Heterogeneous Graph Attention Networks for Scalable Multi-Robot Scheduling with Temporospatial Constraints

Zheyuan Wang and Matthew Gombolay
Georgia Institute of Technology
Atlanta, GA 30332, USA
Email: pjohnwang@gatech.edu, matthew.gombolay@cc.gatech.edu

***Abstract*—Robot teams are increasingly being deployed in environments, such as manufacturing facilities and warehouses, to save cost and improve productivity. To efficiently coordinate multi-robot teams, fast, high-quality scheduling algorithms are essential to satisfy the temporal and spatial constraints imposed by dynamic task specification and part and robot availability. Traditional solutions include exact methods, which are intractable for large-scale problems, or application-specific heuristics, which require expert domain knowledge to develop. In this paper, we propose a novel heterogeneous graph attention network model, called ScheduleNet. By introducing robot- and proximity-specific nodes into the simple temporal network encoding temporal constraints, we obtain a heterogeneous graph structure that is nonparametric in the number of tasks, robots and task resources or locations. We show that our model is end-to-end trainable via imitation learning on small-scale problems, generalizing to large, unseen problems. Empirically, our method outperforms the existing state-of-the-art methods in a variety of testing scenarios.**

I. Introduction

Given the recent developments in robotic technologies and the increasing availability of collaborative robots (cobots), multi-robot systems are increasingly being adopted in various manufacturing and industrial environments [30]. Research in related areas (e.g., multi-robot communication, team formation and control, path planning, task scheduling and routing) has also received significant attention [7]. In this paper, we focus on the problem of multi-robot task allocation and scheduling [15] with both temporal and spatial constraints, which captures the key challenges of final assembly manufacturing with robot teams. To achieve an optimal schedule for a user-specified objective, the robots must be allocated with the proper number of tasks and process these tasks with optimal order, while satisfying temporal constraints such as task deadlines and wait constraints. The addition of spatial constraints (i.e., a location can only be occupied by one robot at a time) makes task allocation difficult because algorithms must reason through inter-coupled, disjunctive time window constraints that govern shared resources.

Traditionally, the scheduling problem is formulated as a mixed-integer linear program (MILP), which can be approached with either exact methods or hand-crafted heuristics. However multi-robot scheduling with both temporal and spatial constraints is generally NP-hard [20]. Exact methods fail to scale to large-scale problems, which is exacerbated by the need for near real-time solutions to prevent factory slow-downs. Moreover, efficient approximation algorithms are hard to design. They not only require domain specific knowledge with respect to each problem configuration that usually takes years to gain, but they also require accurate feature engineering to encode such knowledge, which leaves much to be desired [18].

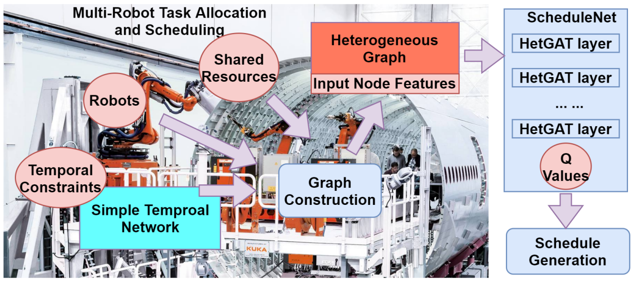


Fig. 1. Overview of the proposed ScheduleNet, which operates on the heterogeneous graph constructed by augmenting the STN of the problem, and predicts Q-values for scheduling. Courtesy: KUKA Robotics.

In recent years, deep neural networks have brought about breakthroughs in many domains, including image classification, nature language understanding and drug discovery, as they can discover intricate structures in high-dimensional data without hand-crafted feature engineering [14]. Promising progress has also been made towards learning heuristics for combinatorial optimization problems by utilizing graph neural networks to learn meaningful representations of the problem to guide the solution construction process [32]. Yet this research focuses on significantly easier problems with a simpler graphical structure, e.g. the traveling salesman problem (TSP).

We propose a novel heterogeneous graph attention network model, called ScheduleNet, to learn heuristics for solving the multi-robot task allocation and scheduling problems with upper- and lowerbound temporal and spatial constraints. Fig. 1 shows the overall framework of our proposed method. We extend the simple temporal network (STN) [4] that encodes the temporal constraints into a heterogeneous graph by adding nodes denoting various components, such as workers (human or robot) and physical locations or other shared resources. By

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

doing so, ScheduleNet directly operates on the heterogeneous graph in a fully-convolutional manner and can estimate the Q-function of state-action pairs to be used for schedule generation. We show that ScheduleNet is end-to-end trainable via imitation learning and generalizes to large, unseen problems with an affordable increase in computation cost. This flexibility allows us to set a new state of the art for multi-robot coordination and in autonomously learning domain-specific heuristics for robotic applications.

II. Related Work

Task assignment and scheduling for multi-robot teams has been studied with various real-world applications, such as manufacturing, warehouse automation and delivery systems [15]. Korsah et al. [13] devised a widely accepted taxonomy, iTax, to categorizethe Multi-Robot Task Allocation (MRTA) problem. Nunes et al. [15] further categorized the extensive research present in the multi-robot task allocation domain and identified possible solutions to this problem.

