LEARNING DYNAMIC PRIORITY SCHEDULING POLICIES WITH GRAPH ATTENTION NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Zheyuan Wang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2022

LEARNING DYNAMIC PRIORITY SCHEDULING POLICIES WITH GRAPH ATTENTION NETWORKS

Thesis committee:

Dr. Matthew Gombolay, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Sonia Chernova
School of Interactive Computing
Georgia Institute of Technology

Dr. Matthieu Bloch, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
Department of Electrical Engineering and Computer Science
University of California, Irvine

Dr. Harish Ravichandar
School of Interactive Computing
Georgia Institute of Technology

Dr. Elias Khalil
Department of Mechanical and Industrial Engineering
University of Toronto

Date approved: December 7, 2022

ACKNOWLEDGMENTS

First, I would like to express my sincere gratitude and appreciation to my wonderful advisor, Dr. Matthew Gombolay. My research would not have been possible without his valuable advice, insightful guidance, and strong support. I really have learned a lot from him about not only research but also enthusiasm, dedication, and professionalism in his work. Besides, I would like to greatly thank my co-advisor, Dr. Matthieu Bloch, for his never-ending help, patience, understanding, and encouragement through hard times. I feel extremely lucky to have worked with them during my Ph.D. studies. Their advisorship had a significant and positive impact on my future career and even life.

I would like to extend my appreciation to the committee members, Dr. Sonica Chernova, Dr. Magnus Egerstedt, Dr. Harish Ravichandar and Dr. Elias Khalil, for their time and consideration in reviewing my Ph.D. thesis, and for being generous in sharing their expertise and advice. Their valuable comments and feedback have been really helpful to improve my research.

I am grateful to be a member of the Klaus 1306 squad with Rohan Paleja, Esmaeil Seraj, and Letian Chen, for the stimulating discussions, for the late nights we worked together, and for all the fun we have had at Georgia Tech. Indeed, they were sincere friends and great collaborators who fully motivated me. My sincere thanks also go to my colleagues in Georgia Teach and the CORE robotics lab members, especially Batuhan Altundas, Joshua Bishop, Manisha Natarajan, and Dr. Nakul Gopalan. Sharing thoughts with them is always fun and refreshing, which offers many inspirations for my own research.

Last but not least, I would like to especially thank my girlfriend, Xiaoyu Liu, and my parents. Their endless love and support made me complete this long trip. I am grateful to them for pulling me through difficult times with their unconditional love. I would never have completed my Ph.D. thesis without their support, encouragement, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Resource optimization plays an important role in many real-world scenarios, including health care, manufacturing and services industries, and more. In those resource-constrained environments, effective sequencing and scheduling of workers and jobs has become a necessity for success. Activities must be scheduled to meet various temporal constraints while using the resources available in an efficient manner. Traditional methods for solving scheduling problems are based on dynamic programming and integer programming formulations of the problems, which can be approached with either exact methods that are computationally expensive and hard to scale, or hand-crafted heuristics that can give high-quality solutions but require a combined, herculean effort from computer scientists, operations researchers, and industrial engineers to develop.

The aim of this thesis is to develop novel graph attention network-based models to automatically learn scheduling policies for effectively solving resource optimization problems, covering both deterministic and stochastic environments. The policy learning methods utilize both imitation learning, when expert demonstrations are accessible at low cost, and reinforcement learning, when otherwise reward engineering is feasible. By parameterizing the learner with graph attention networks, the framework is computationally efficient and results in scalable resource optimization schedulers that adapt to various problem structures.

This thesis addresses the problem of multi-robot task allocation (MRTA) under temporospatial constraints. Initially, robots with deterministic and homogeneous task performance are considered with the development of the RoboGNN scheduler. Then, I develop ScheduleNet, a novel heterogeneous graph attention network model, to efficiently reason about coordinating teams of heterogeneous robots. Next, I address problems under the more challenging stochastic setting in two parts. Part 1) Scheduling with stochastic and dynamic task completion times. The MRTA problem is extended by introducing human co-

workers with dynamic learning curves and stochastic task execution. HybridNet, a hybrid network structure, has been developed that utilizes a heterogeneous graph-based encoder and a recurrent schedule propagator, to carry out fast schedule generation in multi-round settings. Part 2) Scheduling with stochastic and dynamic task arrival and completion times. With an application in failure-predictive plane maintenance, I develop a heterogeneous graph-based policy optimization (HetGPO) approach to enable learning robust scheduling policies in highly stochastic environments.

My research fills the current gap between representation learning and policy learning for solving resource optimization problems by building a unified framework. I further advances the idea of learning to schedule by refining and applying it in more complex and challenging scenarios. Through extensive experiments, the proposed framework has been shown to outperform prior state-of-the-art algorithms in different applications. My research contributes several key innovations regarding designing graph-based learning algorithms in operations research.

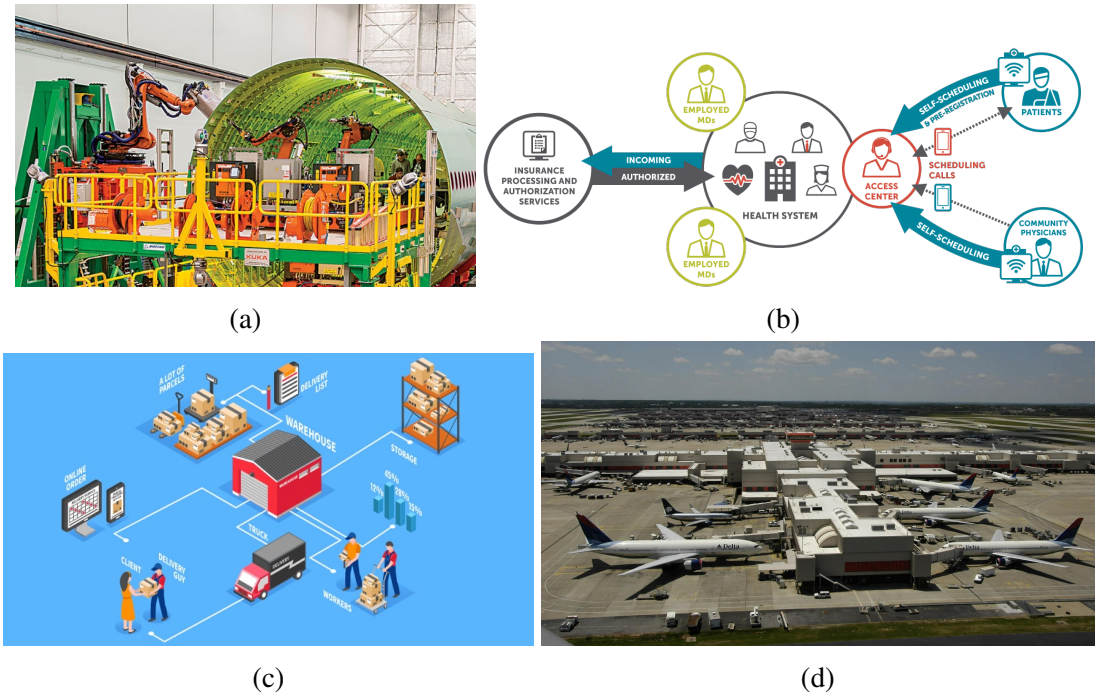
CHAPTER 1
INTRODUCTION

Resource optimization plays an important role in many real-world scenarios, including health care, manufacturing and services industries, and more [1], as shown in Figure 1.1. In those resource-constrained environments, effective sequencing and scheduling of workers and jobs has become a necessity for success. Activities must be scheduled to meet various temporal constraints while using the resources available in an efficient manner.

One popular application of resource optimization techniques lies in task planning of multi-robot systems. Given the recent developments in robotic technologies and the increasing availability of collaborative robots (cobots), multi-robot systems have been adopted in various manufacturing and industrial environments [2]. Research in related areas (e.g., multi-robot communication [3], team formation and control [4], path planning [5, 6], task scheduling and routing [7]) has also received significant attention [8]. Here, we focus on the problem of multi-agent task allocation and scheduling [9] 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 appropriate 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 specific work area can only be occupied by one robot at a time and robots must maintain a minimum distance from other agents while performing a task) makes scheduling even more difficult because one must reason through inter-coupled, disjunctive sequencing constraints that impact shared resource utilization.

One of the rising trends in multi-robot systems is the inclusion of human workers, which typically have latent, dynamic, and task-specific proficiencies, alongside robots [10]. Ef-



(a) (b)

(c) (d)

Figure 1.1: Diverse application domains of resource optimization problems that require coordinating a finite number of resources to accomplish a set of tasks as efficiently as possible: (a) Industrial manufacturing; (b) Patient appointment scheduling in healthcare; (c) Logistics in e-commerce; (d) Airline and crew scheduling. (Images from web. Courtesy: KUKA Robotics, Peerbits, Hartsfield - Jackson Atlanta International Airport)

fective collaboration in human-robot teams must consider the ability of humans to learn and improve in task performance over time [11]. However, it is non-trivial to infer human strengths and weaknesses while ensuring that the team satisfies requisite scheduling constraints, due to the variability in task execution behavior across different individuals, as well as their future task performance affected by human's learning effects with practice [12]. Moreover, a lack of consideration for human preferences and perceived equality may, in the long run, put efficient behavior and fluent coordination at a contradiction [13]. Therefore, enabling a robot to infer human strengths and weaknesses while ensuring that the team satisfies requisite scheduling constraints is a challenging problem worth investigating.

Complicating the allocation and sequencing of workers and tasks in real-world environments are the numerous sources of uncertainty or stochasticity, such as machine breakdowns, unexpected releases of high priority jobs, uncertainty in processing times, etc [14].

Besides the scheduling problem with stochastic human workers, another such example is aircraft maintenance scheduling in which the inter-arrival times of part failures and the resulting service times are latent random variables [15]. This stochasticity makes the scheduling problem more difficult, as the scheduler needs to reason about whether to preemptively service each aircraft. Optimizing aircraft maintenance has drawn keen interest, due to the significant contribution of maintenance costs to overall operating expenses and aircraft availability [16]. One of the most promising strategies of reducing cost is by scheduling predictive maintenance, which entails deciding whether and when to preemptively service one or more of an aircraft's subsystems before the subsystem fails [17]. Research suggests that predictive maintenance could reduce unscheduled work up to 33% [18], which would result in an annual savings of $21.7 billion globally[1].

Traditional methods for solving scheduling in resource optimization are based on dynamic programming and integer programming formulations of the problems [21], which can be approached with either exact methods or hand-crafted heuristics [22, 23]. Exact methods are computationally expensive and usually fail to scale to large-scale problems, which is exacerbated by the need for near real-time solutions to prevent factory slowdowns. On the other hand, application-specific heuristics can give high-quality solutions quickly. However, developing such heuristics often involves a combined, herculean effort from computer scientists, operations researchers, and industrial engineers that leaves much to be desired [24]. For example, predictive aircraft maintenance scheduling is usually performed with ad hoc, hand-crafted heuristics and manual scheduling by human domain experts, which is a time-consuming and laborious process that is hard to scale. Because of these issues, researchers are becoming increasingly interested in developing automatic scheduling solutions that can not only provide high-quality schedules on large scale but also generalize to different application needs.

In recent years, deep neural networks (DNNs) have brought about breakthroughs in

[1] Based upon 2012 figures for worldwide airline revenue of $598 Billion [19] and 11% of revenue allocated for maintenance [20].

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 [25]. The advancements have fostered the idea of leveraging DNNs to solve a plethora of problems in operations research [26]. Particularly, promising progress has been made in learning scalable solvers with graph neural networks (GNNs) via imitation learning (IL) or reinforcement learning (RL), outperforming state-of-the-art, approximate methods [27, 28, 29]. Yet this research focuses on significantly easier problems with a simpler graphical structure, e.g. the traveling salesman problem (TSP). Moreover, the proposed approaches require static, deterministic setting which limits applicability for stochastic resource optimization.

Bridging the gap between deep learning and resource optimization, in this dissertation we build a unified framework of learning scalable scheduling policies for effectively solving resource optimization problems, as shown in Figure 1.2. We combine representation learning and policy learning, while tailoring them for resource optimization problems. For learning on the problem representation, we utilize the graph formulation of problem constraints/components to develop graph neural network-based models. By parameterizing the learner with graph attention networks, our framework is computationally efficient and results in scalable resource optimization schedulers that adapt to various problem structures. For learning on the sequential decision-making process, we consider both deterministic and stochastic environments. We utilize both imitation learning, when expert demonstrations are accessible in low cost, and reinforcement learning, when reward engineering are feasible, to train our scheduling networks.

This chapter serves as a summary of the innovations and findings in this thesis. The following sections mirror the structure of the thesis. We first apply our unified framework to address the problem of multi-robot task allocation and scheduling. In Section 1.1, we consider teams of robots with deterministic and homogeneous task performance (i.e., each robot was equally proficient in completing a given task and the task duration is known be-

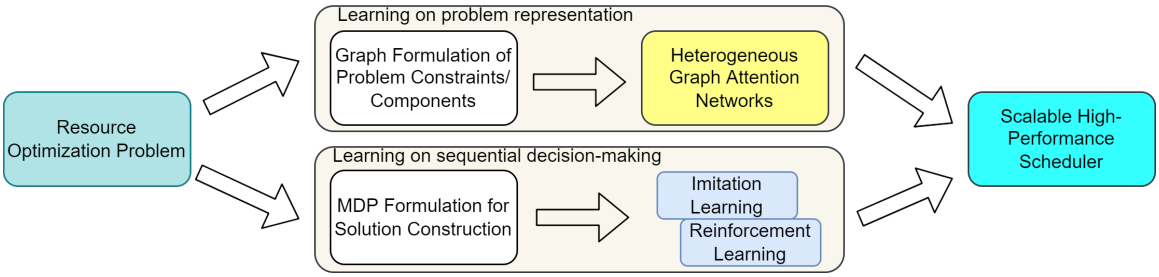


Figure 1.2: The figure depicts the aim of my thesis: building a unified framework of learning scalable scheduling policies for effectively solving resource optimization problems.

forehand). In this scenario, we demonstrate that we are able to train our graph attention network-based model, called RoboGNN, to learn scalable scheduling policies that outperform the existing state-of-the-art methods in a variety of testing cases. Then, in Section 1.2, we build upon this work and propose ScheduleNet, a novel heterogeneous graph attention network model, to efficiently reason about coordinating teams of heterogeneous robots (i.e., robots have varying proficiencies in completing each task).

Next, we adapt this promising framework to tackle problems under the more challenging stochastic setting in two parts: Part 1) Scheduling with stochastic and dynamic task completion times; Part 2) Scheduling with stochastic and dynamic task arrival and completion times, with an application in failure-predictive plane maintenance. To address Part 1, in Section 1.3, we extend the multi-robot task scheduling problem by introducing human co-workers with dynamic learning curves and stochastic task execution. We aim to learn a stochastic scheduling policy. We propose HybridNet, a hybrid network structure that utilizes a heterogeneous graph-based encoder and a recurrent schedule propagator, to carry out fast schedule generation for stochastic human-robot teams in multi-round settings. For Part 2, in Section 1.4, we develop heterogeneous graph based policy optimization (HetGPO) approach to learn scalable scheduling policies for assigning maintenance crews to aircraft. Several variance reduction techniques are developed to enable robust learning in highly stochastic environments. Moreover, we build AirME, a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, as an open-source test bed.

My research further advances the idea of learning to schedule by refining and applying it in more complex and challenging scenarios, where a unified learning framework is built. Through extensive experiments, the proposed framework has been shown to outperform prior state-of-the-art algorithms in different applications. My research contributes several key innovations regarding designing graph-based learning algorithms in operations research.

The remainder of dissertation is organized as follows. Chapter 2 gives the background and summarizes prior work in related areas. Chapter 3 descripes the RoboGNN scheduler for homogeneneous robot teams. Chapter 4 presents the ScheduleNet model for heterogenous robot teams. Chapter 5 covers HybridNet for scheduling mixed human-robot teams. In Chapter 6, we talk about the application of HetGPO in aircraft maintenance scheduling. Chapter 7 concludes the whole research and discusses the future work.

1.1 Scheduling Robots with Graph Attention Networks

Given the recent developments in robotic technologies and the increasing availability of collaborative robots (cobots), multi-robot systems are increasingly being adopted in various environments, including manufacturing, warehouses, and hospitals [2]. Research in related areas (e.g., multi-robot communication, team formation and control, path planning, task scheduling and routing) has also received significant attention. Our research focuses on multi-robot task allocation and scheduling. As an example, consider coordinating a team of robots to construct automotive parts, where different tasks are required at different workstations and have various time requirements to be satisfied (e.g., a certain amount of waiting time is needed between painting tasks to let the previous coat of paint fully dry). The goal is to try to find an optimal schedule, detailing which robot each task is assigned to and in which order the tasks are processed by each robot, while maximizing or minimizing a user-specified objective (e.g., total time used, total resource consumption).

Conventional approaches to multi-robot scheduling involve formulating the problem as

a mathematical program and leveraging commercial solvers or developing custom-made approximate and meta-heuristic techniques. However, multi-robot scheduling with both temporal and spatial constraints is generally NP-hard [22]. This means that exact methods always fail to scale to large-scale problems, which is exacerbated by the need for near real-time solutions to prevent factory slowdowns. Alternatively, heuristic approaches are lightweight and effective. Yet, designing good heuristics involves a combined, herculean effort from computer scientists, operations researchers, and industrial engineers that leaves much to be desired [30]. Moreover, the performance of heuristics is usually bound to specific objective functions. Even with the same kinds of problems, when the optimization objective changes, new efforts are needed to re-design the heuristics to perform well again.

In recent years, deep neural networks have brought about breakthroughs in many domains, including image classification, natural language understanding, and drug discovery, as neural nets can discover intricate structures in high-dimensional data without hand-crafted feature engineering [25]. Can deep learning save us from the tedious work of designing application-specific scheduling solutions? It would be desirable to let the computer autonomously learn scheduling policies without the need for domain experts. Promising progress has been made towards learning heuristics for combinatorial optimization problems. Yet previous research focuses on significantly easier problems than the multi-robot scheduling problem [31]. To push this idea further, we try to develop a novel neural network-based model that learns to reason through the complex constraints of multi-robot scheduling for the purpose of constructing high-quality solutions.

