraints. We extend the graph attention network to deal with directed, weighted graphs by incorporating edge weights during both attention coefficient calculation and node feature aggregation. Our work enables graph neural network to be applied to STNs.
2) We propose a novel graph attention network-based scheduler (RoboGNN) that is non-parametric in both the number of tasks and the number of robots. Benefiting from such scalable structure, the proposed RoboGNN scheduler can be trained via imitation learning on small problems for which expert solution can be easily obtained, and be applied in generating schedules for larger-scale problems.
3) We conduct experiments evaluating the performance of the proposed method, showing the superiority of the trained RoboGNN scheduler—considering solution quality, proportion of instances solve, and computation time—vs. state-of-the-art methods.

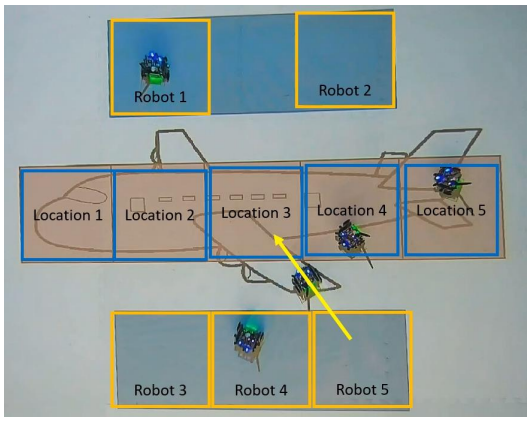


Fig. 5. This figure depicts our demonstration of a 5-robot team completing tasks for airplane fuselage assembly.

IX. CONCLUSIONS

We presented a graph attention network framework to automatically learn a scalable scheduling policy to coordinate multi-robot teams of various sizes. By combining imitation learning with graph attention network in a non-parametric framework, we were able to obtain policy that generated fast, near-optimal scheduling of robot teams. We demonstrated that our network-based policy found significantly more solutions over prior state-of-the-art methods in all testing scenarios. Future research includes extending our work to allow scheduling robots with different capabilities, transfer learning across optimizing different objective functions, and deploying the trained network in a real-world scenario.

REFERENCES

[1] C. Heyer, "Human-robot interaction and future industrial robotics applications," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4749–4754.

[2] H. Raghavan, O. Madani, and R. Jones, "Active learning with feedback on features and instances," *Journal of Machine Learning Research*, vol. 7, no. Aug, pp. 1655–1686, 2006.

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

[4] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018.

[6] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

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

[8] 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.

[9] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European journal of operational research*, vol. 112, no. 1, pp. 3–41, 1999.

[10] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[11] J. Chen and R. G. Askin, "Project selection, scheduling and resource allocation with time dependent returns," *European Journal of Operational Research*, vol. 193, no. 1, pp. 23–34, 2009.

[12] W. Tan and B. Khoshnevis, "A linearized polynomial mixed integer programming model for the integration of process planning and scheduling," *Journal of Intelligent Manufacturing*, vol. 15, no. 5, pp. 593–605, 2004.

[13] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[14] S. M. Mousavi and R. Tavakkoli-Moghaddam, "A hybrid simulated annealing algorithm for location and routing scheduling problems with cross-docking in the supply chain," *Journal of Manufacturing Systems*, vol. 32, no. 2, pp. 335–347, 2013.

[15] L. Zhang and T. Wong, "An object-coding genetic algorithm for integrated process planning and scheduling," *European Journal of Operational Research*, vol. 244, no. 2, pp. 434–444, 2015.

[16] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1114–1120.

[17] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, 2005.

[18] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 430–439, 2011.

[19] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[20] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 554–15 566.

[21] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[22] L. Barbulescu, Z. B. Rubinstein, S. F. Smith, and T. L. Zimmerman, "Distributed coordination of mobile agent teams: the advantage of planning ahead," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 2010, pp. 1331–1338.

[23] B. Coltin and M. Veloso, "Online pickup and delivery planning with transfers for mobile robots," in *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[24] E. Nunes and M. Gini, "Multi-robot auctions for allocation of tasks with temporal constraints," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 2110–2116.

[25] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast scheduling of robot teams performing tasks with temporospatial constraints," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 220–239, 2018.

[26] I. Tsamardinos and M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems," *Artificial Intelligence*, vol. 151, no. 1-2, pp. 43–89, 2003.

[27] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[28] B. Piot, M. Geist, and O. Pietquin, "Boosted bellman residual minimization handling expert demonstrations," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 549–564.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] H. Hellerman, "Some principles of time-sharing scheduler strategies," *IBM Systems Journal*, vol. 8, no. 2, pp. 94–117, 1969.

[32] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, 2020.