Task allocation is essentially an optimization problem, and the most common formalism to capture its constraints is Mixed Integer Linear Programming (MILP). As exact methods for solving MILP yield exponential complexity, researchers have combined MILP and constraint programming (CP) methods into a hybrid algorithm using decomposition [1, 8, 19] to accelerate computation. Other hybrid approaches exploited heuristic schedulers to gain better scalability [2, 3].

Learning heuristics for solving scheduling problems has been examined by several research groups. Wu et al. [28], Wang and Usher [26], Zhang and Dietterich [31] applied reinforcement learning methods to learn domain-specific heuristics for job shop scheduling. Wang and Usher [26] developed a Q-learning based method for the single-machine dispatching rule selection problem. Wu et al. [28] proposed a multi-agent reinforcement learning method, called the ordinal sharing learning (OSL) method, for job-scheduling problems such as realizing load balancing inGrids. However, these methods depend on customized, hand-crafted features to achieve satisfying results. In our method, we exploit the power of deep learning models to automatically learn useful features.

Graph neural networks (GNNs), as an extension of convolutional neural networks to a non-Euclidean domain, have been widely applied in graph-based problems such as node classification, link prediction and clustering, and show convincing performance [29]. Compared to the pervasive use of GNNs in classification problems, their application in solving combinatorial optimization is limited. Khalil et al. [10] input the node embeddings learned by a GNN into a Q-learning module and achieved better performance than previous heuristics on solving minimum vertex cover, maximum cut and TSPs. Kool et al. [12] combined GNNs and policy gradient methods to learn a deterministic policy for TSP and two variants of the Vehicle Routing Problem (VRP). Wang and Gombolay [27] developed a GNN-based model operating on the STN to generate schedules for coordinating multi-robot teams. However, their method uses homogeneous graphs and hard-codes robot and location information as node features, making it not scalable when the number of robots changes. Heterogeneous GNNs, which directly operate on heterogeneous graphs containing different types of nodes and links, have shown good interpretability and model expressiveness compared to traditional GNNs [25], but such a model has never been applied to combinatorial optimization problems. We are the first to utilize heterogeneous GNNs for scheduling multi-robot teams.

III. Problem Overview

A. Problem Statement

We consider the problem of coordinating a multi-robot team in the same space, with temporal and resource/location constraints. The problem falls into the single-task robots (ST), single-robot tasks (SR), time-extended assignment (TA) category with cross-schedule dependencies [XD] under the iTax taxonomy proposed by [13]. We describe its components using a six-tuple $<r, \tau, d, w, Loc, z>$. r are the robot agents that we assume are homogeneous in task proficiency. τ are the tasks to be performed. Each task τ_i is associated with a start time s_i and a finish time f_i and takes a certain amount of time dur_i for each robot to complete. 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. d is the deadline constraint that specifies a task must be completed before a certain time point. w is the wait constraint that specifies the relative relationship between two tasks in time axis. Loc is the list 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 can take different forms depending on end-user applications.

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.

Eq. 1-9 give the mathematical program formation of our problem, where two types of binary decision variables are used: 1) $A_{r,i}$ = 1 for the assignment of robot r to task i and 2) $X_{i,j}$ = 1 denoting task i finishes before task j starts. Additionally, L_{same} is the set of task pairs (i, j) that use the same location and is derived from Loc. These equations can be passed to a MILP-based solver to search for optimal solutions.

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

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

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

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

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

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

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

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

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

As z varies depending on application-specific goals, in Section VI we report the results of minimizing the makespan (i.e., overall process duration, $z = \max_i f_i$) as a generic objective function. To show the generalization of our method, we also consider an application-specific case where we try to minimize the weighted sum of the completion time of all tasks ($z = \sum_i c_i f_i$). We use this objective function as an analogy to the minimization of weighted tardiness in job-shop scheduling [5].

B. MDP Formulation

Given the problem statement, we learn greedy heuristics that construct solutions by appending tasks to an individual robot's partial schedule based on maximizing a score Q-function approximated with a neural network parameterized by θ. 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:

- State x_t at a decision-step t includes the temporal constraints of the problem, represented by a STN, the location information, and all robots' partial schedules constructed so far.
- Action $u = <\tau_i, r_j>$ corresponds to appending an unscheduled task τ_i at the end of the partial schedule of robot r_j.
- Transition T corresponds to deterministically adding the edges associated with the action into the STN and updating the partial schedule of the selected robot.
- Reward R of a state-action pair is defined as the change in objective values after taking the action, calculated as $R = -1 \times (Z_{t+1} - Z_t)$. Z_t denotes the partial objective function at state x_t and is calculated only using scheduled tasks. For example, while minimizing makespan, $Z_t = max_i f_i, \tau_i \in \{partial\ schedules\}$. The reward is multiplied by -1.0 as the objective is minimization. We further divide Z_t by a factor $D > 1$ if x_t is not a termination state. We use D to balance between finding the highest immediate reward (local optimal) and finding the global optimal schedules. If the action results in an infeasible schedule in the next state, a large negative reward M_{inf} is assigned to Z_{t+1}.
- Discount factor, γ.

C. Schedule Generation

Our learned heuristic relies on the evaluation function $Q(x, u)$, which will be learned using a collection of problem instances to estimate the total discounted future reward of state-action pairs and select accordingly. We use scheduling-through-simulation to generate schedules as it has been shown in [27] that this process achieves better performance than using decision-step-based generation. In scheduling-through-simulation, 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 $\tau := \text{argmax}_{\tau \in \boldsymbol{\tau_{avail}}} Q_\theta(x, u)$, where $\boldsymbol{\tau_{avail}}$ is the set of unscheduled tasks and only Q values associated with r_j are considered. To stay in accordance with problem requirements, we impose two rules when adding a new task during transition: 1) the start time of all unscheduled tasks should be no earlier than the start time of the newly added task; and 2) the start time of all unscheduled tasks that share the same location should be no earlier than the finish time of the newly added task.