Approach

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. In this chapter, we develop a novel model, called the RoboGNN scheduler, which is based on the graph attention network (GAT) [32], to learn scheduling policies

that reason about the underlying simple temporal network (STN) structure [33] 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 and 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 warm-starts, and can still be applied to 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.

Results and Contributions

The proposed RoboGNN is the first scheduler 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, enabling GNNs to learn from STN structures. We demonstrate that our approach is able to find high-quality solutions for $\sim$90% of the testing problems involving two to five robots and up to 100 tasks with proximity constraints, which significantly outperforms the 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.

1.2 Heterogeneous Graph Attention Networks for Scalable Multi-Robot Scheduling

In this chapter, we extend our work to allow scheduling robots with different capabilities. We present a novel heterogeneous graph attention network model, called ScheduleNet, to learn a scalable policy for multi-robot task allocation and scheduling problems.

Approach

ScheduleNet extends the simple temporal network (STN) 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 doing so, ScheduleNet is nonparametric in the number of tasks, robots, and task resources and directly estimates the Q-function of state-action pairs to be used for schedule generation. The development of ScheduleNet extends from the RoboGNN scheduler which was limited to modeling only teams of robots with homogeneous task performance (i.e., each robot was equally proficient in completing a given task) and could only consider a more restricted set of shared resource constraints. We build upon this prior work in three key ways. First, we extend ScheduleNet to efficiently reason about coordinating teams of heterogeneous robots (i.e., robots have varying proficiencies in completing each task). Second, we expand the types of spatial constraints from 1-dimensional (1D) locations to 2D areas with minimum-distance constraints. Third, to improve ScheduleNet's ability to coordinate heterogeneous teams with such spatial constraints, we further augment our approach by proposing novel schedule synthesis strategies. These extensions and the accompanying empirical validation and robot demonstration serve to provide a more holistic view of ScheduleNet's capabilities with emphases on its flexibility, scalability, and generalizability.

Results and Contributions

ScheduleNet is the first model to utilize heterogeneous GNNs for scheduling multi-robot teams. We show that ScheduleNet is end-to-end trainable via imitation learning on small-scale problems 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. Our results show that ScheduleNet outperforms benchmark approaches when considering both homogeneous and heterogeneous cases. Our extension even solves random problem instances with up to 10 heterogeneous robots and 200 tasks when no other baseline can solve even a single such instance. Given the high expressiveness of heterogeneous graphs, the research opens up future opportunities in designing graph-based learning algorithms in multi-robot research.

1.3 Recurrent Schedule Propagation for Coordinating Human-Robot Teams

In this chapter, we focus on the problem of multi-agent task allocation and scheduling [9] with mixed human-robot teams over multiple iterations of the same coordination problem. Our work accounts for and leverages stochastic, time-varying human task performance to quickly solve task allocation problems among team members 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) and spatial constraints (i.e., safety distance constraints).

Compared to task scheduling within multi-robot systems, the inclusion of human workers makes scheduling even more challenging because, while robots can be programmed to carry out certain tasks at a fixed rate, human workers typically have latent, dynamic, and task-specific proficiencies. Effective collaboration in human-robot teams requires utilizing the distinct abilities of each team member to achieve safe, effective, and fluent execution.

For these problems, we must consider the ability of humans to learn and improve in task performance over time. To exploit this property, a scheduling algorithm must reason about a human's latent performance characteristics in order to decide whether to assign the best worker to a task now versus giving more task experience to a person who is slower but has a greater potential for fluency at that particular task. However, it is non-trivial to infer human strengths and weaknesses while ensuring that the team satisfies requisite scheduling constraints, due to the uncertainty introduced by variability in task execution behavior across different individuals, as well as uncertainty on future task performance affected by human's learning effects with practice [12].

Recent advances in scheduling methods for human-robot teams have shown a significant improvement in the ability to dynamically coordinate large-scale teams in final assembly manufacturing [34, 22]. Prior approaches typically rely on an assumption of deterministic or static worker-task proficiencies to formulate the scheduling problem as a mixed-integer linear program (MILP), which is generally NP-hard [35].

In previous chapters, we showed that graph neural networks can be combined with imitation learning to efficiently solve multi-agent task allocation and scheduling. However, both RoboGNN and ScheduleNet require deterministic environments with known agent performance, making them not suitable for stochastic human-robot teams. For deep learning-based human-robot scheduling solution, better graph neural network structure and improved policy learning algorithm must be developed.

Approach

In this chapter, we propose a deep learning-based framework, called HybridNet, for scheduling stochastic human-robot teams under temporospatial constraints. HybridNet utilizes a heterogeneous graph-based encoder and a recurrent schedule propagator. The encoder extracts high-level embeddings of the initial environment using a heterogeneous graph representation extended from the STN. By formulating task scheduling as a sequential decision-

making process, the recurrent propagator uses Long Short Term Memory (LSTM) cells to generate the consequential models of each task-agent assignment based on the initial embeddings.

We present a novel policy learning framework that jointly learns how to pick agents and tasks and only needs a single reward at the end of the schedule. By factoring in the action space into an agent selector and a task selector, we enable conditional policy learning with HybridNet. We account for the state and agent models when selecting the agents, and combine the information regarding the tasks, the selected agent and the state for task assignment.

Results and Contributions

HybridNet is the first deep learning-based framework for stochastic human-robot coordination under temporospatial constraints. The novel structure HybridNet uses allows for fast schedule generation while removing the need to interact with the environment between every task-agent pair selection. By factoring in the action space into an agent selector and a task selector, HybridNet is end-to-end trainable via Policy Gradients algorithms that jointly learn how to pick agents and tasks.

We develop a virtual Multi-Round Scheduling Environment (MuRSE) for mixed human-robot teams, capable of modeling the stochastic learning behaviors of human workers. MuRSE is OpenAI gym-compatible and open source, and we expect it to serve as a testbed to facilitate the development of human-robot scheduling algorithms. Using MuRSE, we conducted extensive experiments to benchmark the performance of HybridNet across various problem configurations. Results showed HybridNet consistently outperformed prior human-robot scheduling solutions under both deterministic and stochastic settings.

1.4 Failure-Predictive Maintenance Scheduling using Heterogeneous Graph-Based Policy Optimization

Optimizing aircraft maintenance has drawn keen interest due to the significant contribution of maintenance costs to overall operating expenses and aircraft availability [16]. One of the most promising strategies for reducing cost is by scheduling predictive maintenance, which entails deciding whether and when to preemptively service one or more of an aircraft's subsystems before the subsystem fails [17]. Currently, predictive maintenance scheduling is performed with ad hoc, hand-crafted heuristics and manual scheduling by human domain experts, which is a time-consuming and laborious process that is hard to scale. Because of these issues, researchers are becoming increasingly interested in developing automatic scheduling solutions that can not only provide high-quality schedules on a large scale but also generalize to different application needs.

Approach

In this chapter, we propose an innovative design of the scheduling policy network operating on a heterogeneous graph representation of the predictive-maintenance scheduling environment. Two keys to our approach are: 1) we directly model the dynamic scheduling decisions as nodes within a heterogeneous graph network, allowing for an end-to-end trainable resource scheduling policy that is capable of reasoning over the various interactions within the environment, computationally lightweight, and nonparametric to problem scales; 2) we develop an RL-based policy optimization procedure to enable robust learning in highly stochastic environments for which typical actor-critic RL methods are ill-suited.

We build and open source our simulation environments to facilitate the R&D cycle of stochastic scheduling algorithms. We worked in consultation with aerospace industry partners to develop a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, which we call AirME. The challenges for scheduling in AirME come from the

stochasticity in maintenance tasks and the uncertainty of potential component failures that greatly influence maintenance costs.

Results and Contributions

The proposed heterogeneous graph based policy optimization (HetGPO) approach utilizes a heterogeneous graph neural network-based policy and several variance reduction methods towards robust learning in highly stochastic scheduling scenarios. Using AirME, we empirically validate HetGPO across a set of problem sizes and when optimizing for multiple objective functions. Results across various problem scales and objective functions show the effectiveness of HetGPO over conventional, hand-crafted heuristics and baseline learning methods.

Moreover, HetGPO is designed with the mindset of a general, graph-based policy learning algorithm to solve a broader class of stochastic resource optimization problems that are not restricted to aircraft maintenance scheduling. Both the heterogeneous graph formulation techniques (e.g., the use of "state summary" and "decision value" nodes) and the HetGPO training process can be used in similar stochastic scheduling domains as they require little hand-engineering.

CHAPTER 2
RELATED WORK

Our research draws upon work in multi-agent task scheduling, uncertainty in stochastic scheduling, policy learning for combinatorial optimization, and graph neural networks.

2.1 Multi-Agent Task Scheduling

Task scheduling—allocating agents to tasks and sequencing those tasks—has been one of the key problems in multi-agent systems [36]. To maximize or minimize a given objective depending on application needs, the agents must be allocated with the proper number of tasks and process these tasks with optimal order, while satisfying various types of constraints. However, the problem of optimally scheduling $n \geq 3$ tasks (each with a sequence of n_i operationss) on a set of $m \geq 3$ machines is NP-hard [37]. Solving multi-agent task scheduling problems in an optimal way is a great challenge, especially when heterogeneous agents, complex tasks, and dynamic environments should be considered. Proposed approaches in literature for task scheduling are categorized into centralized and distributed approaches [38]. In centralized approaches, there is only one decision-making unit which is assumed to have full information of the system. This central unit computes the optimal or near-optimal decisions. In distributed methods, each agent need to make its own decision. Several negotiation frameworks are developed for distributed agents to cooperate with each other to maximize the efficiency of the system [39, 40].

2.1.1 Multi-Robot Task Allocation and Scheduling

Task assignment and scheduling for multi-robot teams has been studied with various real-world applications, such as manufacturing, warehouse automation and delivery systems [9]. Gerkey and Mataric [41] devised a widely accepted taxonomy to categorize Multi-Robot

Task Allocation (MRTA) problem according to three criteria. First, they classified robots according to their ability to perform single- or multi-tasks at a time. Second, they distinguished between tasks that require single robots to be performed, and tasks that require the coordinated effort of a team of robots. Third, considering the time needed to complete a task, they distinguished between instantaneous tasks and time-extended tasks. Korsah et al. [42] improved Gerkey and Mataric's taxonomy into iTax, by adding a new dimension defining the degree of interdependence of agent–task utilities, with four possible values: No Dependencies (ND), In-schedule Dependencies (ID), Cross-schedule Dependencies (XD), and Complex Dependencies (CD). According to iTax, our research fits within the XD category, with single-task robots [ST], single-robot tasks [SR], and the time-extended allocation [TA] problem (XD [ST-SR-TA]), while additionally taking into consideration human agents with stochastic behavior. Cross-schedule dependencies exist when the utility of one agent is directly affected by the scheduling commitment of another. Nunes et al. [9] further categorized the extensive research present in this domain, with a focus on temporal and ordering constraints, and summarized widely used models and methods.

MRTA problem is essentially an optimization problem, and the most common formalism to capture its constraints is Mixed Integer Linear Programming (MILP). The complexity of MILP-based solution techniques (e.g., branch-and-bound search) are exponential, leading to computational intractability for large-scale multi-robot teams. Therefore, various hybrid approaches have been proposed that integrated heuristic schedulers within the MILP solver to achieve better scalability characteristics [43]. Koes et al. [44] viewed the problem as a constraint optimization problem and presented a centralized anytime algorithm with error bounds by combining standard MILP solution techniques with domain specific heuristics. Gombolay et al. [22] considered the interval temporal constraints among tasks, along with spatial proximity restrictions on robots, while formulating a MILP model. Their work blended real-time processor scheduling and MILP solvers to develop a fast task sequencer named Tercio, which was tested on KUKA Youbots for assembling a mock air-

plane fuselage. Prorok et al. [45] looked into the problem of finding an optimal distribution of multi-task robots' capabilities among the set of multi-robot tasks.

Gerkey and Mataric [46] implemented and tested a distributed, auction-based, dynamic task allocation technique called MURDOCH, which was built upon a principled, resource centric, publish/subscribe communication model. Nunes and Gini [34] developed auction-based algorithm, TeSSI, to allocate tasks with temporal constraints. TeSSI works both when all the tasks are known upfront and when tasks arrive dynamically. Das et al. [47] presented Consensus Based Parallel Auction and Execution (CBPAE), a distributed algorithm for task allocation in a system of multiple heterogeneous autonomous robots deployed in a healthcare facility. The robots continuously resolve any conflicts in the bids on tasks using inter-robot communication and a consensus process in each robot. Messing et al. [48] formalize a new class of problems named Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) that takes a holistic view of heterogeneous multi-robot coordination by simultaneously considering the problems associated with all four questions (what, who, when, and how): task planning, allocation, scheduling, and motion planning. A unified and interleaved framework named Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS) is proposed to tackle STAP-STC problems by effectively sharing information among system modules.

For situations where task execution is uncertain, Hanna [49] proposed a two-step process that allows robots to take into account the uncertainty when negotiating the allocation of tasks: task selection by Markov decision process (MDP) and allocation using auctions. The MDP uses the notion of expected reward that provides a good trade-off between the reward of selected tasks and the chances to completely execute these tasks.

2.1.2 Scheduling Mixed Human-Robot Teams

As advancements in robot capability progress, they become safer and effective to use in conjunction with humans to complete specialized works, embodying human-robot teams.

Human-robot teams benefit from the distinctive features of these two resources [50]: Humans are efficient in a wide range of tasks and adaptive to changes, while robots are precise and not subject to fatigue. Therefore, these teams combine productivity and flexibility while improving overall working conditions. The close cooperation between both opens up new possibilities for the manufacturing process. Hence, they have been receiving a lot of interest from practitioners in recent years [51].

In robotics, works refer to the need for improving collision detection [52] and the optimization of robots' motion planning [53]. In some contexts, human-factors are critical elements. Thus, several studies report an empirical examination of human-robot trust [54, 55]. Machine learning algorithms are being developed to enhance gesture recognition [56] and human activity prediction [57, 58] so that the robot can identify and adapt to the operators behaviour. There are also efforts put in designing tasks in a collaborative assembly cell, considering the different capabilities of humans and robots [59]. An extensive review of the technological advances and issues related to HRTs is found in [11].

Existing approaches adapted from multi-robot task planning mostly consider humans as agents with assumed or known capabilities, which leads to sub-optimal performance in realistic applications where human capabilities usually change [60, 61]. Based on the furniture assembly process, Rizwan et al. studied the framework of the human-robot collaborative assembly planning, and the dynamic simulation was also conducted in Gazebo using robot operating system (ROS) [62]. For homokinetic joint assembly process, a collaborative human-robot manufacturing cell was developed to reduce the human workload and diminish the strain injury risk [63]. In the augmented environment, based on the real camera images of the operators and the virtual 3D models of the robots, Wang et al. proposed the real-time active collision avoidance method to safeguard the operators under human-robot collaboration [64]. Ding et al. propose a heuristic for the assignment and sequencing of assembly operations of a PLC I/O Module to avoid safety hazards [65]. Nikolakis et al. design a tool that evaluates the assignment alternatives for each operation according to a

utility function composed of many criteria [66].

Recently, Casalino et al. [67] developed a Petri Net model and a scheduler algorithm to simulate and optimise a human-robot team working cell, given the human uncertainty. The scheduler performs an exhaustive search in the reachability graph to prioritise the next task for the robot. They use a prediction algorithm to identify patterns in human activities and adapt the robot schedule accordingly. Zhang et al. [68] proposed a real-time adaptive assembly scheduling approach for human-multi-robot collaboration by modeling and incorporating changing human capability. Xu et al. [61] develop a Bee Algorithm to disassembly tasks in a working cell composed of two operators, a human and a robot. Tasks are classified according to the execution time and difficulty of each operator. The algorithm is multi-objective, considering to minimise the completion time, costs and the difficulty in the execution of tasks. In [12], a learning curve model of human task performance was integrated with genetic algorithms that encode schedules as chromosomes by repeatedly crossing over and mutating the solutions to find the optimal schedule. The results showed that prediction of human performance enhances the ability of the scheduling systems toward better makespan.

2.1.3 Aircraft Maintenance Scheduling

Maintenance as a crucial activity in industry, with its significant impact on costs and reliability, is immensely influential to a company's ability to be competitive in low price, high quality and performance. Any unplanned downtime of machinery equipment or devices would degrade or interrupt a company's core business, potentially resulting in significant penalties and unmeasurable reputation loss [69]. The evolution of modern techniques (e.g., Internet of things, sensing technology, artificial intelligence, etc.) reflects a transition of maintenance strategies from Reactive Maintenance (RM) to Preventive Maintenance (PM) to Predictive Maintenance (PdM).

RM is only executed to restore the operating state of the equipment after failure occurs,

and thus tends to cause serious lag and results in high reactive repair costs [70]. PM is carried out according to a planned schedule based on time or process iterations to prevent breakdown, and thus may perform unnecessary maintenance and result in high prevention costs [71, 72]. In order to achieve the best trade-off between the two, PdM is performed based on an online estimate of the "health" and can achieve timely pre-failure interventions [73, 74]. PdM allows the maintenance frequency to be as low as possible to prevent unplanned RM, without incurring costs associated with doing too much PM. The concept of PdM has existed for many years, but only recently emerging technologies become both seemingly capable and inexpensive enough to make PdM widely accessible [75]. PdM typically involves condition monitoring, fault diagnosis, fault prognosis, and maintenance plans [76]. The enabling technologies have the enhanced potential to detect, isolate, and identify the precursor and incipient faults of machinery equipment and components, monitor and predict the progression of faults, and provide decision-support or automation to develop maintenance schedules.

Aircraft maintenance is performed to prevent or reduce the adverse effect of failures [77]. The aircraft maintenance scheduling is one among the major decisions an airline has to make during its operation. It involves determining which aircraft should fly which segment and when and where each aircraft should undergo different levels of maintenance and checks. and has been a popular application area for operations research studies. Early on, a large-scale mixed integer programming formulation was given in [78], without considering cyclic constraints, heterogeneity in fleet and routine maintenance constraints. Brio et al. [79] introduced a human interaction system to solve the maintenance scheduling problem, with more emphasis placed in human judgment. Hane et al. [80] formulated a basic fleet assignment problem that considered maintenance and crew constraints. However, in [80], only maintenance checks of short duration were considered, with the time frame fixed to one day.