IV. Heterogeneous Graph Attention Network

Traditional graph neural networks (GNNs) operate on homogeneous graphs to learn a universal feature update scheme for all nodes. We instead cast the task scheduling problem into a heterogeneous graph structure, and propose a novel heterogeneous graph attention network, ScheduleNet, that learns per-edge-type message passing and per-node-type feature reduction mechanisms on this graph. One advantage of ScheduleNet is that it directly estimates the Q-value of state-action pairs as its output node feature. In this section, we first describe how to construct the heterogeneous graph given a problem state, x_t. Then we present the building block layer used to assemble a ScheduleNet of arbitrary depth (through stacking this layer), which we call the heterogeneous graph attention layer (HetGAT).

A. Heterogeneous Graph Representation

The temporal constraints in multi-robot task allocation and scheduling problems have been commonly modeled as STNs because the consistency of the upper and lower bound constraints can be efficiently verified in polynomial time [21]. STNs also allow for encoding set-bounded uncertainty. 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 occupying a work location at a time, which is known as the Disjunctive Temporal Problem [22]. To learn a more expressive and scalable representation of the problem, we extend the STN formulation into a heterogeneous graph using the construction process illustrated in Algorithm 1. In a heterogeneous graph, we use a three-tuple, in the form of $<srcName, edgeName, dstName>$, to specify the edge type/relation that connects the two node types (from source node to destination node), which can also be denoted as $(srcName \xrightarrow{edgeName} dstName)$.

In traditional STN formulations, each task, τ_i, is represented by two event nodes: its start time node, s_i, and finish time node, f_i. The directed, weighted edges encode the temporal constraints associating corresponding nodes. Exploiting the fact that task duration is deterministic, we develop a **novel simplification trick** to reduce the model complexity. That is, after running Floyd Warshall's algorithm [6] on the original STN to find its minimum distance graph, we remove all finish time nodes (except f_0) from the distance graph to obtain a new STN. The simplified STN, using only half the nodes, still reserves all the necessary temporal constraints. In this way, each task can be represented by its start time node with task

Algorithm 1: Construct the heterogeneous graph

```
Input: STN, locations $L$, robots $r$ and their partial
       schedules, available actions $\boldsymbol{u_{avail}}$
Output: Heterogeneous graph representation
Run Floyd Warshall's algorihtm on STN to find its
  minimum distance graph, $g_d$;
Remove all $f_i$'s from $g_d$, except $f_0$;
Use $g_d$ as the new STN and $s_i$ as the task node of $\tau_i$;
foreach robot $r_j$ do
    Add a robot node, $r_j$;
    foreach $\tau_m$ assigned to $r_j$ do
        Add an edge $\tau_m \rightarrow r_j$;
    end
end
Connect robot nodes with each other;
foreach location $L_k$ do
    Add a location node, $L_k$;
    foreach $\tau_m$ located in $L_k$ do
        Add an edge $\tau_m \rightarrow L_k$;
    end
end
Connect location nodes with each other;
Add a state node $st$, connect all other nodes to it;
foreach $u_n =< \tau_n, r_n >\in \boldsymbol{u_{avail}}$ do
    Add a value nodes $v_n$;
    Add an edge $\tau_n \rightarrow v_n$;
    Add an edge $r_n \rightarrow v_n$;
    Add an edge $st \rightarrow v_n$;
end
Add self-loops;
return $g_d$.
```

duration now serving as its node feature. Given the partial schedule at the current state, we generate the initial input features of each task node as follows: the first two dimensions are the one-hot encoding of whether a task has been scheduled [1 0] or not [0 1]; the next dimension is the task duration. We denote the edge type from STNs using $(task \xrightarrow{temporal} task)$ as they encode the temporal constraints.

To extend the simplified STN, we add robot and location nodes equaling the number of different robots and locations in the problem, respectively. A robot node is connected to the task nodes that have been assigned to it, with edge relation $(task \xrightarrow{assignedTo} robot)$. All robots are connected with each other to enable message flow between them, with edge relation $(robot \xrightarrow{communicate} robot)$. The initial feature of a robot node is the number of tasks assigned so far. In a similar manner, a location node is connected to the tasks nodes in that location, with edge relation $(task \xrightarrow{locatedIn} location)$. All location nodes are connected with each other, with the relation $(location \xrightarrow{near} location)$. The initial feature of a location node is the number of tasks in that location.

As the Q-function is based on state-action pairs, we also expect the network to learn a state embedding of the problem from all the task, robot, and location node embeddings. To achieve this, we add a state summary node into the graph structure. The state summary node is connected to all the task, robot and location nodes, with edge types $(task \xrightarrow{in} state)$, $(robot \xrightarrow{in} state)$, $(location \xrightarrow{in} state)$, respectively. The initial features of the graph summary node include number of total tasks, number of currently scheduled tasks, number of robots and number of locations.

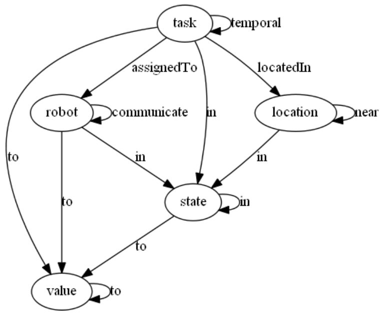


Fig. 2. Meta-graph of the heterogeneous graph built from the STN by adding robot, location, state, and value nodes.

Once the node embeddings are computed using the heterogeneous graph, it is possible to learn a separate Q network consisting of several fully-connected (FC) layers to predict the Q-value of a state-action pair, taking as input the concatenation of embeddings from corresponding state, task, and robot nodes. However, designing a separate Q network on top of GNNs is computationally expensive and not memory efficient, especially when evaluating a large number of state-action pairs at once for parallel computing. Instead, we propose to add value nodes in the graph to directly estimate the Q-values. A value node is connected to corresponding nodes with edge types denoted as $(task \xrightarrow{to} value)$, $(robot \xrightarrow{to} value)$, $(state \xrightarrow{to} value)$. The initial feature of a value node is set to 0. During evaluation, the heterogeneous graph is constructed with the needed Q-value nodes covering task nodes in $\boldsymbol{\tau_{avail}}$ and robot node of r_j, as mentioned in Section III.C. As we are calculating the minimum distance graph of a STN while constructing the heterogeneous graph, we can further filter out the tasks in $\boldsymbol{\tau_{avail}}$ of which the lower bound of task start time is greater than the current time. For all nodes, self-loops are added so that their own features from previous layers are considered for the next layer's computation. Fig. 2 shows the meta-graph containing all the node types and edge types.

B. Heterogeneous Graph Attention Layer

The feature update process in a HetGAT layer is conducted in two steps: per-edge-type message passing followed by per-node-type feature reduction. During message passing, each edge type uses a distinct weight matrix, $W_{edgeName} \in \mathbb{R}^{D\times S}$, to transform the input feature from the source node and then sends the computation result to the destination node, where S is the input feature dimension of the source node, and D is the output feature dimension of the destination node. In the case that several edge types share the same name, we use $W_{srcName,edgeName}$ to distinguish between them. As for edge type $(task \xrightarrow{temporal} task)$ which is the only weighted edge in our heterogeneous graph formulation, the edge attribute, w, is also sent after transformed by $W_{tempEdge} \in \mathbb{R}^{D\times 1}$. During feature reduction, for each edge type that a destination node has, the HetGAT layer computes per-edge-type aggregation result by weighing incoming neighbor features (plus edge attributes if applicable) along the same edge type with feature-dependent and structure-free normalization, in forms of attention coefficients. Those results are then merged to compute the destination node's output feature. The feature update formulas of different node types are listed in Eq. 10-14.

$$\begin{aligned}
\text{Task } h_i' &= \sigma\Big(\sum_{j\in N_{temporal}(i)} \alpha_{ij}^{temporal} (W_{temporal} h_j \\
&\qquad + W_{tempEdge} w_{ji})\Big) &(10)\\
\text{Robot } h_i' &= \sigma\Big(\sum_{j\in N_{assginedTo}(i)} \alpha_{ij}^{assignedTo} W_{assignedTo} h_j \\
&\qquad + \sum_{k\in N_{comm.}(i)} \alpha_{ik}^{comm.} W_{comm.} h_k\Big) &(11)\\
\text{Location } h_i' &= \sigma\Big(\sum_{j\in N_{locateIn}(i)} \alpha_{ij}^{locatedIn} W_{locatedIn} h_j \\
&\qquad + \sum_{k\in N_{near}(i)} \alpha_{ik}^{near} W_{near} h_k\Big) &(12)\\
\text{State } h_i' &= \sigma\Big(\sum_{j\in N_{task,in}(i)} \alpha_{ij}^{task,in} W_{task,in} h_j \\
&\qquad + \sum_{k\in N_{robot,in}(i)} \alpha_{ik}^{robot,in} W_{robot,in} h_k \\
&\qquad + \sum_{m\in N_{loc.,in}(i)} \alpha_{im}^{loc.,in} W_{loc.,in} h_m \\
&\qquad + W_{state,in} h_i\Big) &(13)\\
\text{Value } h_q' &= \sigma\Big(W_{task,to} h_t + W_{robot,to} h_r \\
&\qquad + W_{state,to} h_s + W_{value,to} h_q \Big) &(14)
\end{aligned}$$

In Eq. 10-14, $N_{edgeName}(i)$ is the set of incoming neighbors of node i along a certain edge type, and $\sigma()$ represents the ReLU nonlinearity. The per-edge-type attention coefficient, $\alpha_{ij}^{edgeName}$, is calculated based on source node features and destination node features (plus edge attributes if applicable). More specifically, the attention coefficient for edge type $(task \xrightarrow{temporal} task)$ is calculated by Eq. 15, where $\vec{a}^T_{temporal}$ is the learnable weights, $||$ is the concatenation operation, 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.

$$\alpha_{ij}^{temp.} = \text{softmax}_j \Bigg(\sigma' \Big(\vec{a}^T_{temp.} \Big[W_{temp.}\vec{h}_i || W_{temp.}\vec{h}_j || W_{tempEdge} w_{ji} \Big] \Big) \Bigg) \quad (15)$$

The attention coefficients for other edge types are calculated by Eq. 16. We choose $W_{dstType}$ depending on the destination node type. We use $W_{comm.}$ for robot nodes, W_{near} for location nodes, and $W_{state,in}$ for the state node.

$$\alpha_{ij}^{edgeName} = \text{softmax}_j \Bigg(\sigma' \Big(\vec{a}^T_{edgeName} \Big[W_{edgeName}\vec{h}_i || W_{dstType}\vec{h}_j \Big] \Big) \Bigg) \quad (16)$$

To stabilize the learning process, we utilize the multi-head attention proposed from [23], adapting it to fit the heterogeneous case. We use K independent HetGAT layers to compute nodes features in parallel, and then merge the results as the multi-head output, either by concatenation or by averaging.