Dekker and Scarf [81] stated that the problem underlying aircraft maintenance schedul-

ing is a job scheduling problem on parallel machines (i.e. maintenance technicians) with precedence, deadline, and machine utilization and availability constraints. Sriram et al. [82] considered the problem faced by an airline needing to construct a 7-day planning horizon cyclic schedule with maintenance constraints for a heterogeneous fleet of aircraft. A hybrid heuristic method combining random search and depth first search was proposed to solve the problem efficiently and quickly. Cho [83] addressed the maintenance scheduling process that is unique to low-observable (LO) aircraft. The LO capabilities of an aircraft degrade over time according to a stochastic process and require continuous maintenance attention. Gavranis and Kozanidis [84] proposed an exact solution method to maximize fleet availability by deciding which aircraft to assign to each fight while meeting certain maintenance requirements. Liu et al. [85] designed and implemented an autonomous system that fuses aircraft's condition, strategy, planning and cost to improve the operational support for aircraft maintenance scheduling. Dinis et al. [15] proposed a framework for the qualitative and quantitative characterization of maintenance work to support Maintenance, Repair, and Overhaul (MRO) organizations in performing capacity planning and scheduling.

2.2 Uncertainty in Stochastic Scheduling

Production environments in the real world are subject to many sources of uncertainty or randomness [86], such as machine breakdowns, unexpected releases of high priority jobs, uncertainty in the processing times, etc. Pinedo [87] first considered tractable stochastic scheduling problems of which the deterministic counterparts are NP-hard and developed policies that minimize the expected weighted sum of job completion times. An overview of methodologies that have been developed to address the problem of uncertainty in production scheduling can be found in [88]. In computation of project activities, most often the uncertainties are quantified by using selected probability distribution functions, which help to convert stochastic values to deterministic ones [89].

Cai et al. [90] studied stochastic scheduling on m parallel identical machines with ran-

dom processing times and investigated the usability of the Shortest Expected Processing Time (SEPT) policy and the Longest Expected Processing Time (LEPT) policy. Ramirez et al. [91] developed an execution delay model for runtime prediction, and designed an adaptive stochastic allocation strategy, named Pareto Fractal Flow Predictor (PFFP). Donti et al. [92] proposed to learn probabilistic machine learning models in a manner that directly captures the ultimate task-based objective for which they will be used, within the context of stochastic programming. Their approach was verified on a real-world electrical grid scheduling task, and a real-world energy storage arbitrage task. Sallam et al. [93] proposed a multi-method approach for solving stochastic resource constrained project scheduling problems. Multi-operator differential evolution (MODE) and discrete cuckoo search (DCS) meta-heuristic approaches are utilized in a single framework with an integration of reinforcement learning to select the best one at each evolutionary process.

In this thesis, two sources of uncertainty are considered: 1) agent-centric: uncertainty in the proficiencies of agents and 2) task-centric: uncertainty in the start time, duration, and cost of tasks. The first form is explored in coordinating human-robot teams. For human-robot teams, the major source of uncertainty comes from the human workers with stochastic behavior and internal learning curves hidden from the scheduler. Therefore, we learn a stochastic policy using policy gradient methods instead of training a deterministic greedy Q-function [94]. Moreover, we utilize Kalman filters [12] to provide the scheduler with robust predictions on future human task proficiency. The second form is explored in aircraft maintenance scheduling. In aircraft maintenance scheduling, both the maintenance task and the event of plane failure are stochastic, leading to stochastic and dynamic task arrival and completion times. To learn a robust scheduler in such input-driven environments, we adopt several variance reduction techniques [95, 96].

2.3 Policy Learning for Combinatorial Optimization

Recently, there have been growing efforts in leveraging machine learning (ML) to solve combinatorial optimization problems [97]. In some works, researchers assume expert knowledge about the optimization algorithm, but wants to alleviate the computational burden by approximating some of those decisions with ML. In these cases, the policy is often learned by imitation learning, thanks to demonstrations. On the other hand, expert knowledge may not be satisfactory and researchers wish to find better policy of making decisions. Thus, ML can come into play to train a model through reinforcement learning.

An active research area can be found in the context of branch-and-bound (B&B) in solving MILPs, with a focus on learning branching policies by supervision or imitation of strong branching (SB), a valid but expensive heuristic scheme. Alvarez et al. [98] used a special type of decision tree to approximate strong branching decisions using supervised learning. Khalil et al. [99] formulated branching variable selection (BVS) as a ranking problem and learn instance-specific proxies of SB. In a different vein, Balcan et al. [100] leveraged existing scoring rules by learning weights to combine them, and performed experiments on special BVS as a classification problem on SB expert decisions. Zarpellon et al. [101] aimed instead at learning a policy that generalizes across heterogeneous MILPs. They proposed a novel imitation learning framework, and introduced new input features and architectures to represent branching. An extensive survey on learning and branching in MILPs can be found in [102].

In the case where one cares about discovering new policies, i.e., optimizing an algorithmic decision function from the ground up, the policy may be learned by reinforcement learning without expert inputs. Bello et al. [103] focus on the traveling salesman problem (TSP) and train a recurrent neural network that, given a set of city coordinates, predicts a distribution over different city permutations, via policy gradient methods. Khalil et al. [27] input the node embeddings learned by a graph neural network (GNN) into a deep

Q-learning agent and achieved better performance than previous heuristics on solving minimum vertex cover, maximum cut and TSPs. Kool et al. [28] combined GNNs and policy gradient methods to learn a efficient policy for TSP and two variants of the Vehicle Routing Problem (VRP).

2.4 Graph Neural Networks

Graph neural networks (GNNs) which aim to extend the deep neural network to deal with arbitrary graph-structured data are introduced in [104]. GNNs learn from unstructured data by representing objects as nodes and relations as edges and aggregating information from nearby nodes. GNNs have been widely applied in graph-based problems such as node classification, link prediction and clustering, and show convincing performance [105].

Research in this area generally falls into two categories, namely spectral domain and non-spectral domain. On one hand, spectral approaches work with a spectral representation of the graphs. Bruna et al. [106] extended convolution to general graphs by finding the corresponding Fourier basis. Defferrard et al. [107] utilized K-order Chebyshev polynomials to approximate smooth filters in the spectral domain. Kipf et al. [108] motivated the choice of convolutional architecture via a localized first-order approximation of spectral graph convolutions and presented a scalable approach for semi-supervised learning on graph-structured data. On the other hand, we also have non-spectral approaches that define convolutions directly on the graph and operate on groups of spatially close neighbors. Hamilton et al. [109] introduced GraphSAGE, a graph neural network that generates embeddings by sampling and aggregating features from a node's local neighborhood. Inspired by attention mechanism, Graph Attention Networks (GATs) [32] are proposed to learn the importance between nodes and its neighbors and fuse the neighbors by normalized attention coefficients. Wang et al. [110] applied graph convolutional networks to point cloud classification and segmentation by exploiting GNN's capability to aggregate information from local neighborhoods. A more comprehensive review of GNN approaches and applications

can be found in [111].

Besides homogeneous graphs, heterogeneous graphs containing different types of nodes and links are also being considerded, yielding the development of heterogeneous GNNs [112]. Heterogeneous GNNs have shown good interpretability and model expressiveness compared to traditional GNNs in scenarios such as graph mining tasks [113, 114], malicious account detection [115] and multi-agent reinforcement learning [116, 117].

2.5 Recurrent Neural Networks for Sequence Prediction

Recurrent Neural Networks (RNNs) are a type of Neural Networks where the output from previous step are fed as input to the current step. RNNs are used to accurately and efficiently train sequence prediction tasks, allowing for non-expert systems to reach high-fidelity prediction [118]. Of different RNN structures, the impact of long short-term memory (LSTM) module has been notable in a wide range of applications including language modeling [119], speech-to-text transcription [120, 121], machine translation [122], and more [123, 124]. Long time lags in certain problems are bridged using LSTMs where they also handle noise, distributed representations, and continuous values [125]. The gate structure of the LSTM cell can effectively slow down the gradient disappearance or explosion that may occur in long sequence problems.

GNNs and RNNs can be used together for prediction of complex models based on time-series data, such as traffic speed prediction [126], action recognition [127] and disease prediction [128]. These applications are often deterministic and do not account for the stochasticity associated with human behavior.

2.6 Summary

There has been a wealth of work in representation learning with graph neural networks. A number of GNN-based approaches have been employed to learn policies for solving combinatorial optimization problems. However, these studies have primarily focused on de-

terministic situations with simple graph structures (e.g., traveling salesman problems). On the other hand, much of the literature pertaining to multi-agent task scheduling still relies on designing application-specific heuristics that utilize the problem structure for generating effective solutions.

What I believe is that the resource optimization literature lacks a unified framework that jointly learns on problem representation and on sequential decision-making process to obtain scalable policies for scheduling under various constraints. My work fills this gap in two ways. 1) We are the first to utilize heterogeneous GNNs for representation learning on complex scheduling problems. The use of heterogeneous GNNs brings a flexible framework that can directly learn from the graph structure of the problem, which has been difficult for other types of deep learning models such as CNNs. 2) We develop imitation and reinforcement learning algorithms for robust learning adjusted to various deterministic and stochastic settings.

CHAPTER 3

SCHEDULING ROBOTS WITH GRAPH ATTENTION NETWORKS

3.1 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 [129]. 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.

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 [30].

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) [27, 28], 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 chapter, we develop a novel model, called RoboGNN scheduler, which is based on graph attention network (GAT) [32], to learn scheduling policies that reason about the underlying simple temporal network (STN) structure [33] 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 warm-starts, and still be applied on large-scale problems that are computationally intractable for exact

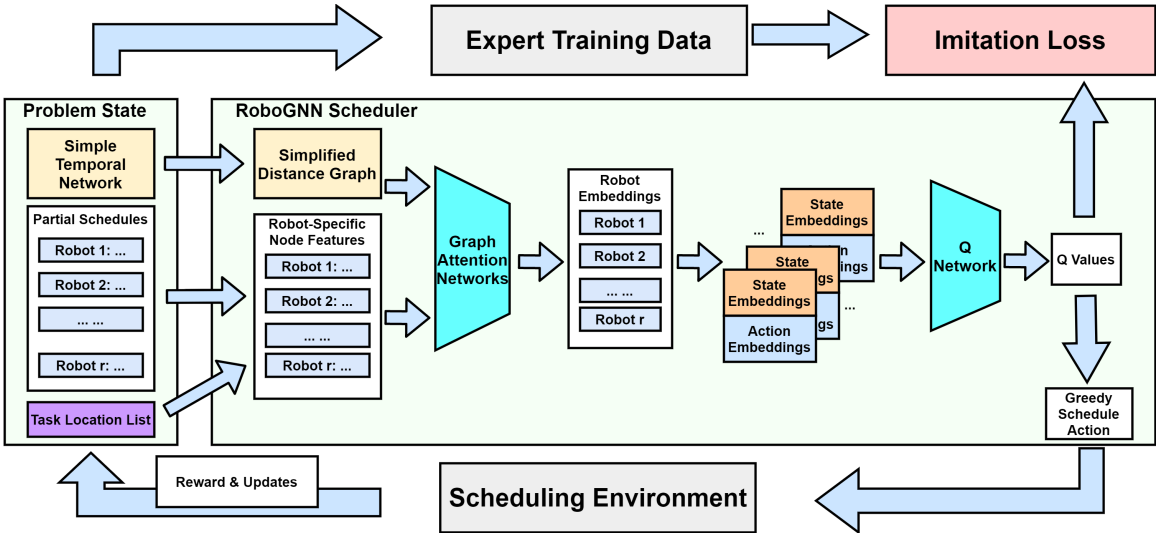


Figure 3.1: The figure depicts the proposed framework, which incorporates graph attention networks and imitation learning for multi-robot scheduling. The RoboGNN scheduler uses a graph attention network, with robot-specific input node features constructed from partial schedules, to extract high level 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.

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 Figure 3.1. 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.

3.2 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 [42], 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 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 Equation 3.1-Equation 3.9. We consider a generic objective function, as application-specific goals vary. In this chapter, 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 3.2 ensures that each task is assigned to only one agent. Equation 3.3-Equation 3.5 ensure that all the temporal constraints are met. Equation 3.6-Equation 3.7 ensure that robots can only perform one task at a time. Equation 3.8-Equation 3.9 account for task locations that can only be occupied by one robot at a time. In section 3.5, 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.

$$\min(z) \tag{3.1}$$

$$\sum_{r \in \boldsymbol{r}} A_{r,i} = 1, \forall \tau_i \in \boldsymbol{\tau} \tag{3.2}$$

$$f_i - s_i = dur_i, \forall \tau_i \in \boldsymbol{\tau} \tag{3.3}$$

$$f_i - s_0 \leq d_i, \forall d_i \in \boldsymbol{d} \tag{3.4}$$

$$s_i - f_j \geq w_{i,j}, \forall w_{i,j} \in \boldsymbol{w} \tag{3.5}$$

$$(s_j - f_i) A_{r,i} A_{r,j} X_{i,j} \geq 0, \forall \tau_i, \tau_j \in \boldsymbol{\tau}, \forall r \in \boldsymbol{r} \tag{3.6}$$

$$(s_i - f_j) A_{r,i} A_{r,j} (1 - X_{i,j}) \geq 0, \forall \tau_i, \tau_j \in \boldsymbol{\tau}, \forall r \in \boldsymbol{r} \tag{3.7}$$

$$(s_j - f_i) X_{i,j} \geq 0, \forall (\tau_i, \tau_j) \in \boldsymbol{L_{same}} \tag{3.8}$$

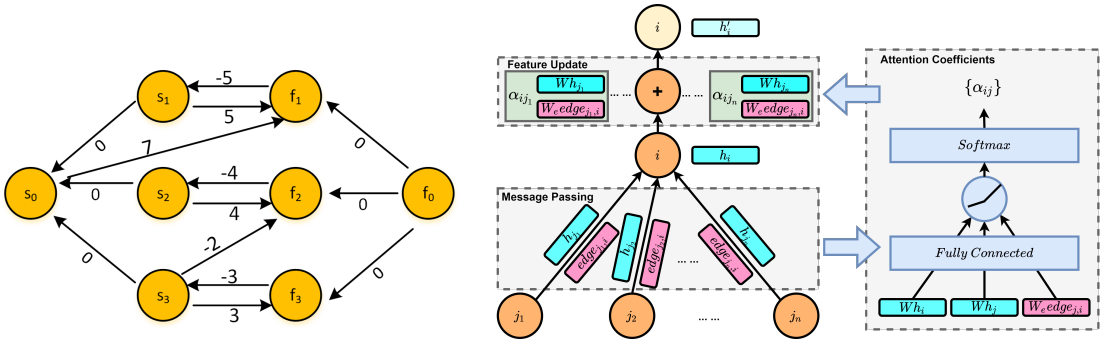
$$(s_i - f_j)(1 - X_{i,j}) \geq 0, \forall (\tau_i, \tau_j) \in \boldsymbol{L_{same}} \tag{3.9}$$

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

3.3 Representation: Graph Attention 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 [130]. 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 work.

Modern GNNs capture the dependence of graphs via message-passing between the



(a) (b)

Figure 3.2: (a) 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. (b) The left-hand side depicts the forward pass of the adapted graph attention layer, 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.

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 [31]. Here, we make use of the graph attention layer (GAT) proposed in [32], 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 3 tasks is shown in Figure 3.2a. For preprocessing purpose, we run Floyd Warshall's all-pairs-shortest-paths algorithm on the original STN to find the minimum distance graph [131]. 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 [27, 28] that only considered the (x,y) position of each node.

Structure Adaptation – The original graph attention network [32] 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 Figure 3.2b: 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 Equation 3.10, 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 at-

tention [32], consisting of K independent GAT layers computing nodes features in parallel and concatenating those features as the output.

$$\vec{h}'_i = \mathrm{ReLU}\Big(\sum_{j \in N(i)} \alpha_{ij} (W\vec{h}_j + W_e edge_{ji}) \Big) \tag{3.10}$$

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 Figure 3.2b using Equation 3.11, 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.

$$\alpha_{ij} = \mathrm{softmax}_j \Bigg(\sigma \left(\vec{a}^T \left[W\vec{h}_i \, \Big\| \, W\vec{h}_j \, \Big\| \, W_e edge_{ji} \right] \right) \Bigg) \tag{3.11}$$

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.

3.4 Learning Scheduling Policies from Expert Demonstrations

3.4.1 MDP Formulation

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 Figure 3.1, 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 Figure 3.1 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 $\tau := \text{argmax}_\tau \hat{Q}_\theta(h_x, h_u)|_{r=r_j}$. Such modification decomposes the scheduling decision into a "picking robot" part followed by a "picking task" part. This allows us to simplify the action space by utilizing a simple agent selector (i.e., selecting the first available robot) and focusing our model on learning to schedule the right task for a given robot.

3.4.2 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 RoboGNN scheduler. However, reinforcement learning relies on finding feasible schedules from scratch to learn useful knowledge. In our problem, most permutations of the schedule are infeasible. As a result, reinforcement learning policies spend much more time than allowed before learning anything of value from exploring infeasible solutions.

Although obtaining optimal solutions of large-scale scheduling problems is compu-

tationally 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 [132].

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 Equation 3.12, 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 $h_{u,ex}$.

$$L_{ex} = \left\| \hat{Q}_\theta(h_x, h_{u,ex}) - R_t^{(n)} \right\|^2 \tag{3.12}$$

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 Equation 3.13, 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.

$$L_{alt} = \frac{1}{N_{alt}} \sum \left\| \hat{Q}_\theta(h_x, h_{u,alt}) - \min(\hat{Q}_\theta(h_x, h_{u,alt}), R_t^{(n)} - q_o) \right\|^2 \tag{3.13}$$

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 [132] in that they only train on the unselected action with the max Q value. Combining Equation 3.12 and Equation 3.13, we calculate the total loss via Equation 3.14, 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.

$$L_{sup} = L_{ex} + \lambda_1 L_{alt} + \lambda_2 L_2 \tag{3.14}$$

3.5 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 demonstra-

tions 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 [133], 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 [134] 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 [135] 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 [22]. 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.