V. Imitation Learning

Under the MDP formulation, our goal is to learn a greedy policy for sequential decision making. Thus, it is natural to consider reinforcement learning algorithms (e.g., Q-learning) for training ScheduleNet. However, reinforcement learning relies on finding feasible schedules to learn useful knowledge. In our problems, most permutations of the schedule are infeasible. As a result, reinforcement learning spends much more time than allowed before learning anything of value exploring infeasible solutions.

Instead, we leverage imitation learning methods that learn from high-quality schedules to accelerate the learning process for quick deployment. In real-world scheduling environments, we often have access to high-quality, manually-generated schedules from human experts who currently manage the logistics in manufacturing environments. Moreover, it is practical to optimally solve small-scale problems with exact methods. Given the scalability of the heterogeneous graph, we expect that exploiting such expert data on smaller problems to train the ScheduleNet can generalize well towards solving unseen problems, even in a larger scale.

Let D_{ex} denote the expert dataset that contains all the state-action pairs of schedules either from exact solution methods or the domain experts. For each transition, we 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 the corresponding Q-value from ScheduleNet towards this value as shown in Eq. 17, where the supervised learning loss, L_{ex}, is computed as the mean squared error between $R_t^{(n)}$ and our current estimate of the expert action u_{ex}.

$$L_{ex} = \Big\| Q(x, u_{ex}) - R_t^{(n)} \Big\|^2 \quad (17)$$

To fully exploit the expert data, we ground the Q values of alternative actions u_{alt} (not selected by the expert) to a value below $R_t^{(n)}$ using the loss shown in Eq. 18, where q_o is a positive constant empirically picked as an offset, and N_{alt} is the number of alternate actions at step t. In accordance with the schedule generation scheme, N_{alt} only considers actions involving the same robot selected by the expert.

$$L_{alt} = \frac{\sum \left\| Q(x, u_{alt}) - \min\left(\begin{bmatrix} Q(x, u_{alt}) \\ R_t^{(n)} - q_o \end{bmatrix} \right) \right\|^2}{N_{alt}} \quad (18)$$

The min term in Eq. 18 ensures that the gradient propagates through all the unselected actions that have Q values higher than $R_t^{(n)} - q_o$. The difference from [17] lies in that they only train on the unselected action with the max Q value. The total supervised loss is shown in Eq. 19, 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.

$$L_{total} = L_{ex} + \lambda_1 L_{alt} + \lambda_2 L_2 \quad (19)$$

VI. Experimental Results

We show the results of optimizing a generic objective function, which is the minimization of total makespan. In Section VI-D, we also consider the application-specific objective function mentioned in Section III-A, in which we minimize the weighted sum of task completion time, to investigate how ScheduleNet generalizes under different use cases.

A. Dataset

To evaluate the performance of ScheduleNet, we generate random problems based on [8]. We simulate multi-agent construction of a large workpiece, e.g. an airplane fuselage, with three different configurations: a two-robot team, a five-robot team, and a ten-robot team. Task duration is generated from a uniform distribution in the interval $[1, 10]$. Approximately 25% of the tasks have absolute deadlines drawn from a uniform distribution in the interval [1, $N \times T$], where N is the number of total tasks. We use $T = 5$ for two-robot teams, $T = 2$ for five-robot teams, and $T = 1$ for ten-robot teams. Approximately 25% of the tasks have wait constraints, and the duration of non-zero wait constraints is drawn from a uniform distribution in the interval $[1, 10]$. We set the number of locations in a problem to be the same as the number of robots, and each task's location is picked randomly.

For each team configuration, problems are generated in three scales: small (16-20 tasks), medium (40-50 tasks) and large (80-100). For each problem scale, we generate 1,000 problems for testing. To train the ScheduleNet model, we generate 1,000 small problems of two-robot teams. We run Gurobi with a cutoff time of 15 minutes on generated problems to serve as exact baselines. This resulted in a total of 17,513 transitions for training. To further examine the scalability of ScheduleNet, we also generate 100 ten-robot team problems in extra-large scale (160-200 tasks), and set the Gurobi cutoff time to be 1 hour, as the MILP formulation involves 300,000+ general constraints and 160,000+ binary variables.

B. Benchmark

We benchmark ScheduleNet against the following methods:

- *EDF* – A ubiquitous heuristic algorithm, earliest deadline first (EDF), that selects from a list of available tasks the one with the earliest deadline, assigning it to the first available worker.
- *Tercio* – A state-of-the-art scheduling algorithm for this problem domain, Tercio [9]. Tercio is a hybrid algorithm that combines mathematical optimization for task allocation and an analytical sequencing test to ensure temporal and spatial feasibility. Hyperparameters are chosen from [9]
- *HomGNN* – A neural-network-based method proposed in [27]. Their method uses a homogeneous GNN to exact problem embedding from the STN, and a separate Q-network consisting of two FC layers to predict the Q-value. We denote this model as HomGNN and use the same hyper-parameters in [27].
- *Exact* – Gurobi, a commercial optimization solver widely used for mixed integer linear programming. Its results represent the exact baseline.