3.5.1 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 Figure 3.3. We also reported mean and

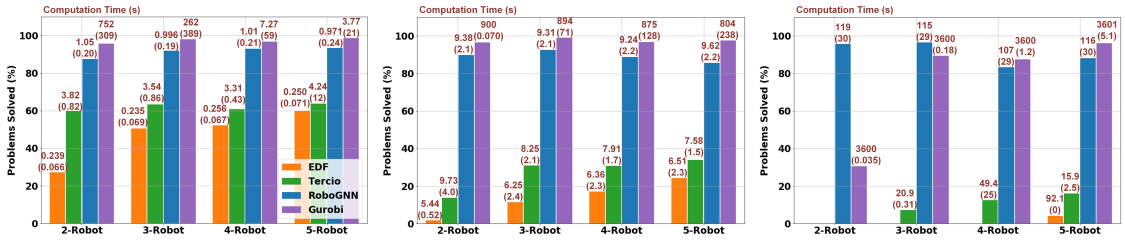
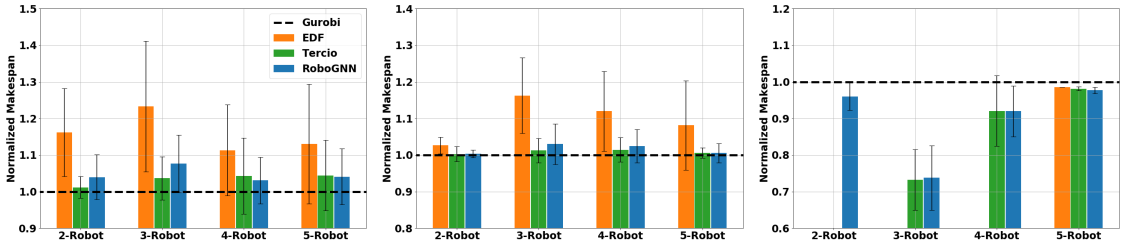


Figure 3.3: Proportion of problems solved for multi-robot scheduling: (a) small problems (16–20 tasks); (b) medium problems (40–50 tasks); (c) large problems (80–100 tasks). Results are grouped in number of robots. Mean and standard deviation of computation times (in parenthesis) for each method is shown above each group's bar.

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.



(a) (b) (c)

Figure 3.4: 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.

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.

3.5.2 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.

Figure 3.4 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 Figure 3.4a and Figure 3.4b. In Figure 3.4c, 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.

3.5.3 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.

3.6 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 Figure 3.5. 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 [136]. A video with a detailed breakdown of the demonstration can be found at http://tiny.cc/y3vgkz.

3.7 RoboGNN Discussion

Combining 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

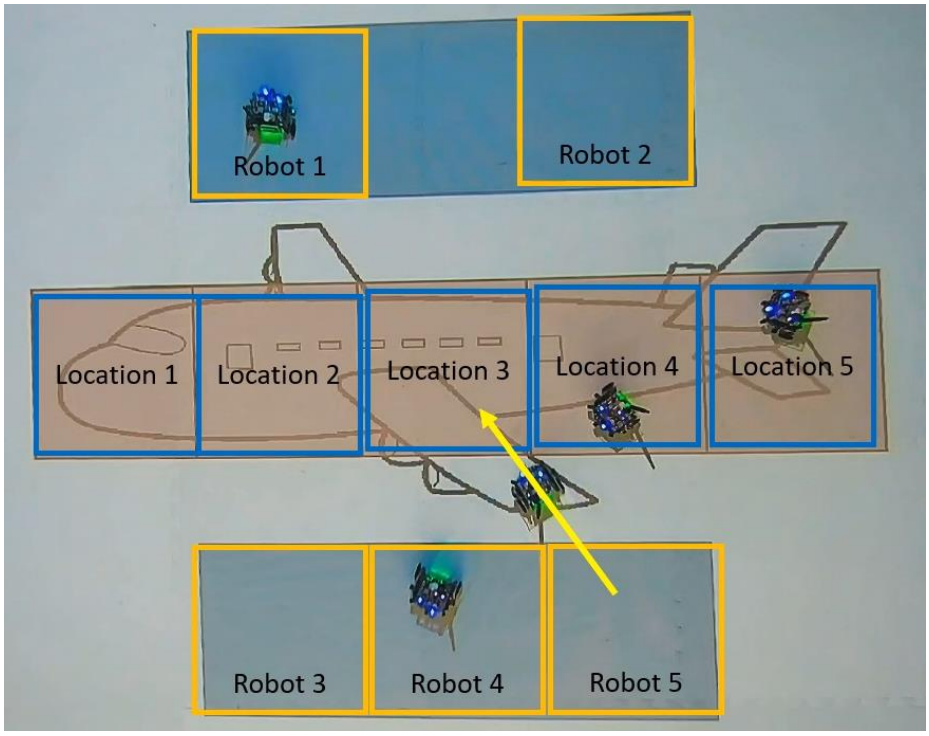


Figure 3.5: This figure depicts our demonstration of a 5-robot team completing tasks for airplane fuselage assembly.

of flexibility in dispatching the robots via the modified STN. We also demonstrated our method on a multi-robot testbed (Figure 3.5). 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.

3.8 Summary

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.

CHAPTER 4
HETEROGENEOUS GRAPH ATTENTION NETWORKS FOR SCALABLE MULTI-ROBOT SCHEDULING

4.1 Introduction

In recent years, deep neural networks have brought about breakthroughs in many domains, including image classification, natural language understanding and drug discovery, as they can discover intricate structures in high-dimensional data without hand-crafted feature engineering [25]. 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 [31]. Yet this research focuses on significantly easier problems with a simpler graphical structure, e.g. the traveling salesman problem (TSP).

In this chapter, 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. Figure 4.1 shows the overall framework of our proposed method. We extend the simple temporal network (STN) [33] 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 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 on small-scale problems 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

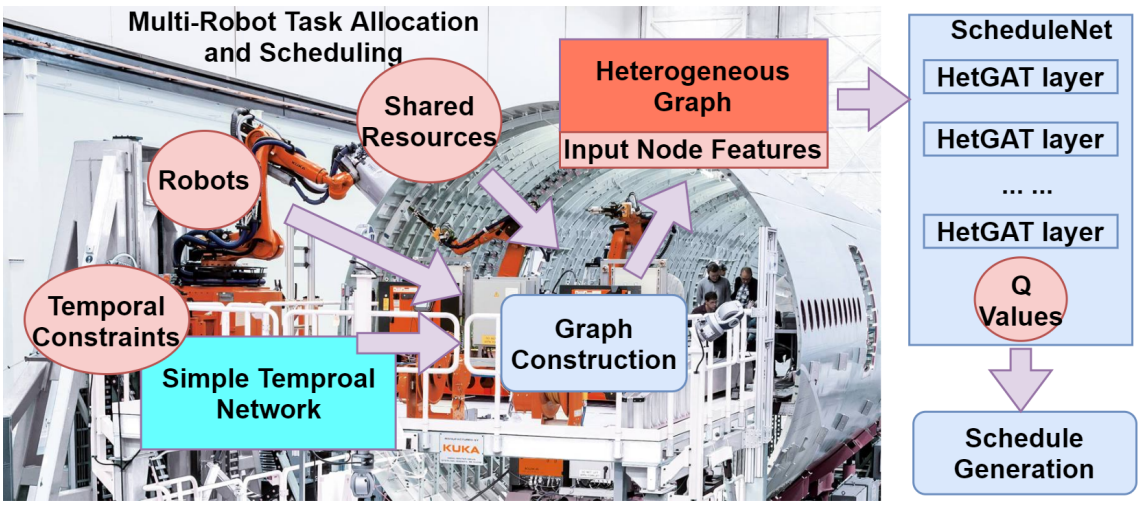


Figure 4.1: Overview of the proposed ScheduleNet, which operates on the heterogeneous graph constructed by augmenting the STN of the problem, and predicts Q-values for scheduling. Courtesy: KUKA Robotics

domain-specific heuristics for robotic applications.

The development of ScheduleNet extends from the RoboGNN scheduler which was limited in modeling only teams of robots with homogeneous task performance (i.e., each robot was equally proficient in completing a given task) and could only consider a more restricted set of shared resource constraints. We build upon this prior work in three key ways. First, we extend ScheduleNet to efficiently reason about coordinating teams of heterogeneous robots (i.e., robots have varying proficiencies in completing each task). Second, we expand the types of spatial constraints from 1-Dimensional (1D) locations to 2D areas with minimum-distance constraints. Third, to improve ScheduleNet's ability to coordinate heterogeneous teams with such spatial constraints, we further augment our approach by proposing novel schedule synthesis strategies. These extensions and the accompanying empirical validation and robot demonstration serve to provide a more holistic view of ScheduleNet's capabilities with emphases on its flexibility, scalability and generalizability. Our results show that ScheduleNet outperforms benchmark approaches when under both the homogeneous and heterogeneous cases. Our extension even solves random problem instances with up to 10 heterogeneous robots and 200 tasks when no other baseline can

solve even a single such instance.

4.2 Problem Overview

4.2.1 Problem Statement

We follow the problem statement presented in section 3.2, with an extension that allows robots to have heterogeneous task proficiency. That is, each task τ_i takes a certain amount of time $dur_{i,r}$ for robot r to complete, as shown in Equation 4.1.

$$f_i - s_i = \sum_{r \in \boldsymbol{r}} dur_{i,r} A_{r,i}, \forall \tau_i \in \boldsymbol{\tau} \tag{4.1}$$

Additionally, $\boldsymbol{L_{same}}$ is replaced with $\boldsymbol{L_{proximity}}$. $\boldsymbol{L_{proximity}}$ is the set of task pairs, $<\tau_i, \tau_j>$, that should be separated along the time axis due to the presence of pairwise proximity constraints on agents performing tasks at the corresponding locations in $\boldsymbol{Loc}$.

We begin our experiments in section 4.4 by considering the problem of coordinating a set of homogeneous robots to complete a set of tasks given temporal constraints and 1D task location constraints (i.e., no two robots can be in the same place at the same time). Here, homogeneity in robot teams refers to teams comprised of robots that are equally proficient in completing a task (i.e., $dur_{i,r} = dur_i, \forall r \in \boldsymbol{r}, \forall \tau_i \in \boldsymbol{\tau}$). This modeling setup is motivated by common manufacturing scenarios in which work locations are along a line with robots moving across a rail to perform assembly tasks of a large workpiece, e.g., the Boeing 777 Fuselage Automated Upright Build process [137]. Under this setting, $\boldsymbol{L_{proximity}}$ consists of all task pairs that require the same location to be completed.

Second, we examine scheduling problems involving heterogeneous robots moving with 2D proximity constraints (section 4.5). The relaxation to heterogeneous robot teams allows for the full expressivity where $dur_{i,r} \neq dur_i$ in general. Furthermore, expanding from 1D location constraints to 2D proximity constraints allows us to model an open factory floor concept where a certain distance must be maintained between robots while executing

tasks. In these experiments, we extend $\boldsymbol{L_{proximity}}$ to include task pairs whose locations fall within the minimum allowed safety distance.

As z varies depending on application-specific goals, we mainly 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, in subsection 4.4.4, 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 [138].

4.2.2 Schedule Generation

Our learned policy 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 Chapter 2 that this process achieves better performance than using decision-step-based generation.

Algorithm 1 illustrates the process of generating schedules for a problem instance using ScheduleNet via scheduling-through-simulation. 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, the policy picks a robot (denoted as the $pickRobot$ function) from $\boldsymbol{r_{avail}}$ and 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. This task allocation step repeats until no robot is available; then, the simulation moves to the next time step, $t + 1$. When considering a team of homogeneous robots, $pickRobot$ can be implemented as simply picking robots from $\boldsymbol{r_{avail}}$ in the same order as their initial index or another static priority queue. However, we empirically found that such a static strategy for ScheduleNet was not efficient for the more difficult problem of task allocation with hetero-

Algorithm 1: Solve a problem instance using ScheduleNet via scheduling-through-simulation

Input: Problem components $<\boldsymbol{r}, \boldsymbol{\tau}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{Loc}>$, maximum allowed time t_{max}
Output: A schedule generated by ScheduleNet

Initialize all robots' pratial schedules as $\varnothing$;
Initialize problem state x with $<\boldsymbol{r}, \boldsymbol{\tau}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{Loc}>$;
$t \leftarrow 0$;
while $t \leq t_{max}$ **do**
    Collects all available robots at t into $\boldsymbol{r_{avail}}$;
    $r \leftarrow \texttt{pickRobot}(\boldsymbol{r_{avail}})$;
    while $r \neq NULL$ **do**
        Collect all unscheduled tasks at t into $\boldsymbol{\tau_{avail}}$;
        if $\boldsymbol{\tau_{avail}} \neq \varnothing$ **then**
            Build the heterogeneous graph g from x;
            Generate input features for nodes in g;
            Run ScheduleNet on g to predict $Q_\theta(x, u)$;
            $\tau \leftarrow \text{argmax}_{\tau \in \boldsymbol{\tau_{avail}}} Q_\theta(x, u)|_{u=<\tau,r>}$;
            Append τ into r's partial schedule and update x;
            if x *becomes infeasible* **then**
                Exit with an infeasible schedule;
            else if *all tasks have been assigned* **then**
                Exit with a feasible schedule;
            end
            Remove r from $\boldsymbol{r_{avail}}$;
            $r \leftarrow \texttt{pickRobot}(\boldsymbol{r_{avail}})$;
        else
            break;
        end
    end
    $t \leftarrow t + 1$;
end
Exit with an infeasible schedule;

geneous robot teams. As such, we developed a set of additional task allocation strategies for dynamically picking heterogeneous robots from $\boldsymbol{r_{avail}}$:

- *First available* – Pick the first robot in $\boldsymbol{r_{avail}}$ according to their original index.
- *Minimum average time on unscheduled tasks* – Compute the average time it takes for each robot in $\boldsymbol{r_{avail}}$ to complete unscheduled tasks, and pick the robot with the smallest such time.
- *Minimum time on any one unscheduled task* – Find the minimum time it takes for each robot in $\boldsymbol{r_{avail}}$ to complete any one unscheduled task, and pick the robot with the smallest such time.
- *Minimum average time on all tasks* – Compute the average time it takes for each robot in $\boldsymbol{r_{avail}}$ to complete all tasks, both scheduled and unscheduled, and pick the robot with the smallest such time.

When solving a given problem instance, we run ScheduleNet in parallel with each task allocation strategy variants for the $pickRobot$ function. Among the feasible solutions produced by the same model with each of the four strategies, we keep the one that yields the best objective function score. This result ensemble of different robot-picking variants proves to find not only more feasible schedules, but also schedules with better makespans than any single strategy alone, as each strategy may work better than another in certain problems but not the others.

4.3 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-values of state-action pairs as its output node features. In this section, we first describe how to construct the heterogeneous graph given a problem state, x_t, starting with homogeneous robot teams and then introducing necessary extensions for heterogeneous robots. Next, 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 (HetGAT) layer.

4.3.1 Heterogeneous Graph Representation

Homogeneous Robot Case

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 [139]. STNs also allow for encoding set-bounded uncertainty. However, as we develop multiple homogeneous 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 [130]. 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 2. 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 $(srcName \xrightarrow{edgeName} dstName)$.

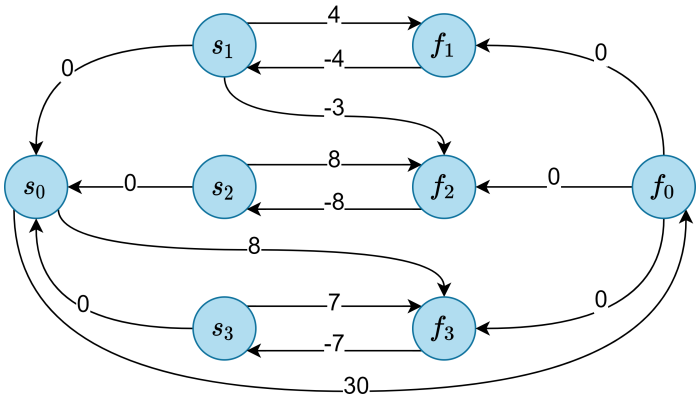
In traditional STN formulations, each task, τ_i, is represented by two event nodes: its start time node, s_i, and finish time node, f_i. The directed, weighted edges encode the temporal constraints associating corresponding nodes. Under the homogeneous robot team setting, task duration is deterministic without knowing the actual assignment. Exploiting this fact, we develop a **novel simplification trick** to reduce the model complexity. That is,

Algorithm 2: Construct the heterogeneous graph for modeling homogeneous robot teams

Input: STN, locations $\boldsymbol{Loc}$, robots $\boldsymbol{r}$ and their partial schedules, available actions $\boldsymbol{u_{avail}}$

Output: Heterogeneous graph representation

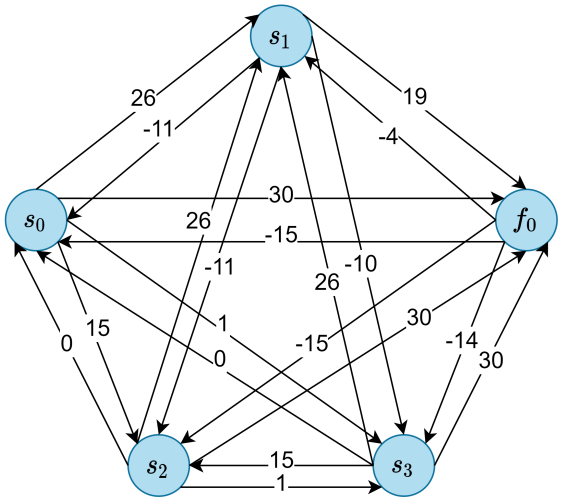
Run Johnson's algorihtm on STN to find its minimum distance graph, g_d;
Remove all f_i's from g_d, except f_0;
Use g_d as the new STN and s_i as the task node of τ_i;
foreach *robot* r_j **do**
    Add a robot node, r_j;
    foreach τ_m *assigned to* r_j **do**
        Add an edge $\tau_m \rightarrow r_j$;
    end
end
Connect robot nodes with each other;
foreach *location* L_k **do**
    Add a location node, L_k;
    foreach τ_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.