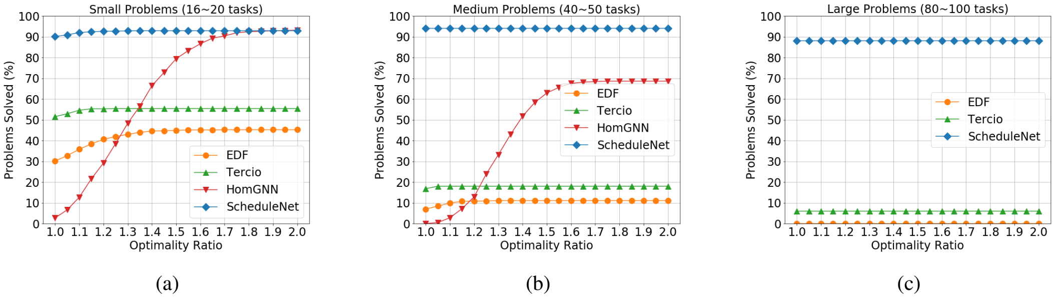
C. Evaluation Results

Metrics – For minimizing the makespan, we use the following metric for evaluation purposes. **M1**: Percentage of problems solved within optimality ratio. A problem is considered solved by an algorithm if the ratio, r, between the objective value it finds and the optimal value is within a certain range (e.g., $r = \frac{z_{algorithm}}{z_{optimal}} \leq 1.1$). Gurobi solutions are used as the optimal value. If the algorithm finds a solution of the problem which Gurobi fails to solve, we set $r = 1$ on this problem during evaluation. By calculating this metric with different optimal ratios, we can obtain a comprehensive view of how the solution quality an algorithm finds is distributed.

Model Details – We implement ScheduleNet using PyTorch [16] and Deep Graph Library [24]. The ScheduleNet used in training/testing is constructed by stacking four multi-head HetGAT layers (the first three use concatenation, and the last one uses averaging). The feature dimension of hidden layers = 64, and the number of heads = 8. We set γ = 0.99, D = 3.0 and used Adam optimizer [11] through training. The training procedure used a learning rate of 10^{-4}, λ_1 = 0.9, λ_2 = 0.1, $q_o = 3.0$ and batch size = 8. Both training and evaluation are conducted on a Quadro RTX 8000 GPU.

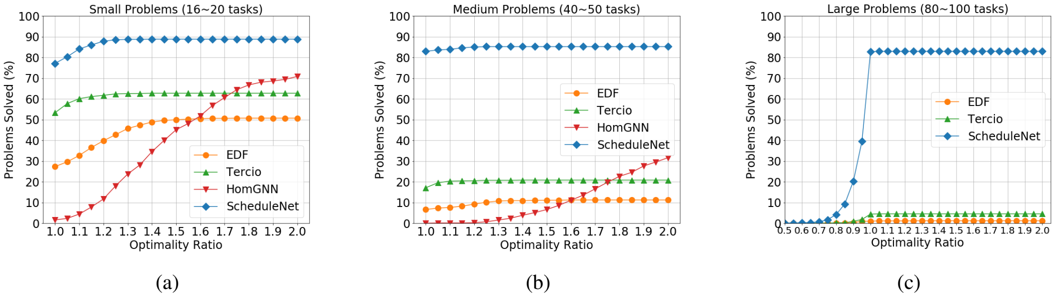
The ScheduleNet was trained on small problems of two-robot teams and the same model was evaluated on all the different problem scales and team configurations. As HomGNN is not scalable in number of robots, for each team configuration, we trained a new model on 1000 small problems and used it for evaluation on the rest. Fig. 3-Fig. 5 compared the evaluation results of different methods using **M1**, where optimality ratio ranges from 1 to 2 with intervals of 0.05 by default.

For small problems, as far as small optimal ratio ($r \leq 1.2$) is concerned, ScheduleNet outperformed three other heuristics (EDF, Tercio, and HomGNN) by a large margin, and achieved significantly closer results to the exact method. This result shows the effectiveness of ScheduleNet in finding high-quality feasible schedules. The only case where HomGNN performed similarly was when examined under a large optimal ratio ($r \geq$



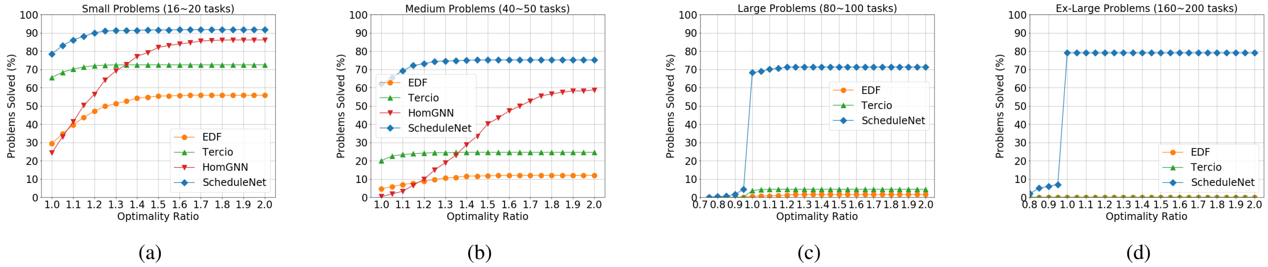
(a) (b) (c)

Fig. 3. Evaluation results on problems of two-robot teams: (a) Small problems; (b) Medium problems; (c) Large problems.