(a) Original STN

Src \ Dst	s_0	f_0	s_1	f_1	s_2	f_2	s_3	f_3
s_0	0	30	26	30	15	23	1	8
f_0	-15	0	-4	0	-15	-7	-14	-7
s_1	-11	19	0	4	-11	-3	-10	-3
f_1	-15	15	-4	0	-15	-7	-14	-7
s_2	0	30	26	30	0	8	1	8
f_2	-8	22	18	22	-8	0	-7	0
s_3	0	30	26	30	15	23	0	7
f_3	-7	23	19	23	8	16	-7	0

(b) Table of shortest distances from each source (src) node to each destination (dst) node



(c) Simplified Minimum Distance Graph

Figure 4.2: An example STN consisting of 3 tasks: (a) the original STN with placeholder start and finish nodes, s_0 and f_0; (b) The shortest distances between all pairs of source (src) and destination (dst) nodes found by an all pairs shortest path (APSP) algorithm, with blue denoting the nodes/edges that are maintained in the simplified graph and orange denoting nodes/edges that are pruned; (c) the simplified minimum distance graph with f_i removed for each task, with the duration of each task encoded in the input node features

after running Johnson's algorithm [140] 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 duration now serving as its node feature. Figure 4.2 illustrates this process with an example problem consisting of 3 tasks. Tasks 1, 2, and 3 are shown with durations 4, 8, and 7, respectively. Task 3 has a deadline constraint: $f_3 \leq 8$. There is a wait constraint between task 1 and task 2: $s_1 \geq f_2+3$. Distances in the blue cells of Figure 4.2b are used to construct the graph in Figure 4.2c. The representation shown in Figure 4.2c effectively describes the temporal constraint representation in Figure 4.2a for the purposes of performing an all-pairs shortest path computation as input to our GNN model. The task durations represented by edges in Figure 4.2a are preserved implicitly in the graph edges shown in Figure 4.2c and are captured by our GNN as node features for the corresponding tasks. 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 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 task 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.

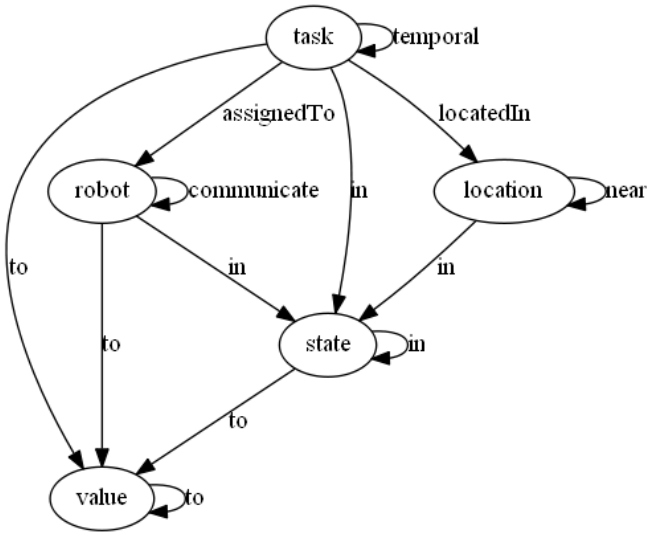
As the Q-function is based on state-action pairs, we also expect the network to learn a

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 the number of total tasks, the number of currently scheduled tasks, the number of robots and the number of locations.

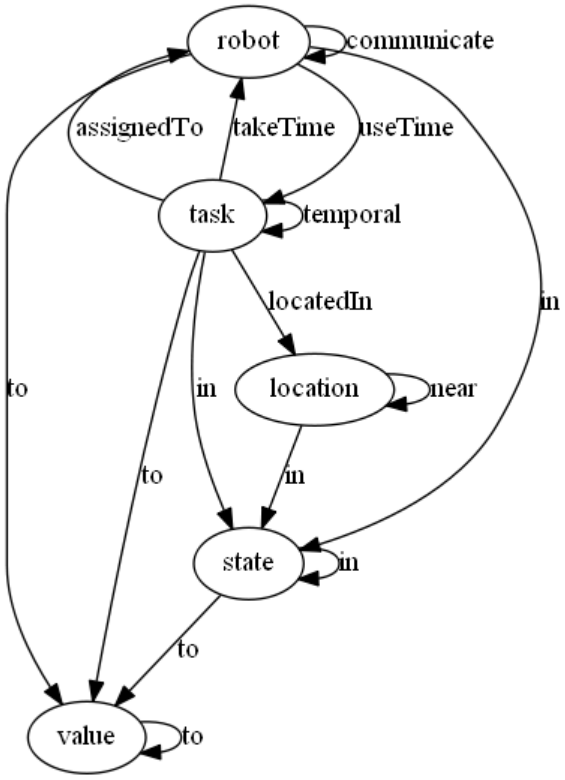
Once the node embeddings are computed using the heterogeneous graph, it is possible to learn a separate Q network consisting of several fully-connected (FC) layers to predict the Q-value of a state-action pair, taking as input the concatenation of embeddings from corresponding state, task, and robot 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 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. The metagraph (or network schema) of the graph constructed with Algorithm 2 is shown in Figure 4.3a, which summarizes all the node types and edge types.

Extension for Heterogeneous Task Completion

In a setting of heterogeneous robot teams, the duration of a task depends on the robot which is assigned to the task. We recall that our problem state consists of a partial schedule in



(a) Homogeneous Robot Team Metagraph



(b) Heterogeneous Robot Team Metagraph

Figure 4.3: Metagraph of the heterogeneous graph built from the STN by adding robot, location, state, and value nodes: (a) team of homogeneous robots; (b) team of heterogeneous robots

which some tasks have already been assigned an agent and then sequenced. For those tasks, we know their duration, which is given by the assigned robot (i.e., $dur_{i,r}$ for task τ_i assigned robot r). However, the duration of unscheduled tasks is yet to be determined, as no robot has been assigned. As such, Equation 4.1 can only be described with a relaxed set bound as shown in Equation 4.2. Here $dur_{i,min}, dur_{i,max}$ are the minimum and maximum amounts of time task τ_i can be finished. Specifically, $dur_{i,min} = \min_r dur_{i,r}$ and $dur_{i,max} = \max_r dur_{i,r}$.

$$dur_{i,min} \leq f_i - s_i \leq dur_{i,max}, \forall \tau_i \in \{unscheduled\} \tag{4.2}$$

Unfortunately, this set bound nullifies our novel simplification trick in Section 3.3.1 for reducing the graph's–and in turn our algorithm's–complexity. To recover our simplification in this more expressive setting, we extend the set of task node features to include multiple descriptive statistics describing the task's possible completion times among all robots where the size is non-parametric in the number of robots. Specifically, the input features of task τ_i consists of the minimum, maximum, mean and standard deviation of $\{dur_{i,r}, \forall r \in \boldsymbol{r}\}$. This modeling approach achieves a potent balance between model complexity and expressivity.

Additionally, we encode information about completion times for unscheduled tasks by augmenting the heterogeneous graph obtained from Algorithm 2. We add two new edge types between nodes of unscheduled tasks and robot nodes: $(task \xrightarrow{takeTime} robot)$ and $(robot \xrightarrow{useTime} task)$. The edge attribute encodes the time used for a robot to complete the task connected via this edge.

A further change we make to Algorithm 2 concerns the handling of edges between location nodes. Because locations are expanded to 2D spatial areas in the heterogeneous robot case, we only connect locations that fall within the minimum allowed safety distance to represent the proximity constraints, instead of connecting location nodes with each other.

Figure 4.3b shows the augmented metagraph with heterogeneous robot teams, including the newly-added edge types between robot nodes and task nodes. The edge type

$(location \xrightarrow{near} location)$, although shares the same name as in Figure 4.3a, now encodes the 2D proximity constraints.

4.3.2 Heterogeneous Graph Attention Layer

Homogeneous Robot Case

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 process the input feature from the source node, N_{src}, and sends the computation result to the destination node, N_{dst}'s mailbox. S is the input feature dimension of N_{src}, and D is the output feature dimension of N_{dst}. In the case that several edge types share the same name, we use $W_{srcName,edgeName}$ to distinguish between them. For example, we distinguish edge types coming into the state nodes by $W_{task,in}$, $W_{robot,in}$, $W_{loc.,in}$ and $W_{state,in}$. As for edge type $(task \xrightarrow{temporal} task)$ which is the only weighted edge in our heterogeneous graph formulation of homogeneous robots, the edge attribute, $edge$, is also sent after transformed by $W_{tempEdge} \in \mathbb{R}^{D \times 1}$.

Feature reduction happens inside each node's mailbox. For each edge type that a node has, the HetGAT layer computes per-edge-type aggregation result by weighing received messages, stored in its mailbox, along the same edge type with normalized attention coefficients that are feature-dependent and structure-free. Those results are then merged to compute a node's output feature. We note that, in the case of coordinating teams of homogeneous robots, task type nodes only ever serve as destination nodes for other task nodes. Task nodes can serve as source nodes for non-task type nodes (e.g., robot nodes). This flow of information from task nodes to robot nodes enables us to extract embeddings for each robot. Embeddings for tasks are extracted from the underlying STN, which already captures information regarding the robots' homogeneous task completion times. The feature update formulas of different node types are listed in Equation 4.3-Equation 4.7.

$$\text{Task } h'_i = \sigma\Big(\sum_{j \in N_{temporal}(i)} \alpha_{ij}^{temporal} (W_{temporal} h_j + W_{tempEdge} edge_{ji}) \Big) \tag{4.3}$$

$$\text{Robot } h'_i = \sigma\Big(\sum_{j \in N_{assginedTo}(i)} \alpha_{ij}^{assignedTo} W_{assignedTo} h_j + \sum_{k \in N_{comm.}(i)} \alpha_{ik}^{comm.} W_{comm.} h_k \Big) \tag{4.4}$$

$$\text{Location } h'_i = \sigma\Big(\sum_{j \in N_{locateIn}(i)} \alpha_{ij}^{locatedIn} W_{locatedIn} h_j + \sum_{k \in N_{near}(i)} \alpha_{ik}^{near} W_{near} h_k \Big) \tag{4.5}$$

$$\text{State } h'_i = \sigma\Big(\sum_{j \in N_{task,in}(i)} \alpha_{ij}^{task,in} W_{task,in} h_j + \sum_{k \in N_{robot,in}(i)} \alpha_{ik}^{robot,in} W_{robot,in} h_k + \sum_{m \in N_{loc.,in}(i)} \alpha_{im}^{loc.,in} W_{loc.,in} h_m + W_{state,in} h_i \Big) \tag{4.6}$$

$$\text{Value } h'_q = \sigma\Big(W_{task,to} h_t + W_{robot,to} h_r + W_{state,to} h_s + W_{value,to} h_q \Big) \tag{4.7}$$

In Equation 4.3-Equation 4.7, $N_{edgeName}(i)$ is the set of incoming neighbors of node

i along a certain edge type, and $\sigma()$ represents the ReLU nonlinearity. Prior work has shown that attention mechanisms are beneficial for representation learning on homogeneous graphs [141, 28]. Thus, we extend the attention models from prior work to reason about task scheduling with heterogeneous graph networks. Specifically, 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 Equation 4.8, where $\vec{a}_{temporal}^{T}$ 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.

$$\alpha_{ij}^{temp.} = \text{softmax}_j\Bigg(\sigma'\Big(\vec{a}_{temp.}^{T} \Big[W_{temp.}\vec{h}_i || W_{temp.}\vec{h}_j || W_{tempEdge} edge_{ji}\Big]\Big)\Bigg) \tag{4.8}$$

For edge types connecting the same type of nodes, the attention coefficients can be computed by Equation 4.9.

$$\alpha_{ij}^{edgeName} = \text{softmax}_j\Bigg(\sigma'\Big(\vec{a}_{edgeName}^{T} \Big[W_{edgeName}\vec{h}_i || W_{edgeName}\vec{h}_j\Big]\Big)\Bigg) \tag{4.9}$$

However, Equation 4.9 does not hold for edges where the source node, h_j, and destination node, h_i, are of different node types. Take the edge type $(task \xrightarrow{assignedTo} robot)$ as an example, the message passing weights, $W_{assignedTo}$, are only defined and trained for processing the source node type features (of task nodes) and are thus not adequate for processing the destination node type features (of robot nodes) for attention computa-

tion. Therefore, we change Equation 4.9 into Equation 4.10 by using both $W_{edgeName}$ and $W_{dstType}$ to account for differing types of source and edge nodes. While these additional parameters improve model expressivity, there is a cost in terms of computational memory and speed. In practice, we find that we can achieve a helpful tradeoff between expressivity and computational costs by employing weight sharing. Specifically, we set $W_{dstType}$ to be equal to $W_{comm.}$, W_{near}, and $W_{state,in}$ when the destination node type is robot, location and state, respectively.

$$\alpha_{ij}^{edgeName} = \mathrm{softmax}_j\Bigg(\sigma'\Big(\vec{a}_{edgeName}^{T}\left[W_{dstType}\vec{h}_i || W_{edgeName}\vec{h}_j\right]\Big)\Bigg) \tag{4.10}$$

To stabilize the learning process, we utilize the multi-head attention proposed in [141], adapting it to fit the heterogeneous case. We use K independent HetGAT layers to compute node features in parallel and then merge the results as the multi-headed output via the concatenation operation for each multi-head layer in ScheduleNet, except for the last layer which employs averaging. Considering that ScheduleNet utilizes a fully convolutional structure where the last graph layer directly predicts Q-values as the 1-dimensional output feature of value nodes, merging multi-head results with concatenation is no longer viable for the last layer as it would give a K-dimensional output.

Extension for Heterogeneous Task Completion

Because the newly-added edge type $(robot \xrightarrow{useTime} task)$ only accounts for unscheduled tasks, the feature update formula for scheduled tasks remains the same as Equation 4.3. For unscheduled tasks, we change Equation 4.3 by including terms accounting for the message coming through the new edge type, as shown in Equation 4.11.

$$
\begin{aligned}
h_i' = \sigma\Big(& \sum_{j \in N_{temporal}(i)} \alpha_{ij}^{temporal} (W_{temporal} h_j \\
& + W_{tempEdge} edge_{ji}) \\
& + \sum_{k \in N_{useTime}(i)} \alpha_{ik}^{useTime} (W_{useTime} h_k \\
& + W_{useTimeEdge} edge_{ki}^{'} \Big)
\end{aligned} \tag{4.11}
$$

In Equation 4.11, $edge_{ki}^{'}$ is the attribute of the new edge, and the corresponding attention coefficient, $\alpha_{ik}^{useTime}$, is computed by Equation 4.12.

$$
\begin{aligned}
\alpha_{ik}^{useTime} = \mathrm{softmax}_k \Bigg(& \sigma' \Big(\vec{a}_{useTime}^{T} \\
& \Big[W_{temp.} \vec{h}_i || W_{useTime} \vec{h}_k || W_{useTimeEdge} edge_{ki}^{'} \Big] \Big) \Bigg)
\end{aligned} \tag{4.12}
$$

Similarly, the addition of new edge type $(task \xrightarrow{takeTime} robot)$ changes the feature update equation of robot nodes from Equation 4.4 in the homogeneous case to Equation 4.13 in the heterogeneous case.

$$
\begin{aligned}
h_i' = \sigma\Big(& \sum_{j \in N_{assginedTo}(i)} \alpha_{ij}^{assignedTo} W_{assignedTo} h_j \\
& + \sum_{k \in N_{comm.}(i)} \alpha_{ik}^{comm.} W_{comm.} h_k \\
& + \sum_{m \in N_{takeTime}(i)} \alpha_{im}^{takeTime} (W_{takeTime} h_m \\
& + W_{takeEdge} edge_{mi}^{''} \Big)
\end{aligned} \tag{4.13}
$$

In Equation 4.13, $edge''_{mi}$ is the attribute of the corresponding edge, and the corresponding attention coefficient, $\alpha_{im}^{takeTime}$, is computed according to Equation 4.14.

$$\alpha_{im}^{takeTime} = \text{softmax}_m\Bigg(\sigma'\Big(\vec{a}_{takeTime}^{T} \Big[W_{comm.}\vec{h}_i || W_{takeTime}\vec{h}_m || W_{takeTimeEdge} edge''_{mi}\Big]\Big)\Bigg) \tag{4.14}$$

Even though locations are extended from 1D to 2D areas, Equation 4.5 still applies to location nodes. Because now $N_{near}(i)$ only considers neighbor locations falling within the allowed safety distance instead of all locations, and $W_{near}, \alpha_{ik}^{near}$ will learn to encode the corresponding proximity constraints. Finally, we note that the update equations for state and value nodes (Equation 4.6 and Equation 4.7) remain the same as in the homogeneous robot case.

4.4 Experimental Results on Homogeneous Robots

In this section, we evaluate the performance of ScheduleNet on problems involving homogeneous robot teams. We show the results of optimizing a generic objective function, which is the minimization of total makespan. In subsection 4.4.4, we also consider the application-specific objective function mentioned in subsection 4.2.1, in which we minimize the weighted sum of task completion time, to investigate how ScheduleNet generalizes under different use cases.

4.4.1 Dataset

To evaluate the performance of ScheduleNet, we generate random problems based on [142]. 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.

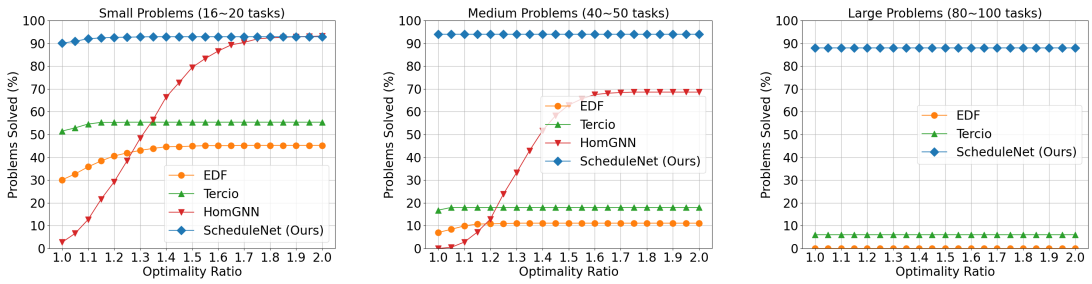
For each team configuration, problems are generated in three scales: small (16-20 tasks), medium (40-50 tasks) and large (80-100). For each problem scale, we generate 1,000 problems for testing. To train the ScheduleNet model, we generate another 1,000 small problems with two-robot teams. We run Gurobi with a cutoff time of 15 minutes on generated problems to serve as exact baselines for test set and expert demonstrations for training set of imitation learning. 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.

4.4.2 Benchmark

We benchmark ScheduleNet against these 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 [22]. 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 [22].



(a) (b) (c)

Figure 4.4: Evaluation results on problems of two-robot teams of homogeneous robots: (a) Small problems; (b) Medium problems; (c) Large problems

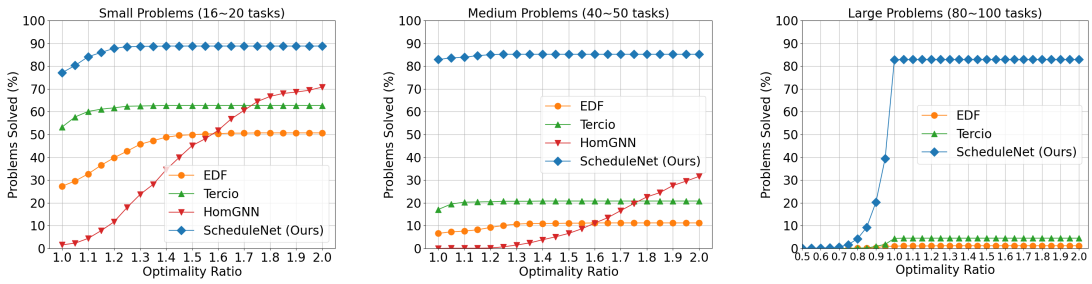
- *HomGNN* – A neural-network-based method proposed in [143]. Their method uses a homogeneous GNN to exact problem embedding from the STN, and a separate Q-network consisting of two FC layers to predict the Q-value. We denote this model as HomGNN and use the same hyper-parameters in [143].

- *Exact* – Gurobi, a commercial optimization solver widely used for mixed integer linear programming. Its results represent the exact baseline.

4.4.3 Evaluation Results

Metrics – For minimizing the makespan, we use the following metric for evaluation purpose. **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 within cutoff time, 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 [144] and Deep Graph Library [145]. 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



(a) (b) (c)

Figure 4.5: Evaluation results on problems of five-robot teams of homogeneous robots: (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

feature dimension of hidden layers = 64, and the number of heads = 8. We set γ = 0.99, D = 3.0 and used Adam optimizer [134] 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 were conducted on a Quadro RTX 8000 Graphics Processing Unit (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. Figure 4.4-Figure 4.6 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 ($r \geq 1.8$), 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,

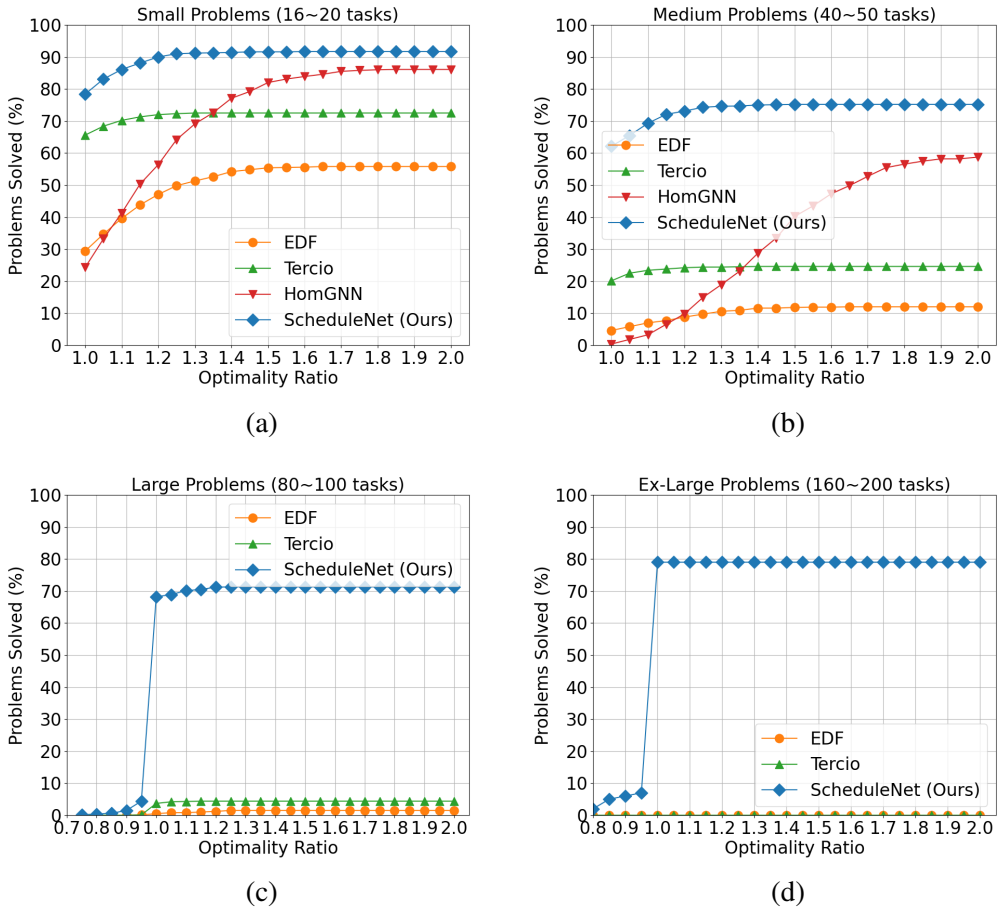


Figure 4.6: Evaluation results on problems of ten-robot teams of homogeneous robots: (a) Small problems; (b) Medium problems; (c) Large problems; (d) Ex-Large problems. In the Large and Ex-Large problems cases, ScheduleNet is able to find solutions that outperform Gurobi as denoted by data points left of a 1.0 optimality ratio

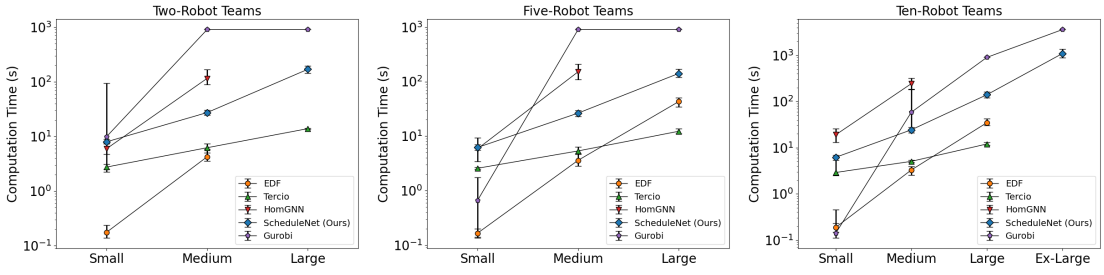
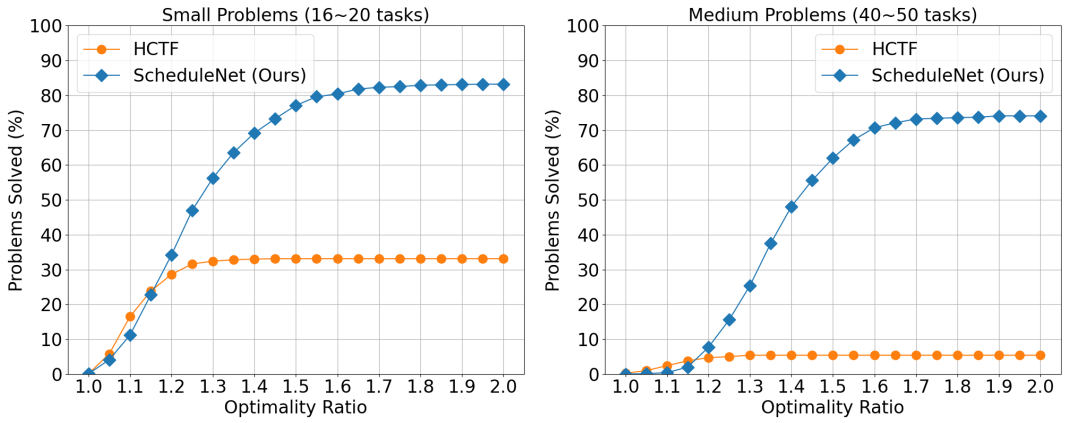


Figure 4.7: Running time statistics on different problems of homogeneous robots: (a) Two-robot teams; (b) Five-robot teams; (c) Ten-robot teams. Error bars denote the 25th and 75th percentile. Results for EDF, Tercio, and HomGNN are not shown in cases when no solutions are found within the allowed cutoff time

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 Figure 4.5c, Figure 4.6c and Figure 4.6d. 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 Figure 4.7, 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 cuttoff 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 evaluation. As a comparison,



(a) (b)

Figure 4.8: Evaluation results of minimizing the weighted sum of completion times on five-robot teams of homogeneous robots: (a) Small problems; (b) Medium problems

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

4.4.4 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 subsection 4.4.1. Additionally, each task was associated with a real number cost, c, drawn from a uniform distribution in the interval [1, 10]. This cost is added to the input features of task nodes. 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. Figure 4.8 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.

4.5 Experimental Results on Heterogeneous Task Completion

In this section, we evaluate the performance of ScheduleNet for coordinating robots that are heterogeneous in task completion time with the objective of minimizing the team's makespan.

4.5.1 Dataset and benchmark

We generate random problems following the same setting and hyper-parameters as described in Section subsection 4.4.1, with the following two differences:

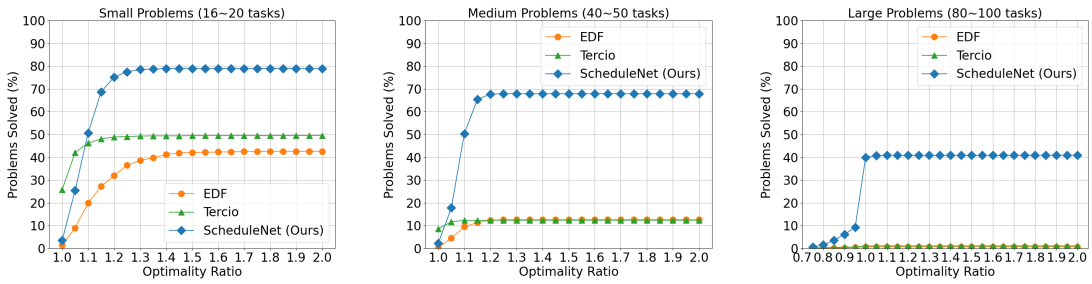
1. For each task, τ_i, we sample a $mean$ value from [1, 10] and a gap value from [1, 3], both with uniform distributions. Then we sample $dur_{i,r}$ for each robot from a uniform distribution in the interval $[mean - gap, mean + gap]$ (clamped at [1, 10] if applicable).

2. The task locations are randomly sampled from the 2D map. We use 2x2 for two-robot teams, 3x3 for five-robot teams, and 5x5 for ten-robot teams. The safety distance is set to 1. If $|Loc_i - Loc_j| \leq 1$, then $(\tau_i, \tau_j) \in \boldsymbol{L_{proximity}}$.

Considering that HomGNN [143] is not designed for handling heterogeneous task completion among robots, we benchmark ScheduleNet against the remaining set of methods described in subsection 4.4.2: EDF, Tercio and Exact (i.e., a MILP solved by Gurobi).

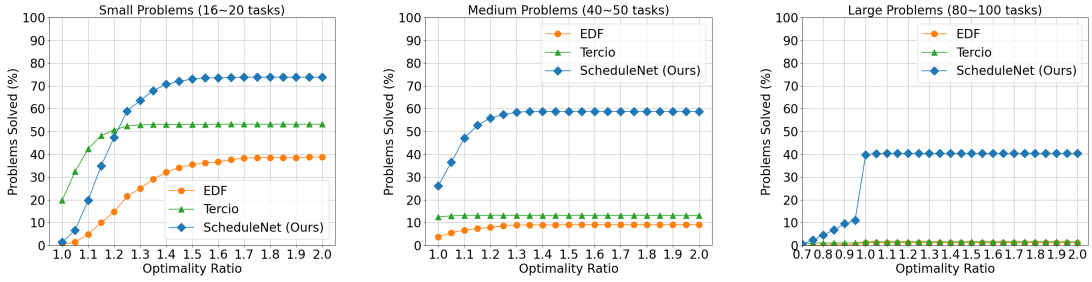
4.5.2 Evaluation Results

Model Details – For performance evaluation, we use the same **M1** metric as used in subsection 4.4.3. The ScheduleNet model also consisted of 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 $\gamma = 0.95$, $D = 3.0$ and



(a) (b) (c)

Figure 4.9: Evaluation results on problems of two-robot teams of heterogeneous robots: (a) Small problems; (b) Medium problems; (c) Large problems



(a) (b) (c)

Figure 4.10: Evaluation results on problems of five-robot teams of heterogeneous robots: (a) Small problems; (b) Medium problems; (c) Large problems

used Adam optimizer. The training procedure used a learning rate of 3×10^{-4}, $\lambda_1 = 0.9$, $\lambda_2 = 0.3$, $q_o = 3.0$ and batch size = 8. Both training and evaluation were conducted on a Quadro RTX 8000 GPU.

Same as in the homogeneous robot case, we trained the ScheduleNet for heterogeneous robot teams on small problems of two-robot teams and evaluate the same model on varying problem scales and team configurations. Evaluation results using **M1** are shown in Figure 4.9-Figure 4.11 where optimality ratios range from 1 to 2 with intervals of 0.05 by default.

For small and medium problems, ScheduleNet outperformed EDF and Tercio for medium-to-large optimality ratios ($r \geq 1.2$) other than small problems of ten-robot teams, and obtained results close to Tercio for $r \geq 1.5$ while consistently beating EDF. The improvement in performance of ScheduleNet over the baseline models was particularly significant

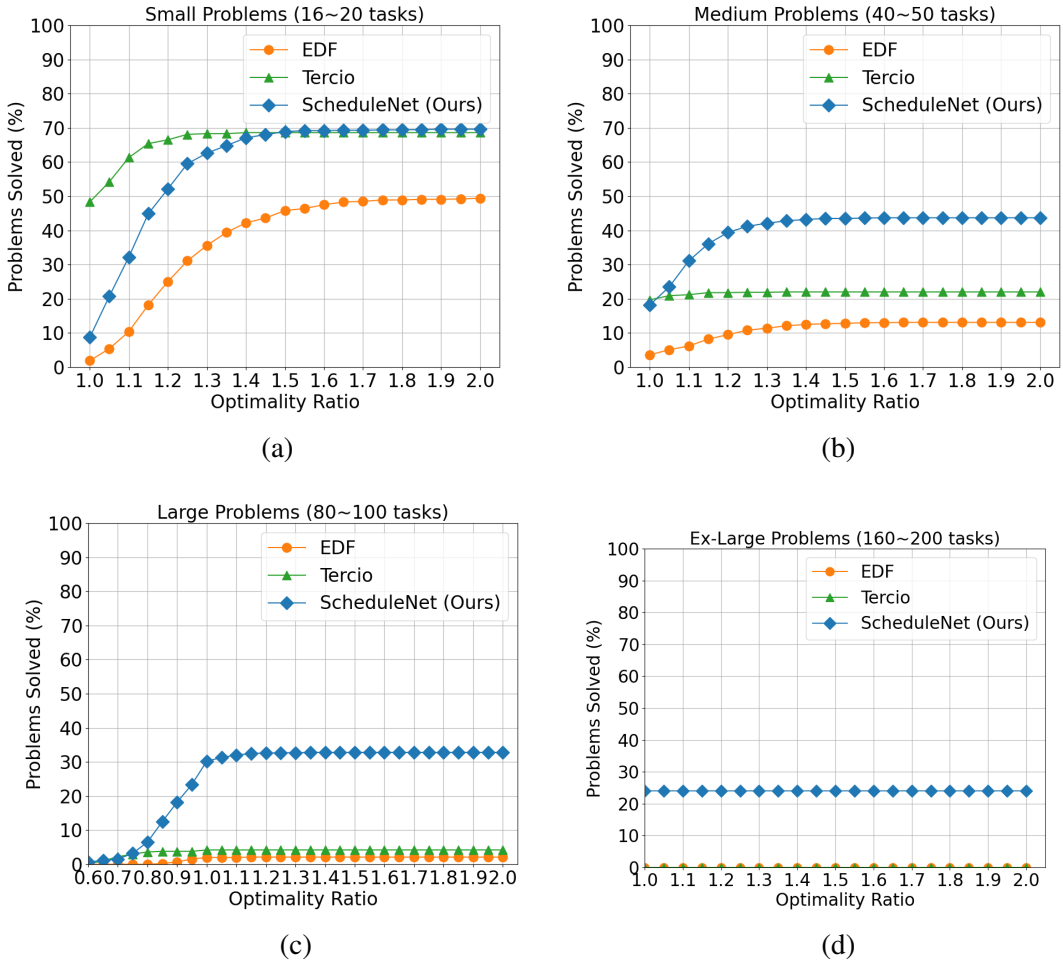
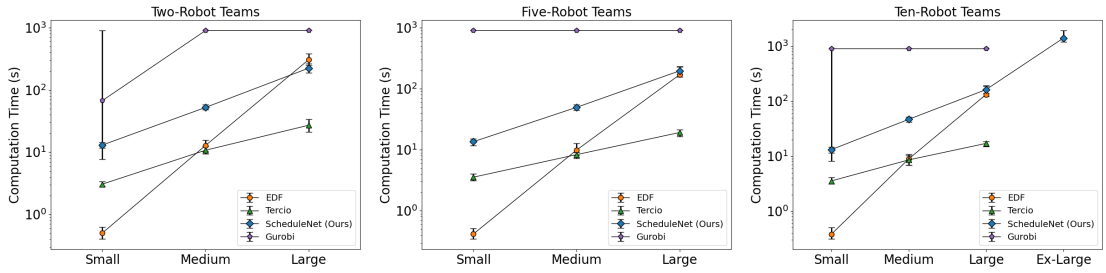


Figure 4.11: Evaluation results on problems of ten-robot teams of heterogeneous robots: (a) Small problems; (b) Medium problems; (c) Large problems; (d) Ex-Large problems

in medium problems of two-robot and five-robot teams in which ScheduleNet found feasible solutions for more than half of the problems whereas Tercio and EDF found less than 15% for medium-to-high optimality ratios ($r \geq 1.2$). Compared to evaluation results for homogeneous robots, high-quality schedules for heterogeneous robots were much harder to find for all three methods with a much lower success rate overall in finding a feasible schedule.