(a) (b) (c)

Fig. 4. Evaluation results on problems of five-robot teams: (a) Small problems; (b) Medium problems; (c) Large problems. For 40% of the large problems, ScheduleNet's solutions outperform Gurobi within cutoff time as denoted by data points left of the 1.0 optimality ratio.



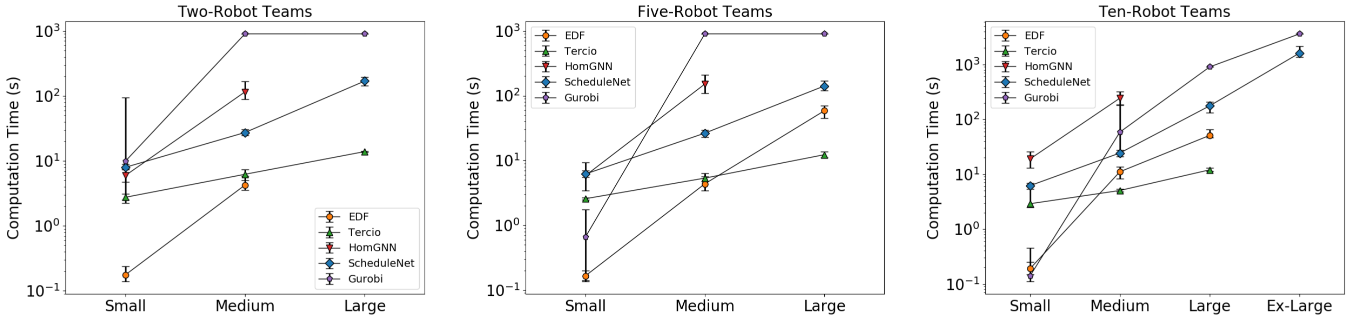
(a) (b) (c) (d)

Fig. 5. Evaluation results on problems of ten-robot teams: (a) Small problems; (b) Medium problems; (c) Large problems; (d) Ex-Large problems. In the Large and Ex-Large problems cases, ScheduleNet is able to find solutions that outperform Gurobi as denoted by data points left of a 1.0 optimality ratio.

1.8), indicating HomGNN was able to find more low-quality solutions, which is often not preferred.

For medium problems, both EDF and Tercio tended to find high-quality schedules, but with a low percentage, while HomGNN found more feasible low-quality solutions. Again, ScheduleNet model significantly outperformed the other three methods. Even though only trained with small problems, the performance of ScheduleNet remained consistent in solving medium and large problems, where a notable performance drop was observed for other methods. HomGNN failed to find solutions on large problems within Gurobi cuttoff time (at least 40 minutes vs. 15 minutes), thus was not reported. During evaluation on large and ex-large problems, we found that for some problems the solutions found by SchedulerNet had better makespans than those found by Gurobi under its cutoff time. Therefore, we extended the opimality ratio to the smallest value under which ScheduleNet still solved at least one problem in Fig. 4(c), Fig. 5(c) and Fig. 5(d). For ex-large problems, Gurobi failed to find most of the feasible solutions within the one hour cuttoff time (8 solved out of 100), while ScheduleNet managed to find substantially more feasible schedules (79 solved). These results demonstrated that our model can transfer knowledge learned on small problem to help solve larger problems, by exploiting the scalabililty within heterogeneous graph formulation.

We reported computation time of different methods in Fig. 6, where only feasible solutions were counted for each method. Due to differences in implementation details, CPU/GPU utilization, besides directly comparing the raw numbers, we also focused on the time changes of each method with respect to increasing problem sizes. When problem size increased, the performance of ScheduleNet stayed consistent with an affordable increase in computation time, which was less than Gurobi. This was largely due to the fully convolutional structure as well as the STN simplification trick that greatly reduced its model complexity and computation cost. As ten-robot team imposes a larger number of robot-related constraints than other team sizes, it took Gurobi less time to find solutions for ten-robot problems than two- and five-robot problems. In contrast, HomGNN failed to scale up to 100 tasks within Gurobi cutoff time. This was mainly due to its structure, where FC layers are stacked on top of a GNN for Q value prediction, making the model complexity proportional to $2 \times N_{task} \times N_{action}$ during parallel evalua-



(a)

(b)

(c)

Fig. 6. Running time statistics on different problems: (a) Two-robot teams; (b) Five-robot teams; (c) Ten-robot teams. Error bars denote the 25th and 75th percentile. Results for EDF, Tercio, and HomGNN are not shown in cases when no solutions are found within the allowed cutoff time.

tion. As a comparison, the structure complexity of ScheduleNet is only proportional to $N_{task} + N_{action}$, considering $N_{robot}, N_{location} \ll N_{task}$.