For large and extra-large problems, we extended the optimality ratios (measured relative to the solution returned by Gurobi) to the smallest value under which ScheduleNet solved at least one problem. In addition to finding schedules that were of higher quality

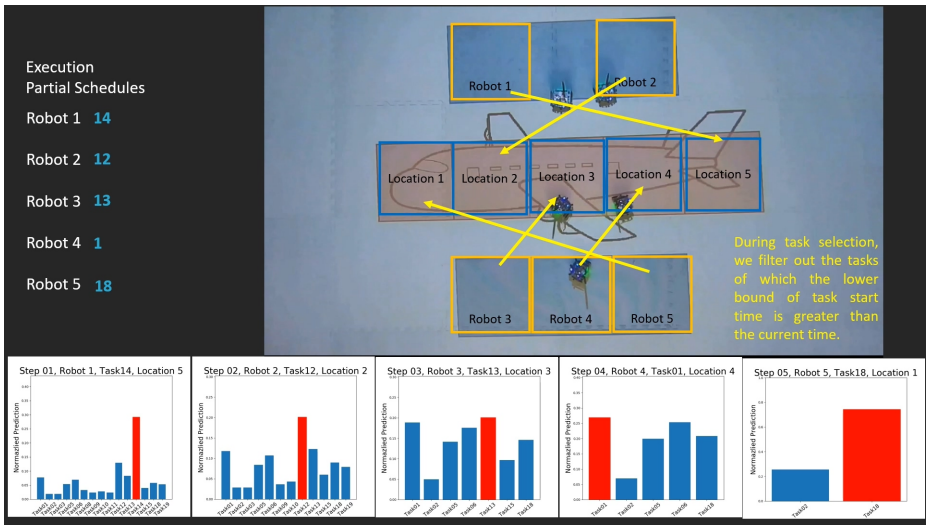


(a) (b) (c)

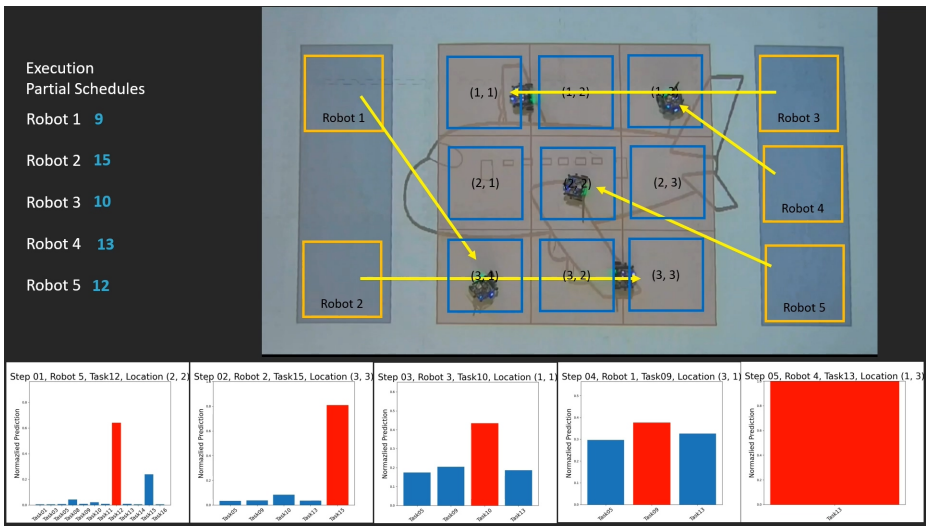
Figure 4.12: Running time statistics on different problems of heterogeneous robots: (a) Two-robot teams; (b) Five-robot teams; (c) Ten-robot teams. Error bars denote the 25th and 75th percentile

than those found by Gurobi for $\sim$ 10% of problems, ScheduleNet models significantly outperformed EDF and Tercio by finding feasible solutions for $>30\%$ of large problems compared to less than 5% by the latter baselines. **Notably, for extra-large problems, EDF, Tercio and Gurobi all failed to find any feasible solutions, whereas ScheduleNet was able to find schedules for up to 24% of the problems.** These results further demonstrate ScheduleNet's capability of generalizing learned knowledge to solving larger unseen problems.

Running time statistics of different methods were shown in Figure 4.12, where we counted only feasible solutions found by each method. Because these problems of coordinating heterogeneous robot teams were much harder than the homogeneous robot case (Figure 4.7), computation times in heterogeneous case (Figure 4.12) increased for all three methods. Furthermore, Gurobi timed out significantly more frequently. For ScheduleNet, similar time change patterns with respect to increasing problem scales can be observed as the homogeneous robot case. Nonetheless, computation times of ScheduleNet are shorter than those of Gurobi and show a much better balance between solution quality and solving speed than EDF and Tercio, making ScheduleNet much more viable in practice.



(a)



(b)

Figure 4.13: Demonstration of a 5-robot team completing tasks for airplane fuselage assembly. The ScheduleNet outputs for each step are plotted at the bottom, with the selected task assignment highlighted in red. (a) homogeneous robots with 1D task locations; (b) heterogeneous robots with 2D task locations

4.6 Robot Demonstration

We demonstrate our trained ScheduleNet model to coordinate the work of a five-robot team in a simulated environment for airplane fuselage construction, covering both homogeneous robot case in 1D space and heterogeneous robot case in 2D space. Our demo leverages the Robotarium, a remotely accessible swarm robotics research testbed with GRITSBot X robots [136]. Examples of scheduling homogeneous robot teams and heterogeneous robot teams are shown in Figure 4.13a and Figure 4.13b, respectively. The ScheduleNet outputs for each step are depicted in bar plots at the bottom of each figure, with the selected task assignment highlighted in red. A detailed breakdown of the scheduling process with those examples can be found in the supplementary video.

4.7 ScheduleNet Discussion

Our empirical results and 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 demonstrate that our approach:

1. Outperforms prior work in multi-robot scheduling both in terms of schedule optimality and the total number of feasible schedules found (Figure 4.4-Figure 4.6, Figure 4.9-Figure 4.11).

2. Achieves this superior performance in a scalable framework that allows us to train via imitation-based Q-learning on smaller problems to provide high-quality schedules on much larger problems (Figure 4.6d, Figure 4.11d).

3. Autonomously learns scheduling policies on multiple application domains (Figure 4.5 vs. Figure 4.8, attaining an order of magnitude speedup vs. an exact method (Figure 4.7, Figure 4.12).

4. Leverages a highly flexible framework that models homogeneous robots and heterogeneous robots via graph augmentation. Given the high expressiveness of heterogeneous graphs, our research opens up future opportunities in designing graph-based learning algorithms in multi-robot research.

We also note our algorithm's limitations. First, high-quality expert data are required to train ScheduleNet as the loss function assumes the experts choose the optimal scheduling action at each time step. Therefore, sub-optimal expert data would likely lead to degrade model performance. In this chapter, we assume it is practical to obtain high-quality solutions for small-scale problems where the problem complexity allows for exact algorithms to be used. Nonetheless, we propose in future work to investigate computational formulations of ScheduleNet that can explicitly reason about sup-optimality in expert scheduling demonstrations. Second, although ScheduleNet scales to different problem and team sizes, we observe a performance drop when the problem scale increases. Considering that our approach only trains on small scale problems, applying the learned representation to large scale problems, we propose further exploration into fine-tuning and transfer learning methods for improving the performance of ScheduleNet as problem sizes increases. Another limitation is that ScheduleNet assumes task durations to be deterministic and known a priori. Task performance in the real world is subject to many sources of uncertainty or randomness [86], such as machine breakdowns, unexpected releases of high priority jobs, or uncertainty in the processing times. Although the speedup of ScheduleNet vs. exact methods allows us to dynamically re-schedule in a timely manner in response to unexpected disturbance during execution, we propose to extend our approach to reason about stochasticity in task completion times to ensure robust schedule execution.

4.8 Summary

We presented a novel heterogeneous graph attention network model, called ScheduleNet, to learn a scalable policy for multi-robot task allocation and scheduling problems. By intro-

ducing 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, and generalizes well to large, unseen problems. Empirically, we showed that our method outperformed existing state-of-the-art methods in a variety of testing scenarios involving both homogeneous robot teams and heterogeneous robot teams.

CHAPTER 5
RECURRENT SCHEDULE PROPAGATION FOR COORDINATING STOCHASTIC HUMAN-ROBOT TEAMS

5.1 Introduction

In this chapter, we focus on the problem of multi-agent task allocation and scheduling [9] with mixed human-robot teams over multiple iterations of the same coordination problem. Our work accounts for and leverages stochastic, time-varying human task performance to quickly solve task allocation problems among team members 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) and spatial constraints (i.e., safety distance constraints).

Compared to task scheduling within multi-robot systems, the inclusion of human workers makes scheduling even more challenging because, while robots can be programmed to carry out certain tasks at a fixed rate, human workers typically have latent, dynamic, and task-specific proficiencies. Effective collaboration in human-robot teams requires utilizing the distinct abilities of each team member to achieve safe, effective, and fluent execution. For these problems, we must consider the ability of humans to learn and improve in task performance over time. To exploit this property, a scheduling algorithm must reason about a human's latent performance characteristics in order to decide whether to assign the best worker to a task now versus giving more task experience to a person who is slower but has a greater potential for fluency at that particular task. However, it is non-trivial to infer human strengths and weaknesses while ensuring that the team satisfies requisite scheduling constraints, due to the uncertainty introduced by variability in task execution behavior across different individuals, as well as uncertainty on future task performance affected by

human's learning effects with practice [12]. Moreover, a lack of consideration for human preferences and perceived equality may, in the long run, put efficient behavior and fluent coordination at a contradiction [146].

Recent advances in scheduling methods for human-robot teams have shown a significant improvement in the ability to dynamically coordinate large-scale teams in final assembly manufacturing [34, 22]. Prior approaches typically rely on an assumption of deterministic or static worker-task proficiencies to formulate the scheduling problem as a mixed-integer linear program (MILP), which is generally NP-hard [35]. Exact methods are hard to scale and often fail to consider the time-varying stochastic task proficiencies of human workers over multi-round schedule execution that could result in significant productivity gains. The heuristic approaches may be able to determine task assignments; however, such approaches require domain specific knowledge that takes years to gain. We desire a scalable algorithmic approach that can automatically learn to factor in human behavior for fast and fluent human-robot teaming.

In Chapter 2 and Chapter 3, we show that graph neural networks can be combined with imitation learning to efficiently solve multi-agent task allocation scheduling. However, both RoboGNN and ScheduleNet requires deterministic environments with known agent performance, making them not suitable for stochastic human-robot teams.

In this chapter, we propose a deep learning-based framework, called HybridNet, for scheduling stochastic human-robot teams under temporospatial constraints. Figure 5.1 shows the overall framework operating for multi-round scheduling. HybridNet utilizes a heterogeneous graph-based encoder and a recurrent schedule propagator. The encoder extracts high-level embeddings of the scheduling problem using a heterogeneous graph representation extended from the simple temporal network (STN) [33]. By formulating task scheduling as a sequential decision-making process, the recurrent propagator uses Long Short Term Memory (LSTM) cells to carry out fast schedule generation. The resulting policy network provides a computationally lightweight yet highly expressive model that is

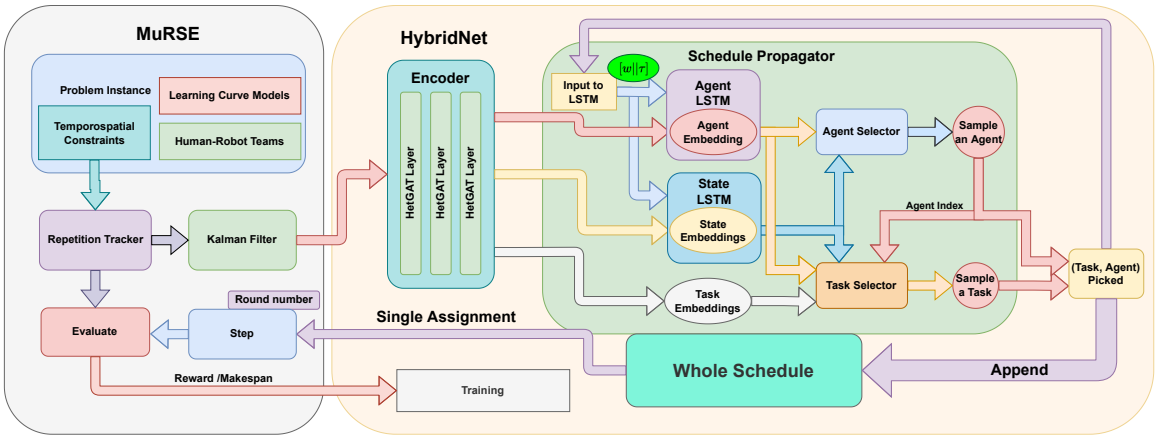


Figure 5.1: Overview of Multi-Round Scheduling Environment with HybridNet Scheduler. Left: MuRSE is developed to simulate a human-robot scheduling problem over multiple iterative rounds of execution, accounting for changes in human task performance. Right: HybridNet consists of a heterogeneous graph-based encoder to extract high-level embeddings of the problem and a recurrent schedule propagator for fast schedule generation.

end-to-end trainable via reinforcement learning algorithms.

The primary contributions of our work are:

- We propose a deep learning-based framework, HybridNet, for human-robot coordination under temporospatial constraints. HybridNet consist of a Heterogeneous Graph-based encoder and a Recurrent Schedule Propagator. The encoder extracts essential embeddings about the initial environment, while the propagator generates the consequential models of each task-agent assignments based on the initial embeddings. Inspired by the sensory encoding and recurrent processing of the brain, this approach allows for fast schedule generation, removing the need to interact with the environment between every task-agent pair selection.

- We develop a virtual Multi-Round Scheduling Environment (MuRSE) for mixed human-robot teams, capable of modeling the stochastic learning behaviors of human workers. MuRSE is OpenAI gym-compatible and we expect it to serve as a testbed to facilitate the development of human-robot scheduling algorithms.

- We present a novel policy learning framework that jointly learns how to pick agents and tasks without interacting with the environment between intermediate scheduling decisions and only needs a single reward at the end of schedule. By factoring in the action space into an agent selector and a task selector, we enable conditional policy learning with HybridNet. We account for the state and agent models when selecting the agents, and combine the information regarding the tasks, selected agent and the state for task assignment. As a result, HybridNet is end-to-end trainable via Policy Gradients algorithms.

- We conducted extensive experiments to benchmark the performance of HybridNet across various problem configurations, plus detailed ablation studies. Results showed HybridNet consistently outperformed prior human-robot scheduling solutions under both deterministic and stochastic settings.

5.2 Human-Robot Team Scheduling Problem

5.2.1 Problem Statement

We consider the problem of coordinating a mixed human-robot team in the same space, with temporal constraints related to probabilistic task completion times, wait and deadline constraints as well as spatial constraints. The problem extends subsection 4.2.1 by including stochastic human workers. We cast our problem in a multi-round setting and consider iterative schedule generation followed by schedule execution for a fixed number of rounds.

We describe the components of the problem using an eight-tuple $<n_r, \boldsymbol{a}, \boldsymbol{\tau}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{rep}, \boldsymbol{Loc}, z>$. n_r is the round number. $\boldsymbol{a}$ consists of all the agents available, with the first N_r agents denoting robot workers and the next N_h denoting human workers. $\boldsymbol{\tau}$ 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 $dur_{i,a}$ for agent a to complete. We introduce s_0 as the time origin and f_0 as the time point when all tasks are completed. $\boldsymbol{d}$ contains the deadline constraints.

$d_i \in \boldsymbol{d}$ specifies before which task τ_i has to be completed, i.e., $f_i \leq d_i$. $\boldsymbol{w}$ is the set of wait constraints. $w_{i,j} \in \boldsymbol{w}$ specifies the wait time between task τ_i and task τ_j. $\boldsymbol{rep}$ stores the repetition counter of every task-human worker pair. $rep^r_{i,j} \in \boldsymbol{rep}$ is the number of repetitions that human worker a_j has completed for task τ_i in the past $n_r - 1$ rounds. $\boldsymbol{Loc}$ is the list of all task locations. Finally, z is an objective function to minimize that can take different forms depending on end-user applications.

A solution in each round 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. As z varies depending on application-specific goals, we report the results of minimizing the makespan (i.e., overall process duration, $z = \max_i(f_i - f_0)$) as a generic objective function. For multi-round scheduling, we consider either the makespan in the final round or a weighted sum of makespan from all rounds.

In a multi-round scheduling environment, while robot task completion times are fixed across different rounds, human task completion times are stochastic and not known beforehand, which prohibits us from passing the math formulation of the problem to a MILP solver to search for optimal solutions. Also, human workers learn over time based on their internal learning curve models [147], as shown in prior work for assembly tasks [12].

5.2.2 Multi-Round Scheduling Environment

The Multi-Round Scheduling Environment (MuRSE) is developed to simulate a human-robot scheduling problem over multiple iterative rounds of execution, accounting for changes in the task performance of human workers based on previous round. Each round is a step in our OpenAI Gym-based environment implementation, which takes as input the complete schedules of team members, and then simulates the outcome of schedule execution.

Each round's execution is considered finished when all the tasks are assigned to one of the agents or if the provided schedule is determined to be infeasible under the problem constraints. The environment checks the feasibility of the provided schedule given the

constraints of the problem. If the schedule is feasible, the environment computes the total duration of task completion. If the schedule does not satisfy the constraints, it is determined to be infeasible and the list of tasks that could not been scheduled are returned.