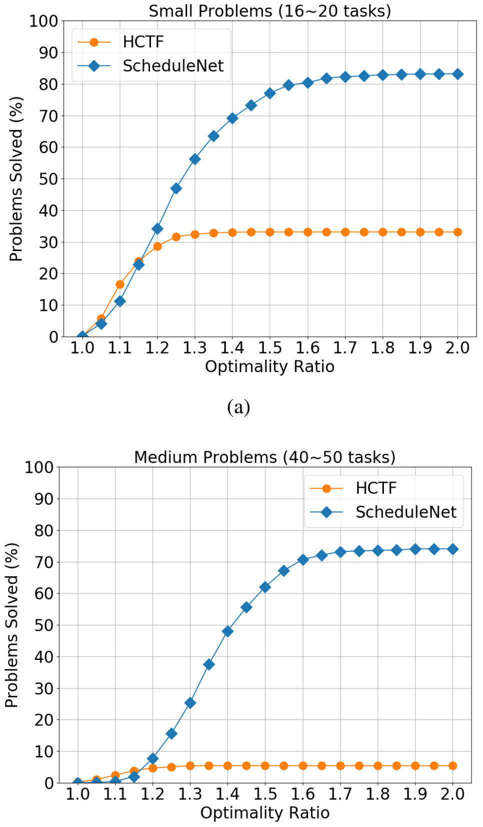
D. Application-Specific Objective Function

To evaluate the performance of our proposed method under a different objective function, $z = \sum_i c_i f_i$, we generated problems involving five-robot teams with two scales: small and medium, following the same parameters as used in Section VI-A. Additionally, each task was associated with a real number cost, c, drawn from a uniform distribution in the interval [1, 10]. For each problem scale, 1,000 problems were generated for testing. We generated 1000 small problems for training the ScheduleNet. We ran Gurobi on all problems with a cutoff time of 15 minutes to serve as exact baselines. We used the same set of parameters during training as used in the total makespan case, except $q_o = 30$, considering the reward was generally larger. We compare ScheduleNet against a Highest Cost Tardiness First (HCTF) priority heuristic which assigns the task with the highest cost to the first available worker in every scheduling decision. Fig. 7 shows the evaluation results. For $r \leq 1.2$ both methods solved similar number of problems. However, under larger optimaliy ratios, ScheduleNet started to outperform HCTF, resulting in a better overall performance.

E. ScheduleNet Takeaways

Our empirical analysis demonstrates that ScheduleNet establishes a state-of-the-art in autonomously learning heuristics for coordinating teams of robots in a computationally efficient framework. In particular, we are to:

1) Outperform prior work in multi-robot scheduling both in terms of schedule optimality and the total number of feasible schedules found (Fig. 3-5).
2) Achieve this superior performance in a flexible framework that allows us to train via imitation-based Q-learning on smaller problems to provide high-quality schedules on larger problems.
3) Autonomously learn scalable scheduling heuristics on multiple application domains (Fig. 3 vs. Fig. 7), attaining an order of magnitude speedup vs. an exact method (Fig. 6).



(a)

(b)

Fig. 7. Evaluation results of minimizing the weighted sum of completion times on five-robot teams: (a) Small problems; (b) Medium problems.

VII. Conclusion

We presented a novel heterogeneous graph attention network model, called ScheduleNet, to learn scalable policy for multi-robot task allocation and scheduling problems. By introducing robot- and proximity-specific nodes into the simple temporal network that encodes the temporal constraints, we obtained a heterogeneous graph structure that is nonparametric in the number of tasks, robots and task resources. We showed that the model is end-to-end trainable via imitation learning with expert demonstrations. Empirically, we showed that our method outperformed existing state-of-the-art methods in a variety of testing scenarios.

References

[1] Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[2] Elkin Castro and Sanja Petrovic. Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *Journal of Scheduling*, 15 (3):333–346, 2012.

[3] Jiaqiong Chen and Ronald G Askin. Project selection, scheduling and resource allocation with time dependent returns. *European Journal of Operational Research*, 193 (1):23–34, 2009.

[4] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

[5] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35(8):2599–2616, 2008.

[6] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[7] Eduardo Feo Flushing, Luca M Gambardella, and Gianni A Di Caro. Simultaneous task allocation, data routing, and transmission scheduling in mobile multi-robot teams. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1861–1868. IEEE, 2017.

[8] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Robotics: Science and System*, pages 49–56, 2013.

[9] Matthew C Gombolay, Ronald J Wilcox, and Julie A Shah. Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics*, 34(1):220–239, 2018.

[10] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2019.

[13] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[15] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, 2017.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[17] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted bellman residual minimization handling expert demonstrations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 549–564. Springer, 2014.

[18] Hema Raghavan, Omid Madani, and Rosie Jones. Active learning with feedback on features and instances. *Journal of Machine Learning Research*, 7(Aug):1655–1686, 2006.

[19] Huizhi Ren and Lixin Tang. An improved hybrid milp/cp algorithm framework for the job-shop scheduling. In *2009 IEEE International Conference on Automation and Logistics*, pages 890–894. IEEE, 2009.

[20] Marius M Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16(2): 161–174, 1986.

[21] Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. *Master's thesis, University of Pittsburgh*, 2000.

[22] Ioannis Tsamardinos and Martha E Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1-2):43–89, 2003.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[24] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[25] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032. ACM, 2019.

[26] Yi-Chi Wang and John M Usher. Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18(1):73–82, 2005.

[27] Zheyuan Wang and Matthew Gombolay. Learning to Dynamically Coordinate Multi-Robot Teams in Graph Attention Networks. *arXiv preprint arXiv:1912.02059*,

2019.

[28] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27(5):430–439, 2011.

[29] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

[30] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12): 399, 2013.

[31] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[32] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.