We formulate scheduling in MuRSE as a Partially Observable Markov Decision Process (POMDP) using seven-tuple $\langle S, A, T, R, \Omega, O, \gamma \rangle$ shown below:

- States: The problem state S in MuRSE is a joint state consisting of all the agents' states plus the problem constraints.
- Actions: Actions at round t within MuRSE refers to a complete set of task allocations made up of a list of task-agent pairs, denoted as $A_t = [\langle \tau_{i_1}, a_{j_1} \rangle, \langle \tau_{i_2}, a_{j_2} \rangle, ...]$, to be executed in order.
- Transitions: T corresponds to executing the action in MuRSE and proceed to next round.
- Rewards: R_t is based on the scheduling objective a user wants to optimize. In subsection 5.2.5 we show how to compute R_t when optimizing makespan.
- Observations: Ω is the estimated performance of all the task-agent pairs, plus the observable part of the problem state.
- Observation Function: O is handled by the *Learning Curve Estimator* explained in the subsection 5.2.4.
- Discount factor, γ.

5.2.3 Agent Modeling

MuRSE stores the latent information of the whole team and models each agent as either a robot or a human worker to keep track of each agent's proficiency. The update of the environment happens at the end of each round, allowing agents to modify themselves based on their internal models based on the (task-agent) pairs executed this round.

Deterministic Robot Model

We generate the robot task completion times randomly through uniform distribution. Such completion times are kept fixed across different rounds for each robot.

Stochastic Human Model

In our system, we leverage the findings of [12] to account for humans learning over time, both in problem generation as part of the environment and a learning curve predictor as part of the scheduling policy. The human learning curve follows an exponential function of generic form over the course of multiple iterations as shown below:

$$y = c + ke^{-\beta i} \tag{5.1}$$

where i is the number of iteration the human has previously executed a task and c, k, β parameters.

We generate the human task completion times randomly based on Equation 5.1, and set the environment to provide Deterministic and Stochastic performance for human learning. The task duration parameters of the human learning model, c, k, β are built from the randomly sampled initial task completion time for the first round. For stochastic performance, the standard deviations are sampled from a Normal distribution following the method described in [12].

5.2.4 Learning Curve Estimator

The scheduler is given an estimate of the performance of the human agents for each task based on the information about the task duration of the previous executions of the task-agent pair through the *Learning Curve Estimator* as part of our MuRSE implementation. We implement a black box model based on the insights from [12] to simulate a Stochastic Human Learning Estimator. As an Agent completes a task in multiple rounds, its model

records the actual task completion duration, allowing *Learning Curve Estimator* to predict the next task-agent duration more accurately.

We design our learning curve model update as an adaptive Kalman filter, where our state vector is composed of learning curve parameters and our observation consists of observed task duration over multiple rounds. We model each individual's learning curve parameters as a hidden but static state, with the best initial guess as the population average. The model is further refined using the knowledge of the past experiences after the completion of each round in MuRSE. The task performance of each task-agent assignment is recorded from the actual agent performance to increase knowledge of the learning curve of the human agents. We update the Kalman Filter using the observed human task durations after every round following the Algorithm 2 from [12].

5.2.5 Reward Design

The total reward, R_t, at round t for the schedule generated by MuRSE is calculated based on the feasible, A', and infeasible, $\tilde{A}'$, subsets of task allocations, such that $A_t = A'_t \cup \tilde{A}'_t$. Specifically, the reward, R_t, is a combination of the expected reward for the feasible subset of task-agent assignments, $R_t(A'_t)$, and the reward from the assignment of the infeasible subset of task-agent assignments, $R_t\tilde{A}'_t$. The infeasible subset reward is computed by assigning every unfinished task to the agent that will complete it in the longest possible duration, multiplied by an infeasibility penalty coefficient, C_i, as in Equation 5.2.

$$R_t = \sum_{i \in A'_t} R\left(\tau_i, a_i\right) + C_i \text{max}_{a_j} \left(\sum_{i \in \tilde{A}'} R\left(\tau_i, a_j\right) \right) \tag{5.2}$$

The total reward, R_t, favors schedules with more feasible task allocations and enables learning from infeasible explorations during training.

5.3 HybridNet Scheduling Policy

As shown in Figure 5.1, our HybridNet framework consists of a heterogeneous graph-based encoder to learn high level embeddings of the scheduling problem, and a recurrent schedule propagator to generate the team schedule sequentially. This hybrid network architecture enables directly learning useful features from the problem structure, owing to the expressiveness of heterogeneous graph neural networks, and at the same time efficiently constructing the schedule with our LSTM-based propagator. As a result, HybridNet does not require interaction with the environment between every task-agent pair selection, which is necessary but computationally expensive in prior work [148, 149].

We denote the policy learned by HybridNet as $\pi_\theta(A|S)$, with θ representing the parameters of the neural network. At round t, an action takes the form of an ordered sequence of scheduling decisions, $A_t = \{d_1, d_2, ..., d_n\}, d_i = \langle \tau_i, a_j \rangle$, where a latter decision, d_i, is conditioned on its former ones, $d_{1:i-1}$. Then, the policy can be factorized as

$$p_\theta(A_t|S_t) = \prod_{i=1}^{n} p_\theta(d_i|S_t, d_{1:i-1}). \tag{5.3}$$

Using the Recurrent Schedule Propagator, HybridNet recursively computes the conditional probability, $p_\theta(d_i|S_t, d_{1:i-1})$, for sampling a task-agent pair. At the end, the network collects all the decisions and sends to the environment for execution.

5.4 Heterogeneous Graph Encoder

We cast the task scheduling problem for human-robot teams into a heterogeneous graph structure and adapt the heterogeneous graph attention (HetGAT) layer proposed in [149]. Although the original HetGAT layer has been shown effective in representation learning of multi-robot scheduling problems, it requires a deterministic setting and thus is not suitable for reasoning with stochastic human workers. Here, we make several key innovations that allow HetGAT to process stochastic human-robot teams.

5.4.1 HetGAT Layer for Stochastic Human-Robot Teams

At the start of each round for a given human-robot scheduling problem with observation o_t, we construct the STN from constraints following [149]. In a STN, 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. For stochastic human workers, the estimated task complete times from the learning curve estimator are used.

Next, we build the heterogeneous graph representation by extending from the STN to include agent nodes, location nodes and a state summary node. In a heterogeneous graph, we use a three-tuple, $<srcName, edgeName, dstName>$, to specify the edge type/relation that connects the two node types (from source node to destination node). The metagraph summarizing all the node types and edge types of the heterogeneous graph is shown in Figure 5.2.

Then, a HetGAT layer computes the output node features by performing per-edge-type message passing followed by per-node-type feature reduction, while utilizing a feature-dependent and structure-free attention mechanism.

Improvement #1

The first innovation we make lies in the STN simplification process. Here, we develop a more efficient method than the one used in [149]. For every finish node, f_i ($i > 0$), we re-route all its incoming edges to its correspond start node, s_i. That is, we replace edge $(src \rightarrow f_i)$ with a new edge $(src \rightarrow s_i)$ by combing the corresponding edge weights. Then we remove all finish time nodes (except f_0) from the STN to obtain a simplified version that reserves most of the information of the original STN while using only half the nodes. In this way, each task can be represented by its start time node. We denote the edge type from STNs using $<task, temporal, task>$as they encode the temporal constraints. To facilitate message passing among task nodes, for each edge present in the simplified STN,

we add a reverse edge with the same edge weight, denoted as edge type $<task, aug, task>$. Compared to the method used in [149], our method does not require finding the minimum distance graph of the STN, which are not robust against stochastic task execution. The re-routing also reduces the graph structure from a dense STN to a sparse one that are more computationally efficient.

Improvement #2

In [149], the same edge features are passed as input to all HetGAT layers, making the edge features used "shallow". In HybridNet, we also learn "deep" embeddings for edge features. This is achieved by assigning learnable weights to edge attributes and outputting the results to be used for next HetGAT layer. Also, the edge attributes of relations $<task, assignedTo, agent>$,$<task, takeTime, agent>$, and $<agent, useTime, task>$ now use the estimated duration and standard deviation of the corresponding (task, agent) pair instead of a single value.

5.4.2 Encoder Network

We directly build our encoder on the graph by stacking several HetGAT layers sequentially to obtain a fully graph convolutional structure that is nonparametric on problems sizes and end-to-end trainable. The encoder utilizes multi-layer structure to extract high-level embeddings of each node that will be send to the recurrent propagator for fast schedule generation.

5.5 Recurrent Schedule Propagator

Prior work in GNN-based schedulers [148, 149] uses interactive scheduling to generate the whole solution, which requires regenerating the graph representation and recomputing the relevant embeddings between each task-agent assignment, which could be ineffective when problem scales up. By utilizing an LSTM-based Recurrent Predictor, we propagate forward

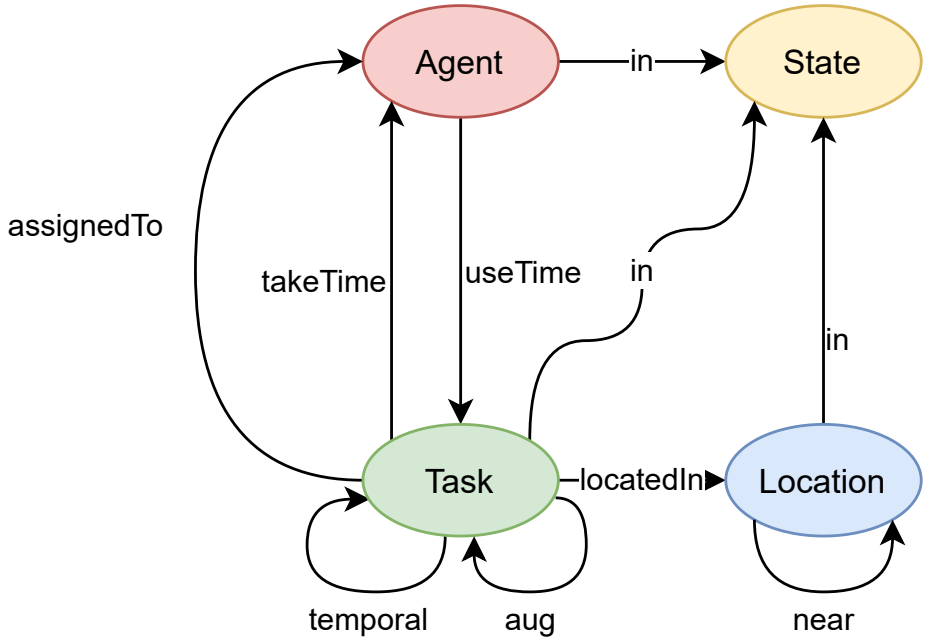


Figure 5.2: Metagraph of the heterogeneous graph built from the STN by adding agent and state summary nodes.

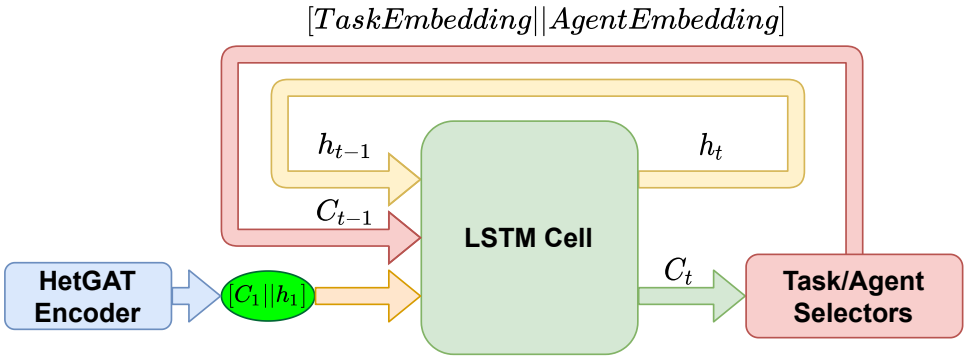


Figure 5.3: LSTM based Schedule Propagator Model taking initial input from the Encoder or the picked Task-Agent Assignment for Agent and State Encoding.

consequences of each assignment, recreating the encoded information without querying the environment, leading to significantly reduction in computational complexity compared to pure GNN-based methods.

As $d_i = \langle \tau_i, a_j \rangle$, one straightforward learning approach is to learn a policy that directly outputs the task-agent pair, i.e., $\pi_{joint} = p(\tau_i|\cdot) \cdot p(a_j|\cdot)$. However, this approach fails to capture the underlying composite and conditional nature of the scheduling decisions, where the task to schedule is strongly dependent on the picked agent. Also, jointly outputing task and agent requires evaluating all possible task agent pair combiniations, which is inefficient when problem size scales up. Instead, we explicitly factor the action space into an agent selector and a task selector and aim to learn a conditional policy. That is, $\pi_{factor} = \pi_{agent}(a_j|\cdot) \cdot \pi_{task}(\tau_i|a_j, \cdot)$. This factorization allows the policy to reason about the conditional dependence of tasks on heterogeneous agents.

The pseudo-code for scheduling generation with HybridNet is presented in algorithm 3. The Recurrent Schedule Propagator takes as input the Task, State and Agent embeddings generated by the Heterogeneous Graph Encoder and sequentially generates task-agent pairs based on the encoded information. To predict the consecutive encoding of state and agents, we use LSTM cells to recursively generate the Agent and State embeddings after each agent-task assignment, as shown in line 14-15, where $||$ denotes concatenation operation. The use of LSTM removes the requirement of interacting with the Environment during intermediate scheduling decisions, and directly outputs the whole schedule at the end.

The key component of the Schedule Propagator is the use of LSTM. By leveraging the LSTM cells, we remove the need to use GNN models for task allocation for all but the first task-agent assignment. After each task-agent pair selection, the state and agent embeddings are updated using the state LSTM and agent LSTM, respectively. The LSTM Cell stores the hidden and cell data from the previous step of the task allocation and predicts the next step based on the input using Equation 5.4 [125].

Algorithm 3: Schedule Generation using HybridNet

Input: graph g, features f, unscheduled tasks u
Output: *schedule*

schedule $= [\,], t = 0$
$(h_a^t, c_a^t, h_\tau^t, c_\tau^t, h_s^t, c_s^t) \leftarrow Encoder(g, f)$
while $|u| \neq 0$ **do**
  $p_a^t \leftarrow$ *AgentSelector*(h_s^t, h_a^t)
  $a_j \leftarrow$ *Sampling*(p_a^t)
  $p_\tau^t \leftarrow$ *TaskSelector*$(h_\tau^t, h_s^t, h_{a_j}^t)$
  $\tau_i \leftarrow$ *Sampling*(p_τ^t)
  schedule.append$(\langle \tau_i, a_j \rangle)$
  unscheduledTasks.remove(τ_i)
  if $|$*unscheduledTasks*$| == 0$ **then**
    return *schedule*
  end
  $t \leftarrow t + 1$
  $h_s^t, c_s^t \leftarrow LSTM_s((h_\tau^{t-1}[\tau_i] || h_a^{t-1}[a_j]), h_s^{t-1}, c_s^{t-1})$
  $h_a^t, c_a^t \leftarrow LSTM_a((h_\tau^{t-1}[\tau_i] || h_a^{t-1}[a_j]), h_a^{t-1}, c_a^{t-1})$
end

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_t, x_t] + b_i) \\
\tilde{c}_t &= \textit{tanh}(W_c[h_{t-1}, x_t] + b_c) \\
c_t &= f_t c_{t-1} + i_t \tilde{c}_t \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o \\
h_t &= o_t \textit{tanh}(c_t)
\end{aligned}
\tag{5.4}
$$

The Encoder output is in the form of $[h_1, c_1]$ where the initial hidden state, h_1, and initial cell state, c_1, are used as the initial inputs for Equation 5.4. Equation 5.4 calculates the next hidden state, h_t, and cell state, c_t, using the internal values f_t, i_t and $\tilde{c}_t$. f_t is the forget gate output generated from the previous hidden state h_{t-1} and input x_t, using the forget gate weights W_f. i_t is the input gate output computed using h_{t-1} and x_t, along with the input

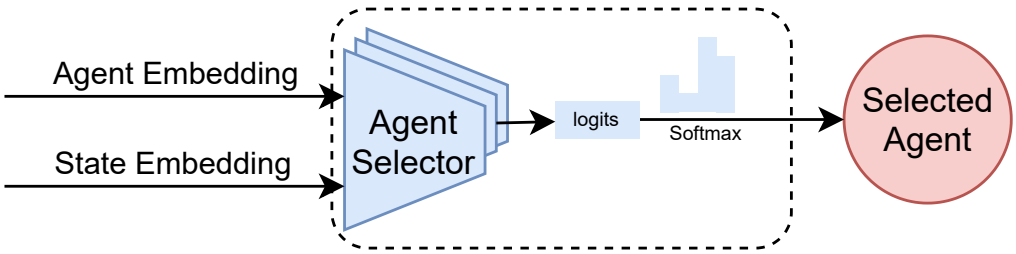


Figure 5.4: Agent Selector Model using Softmax based Sampling.

gate weights W_i. The Cell State, c_t, is based on the previous cell state, c_{t-1}, and the outputs of the input gate and calculated using the cell state gate weights W_c. The learnable weights W_f, W_i and W_c are trained using the training method described in section 5.6 along with the Heterogeneous Graph Encoder.

5.5.1 Agent Selector

The Agent Selector selects the new agent for the next decision d based on the state and agent information. Specifically, the concatenated state-agent embeddings are processed by a feed-forward neural network, f_a, to compute the likelihood of selecting each agent for the next task-agent pair, using Equation 5.5. A softmax operation is performed to convert the raw predictions into a probability distribution. After the selection of the agent, the agent embedding of the chosen agent is updated based on the selected task and state embeddings, as state change only happens for the assigned agent. This approach allows the agent selector to consider how busy each agent is, based on the inherent information presented in the embeddings.

$$\pi_{agent}(a_j|s) = softmax_i(f_a([h_{a_j}||h_s])) \tag{5.5}$$

5.5.2 Task Selector

Next, the Schedule Propagator uses the Task Selector to assign the task for the selected agent based on the state, agent and unscheduled task embeddings. As shown in Equa-

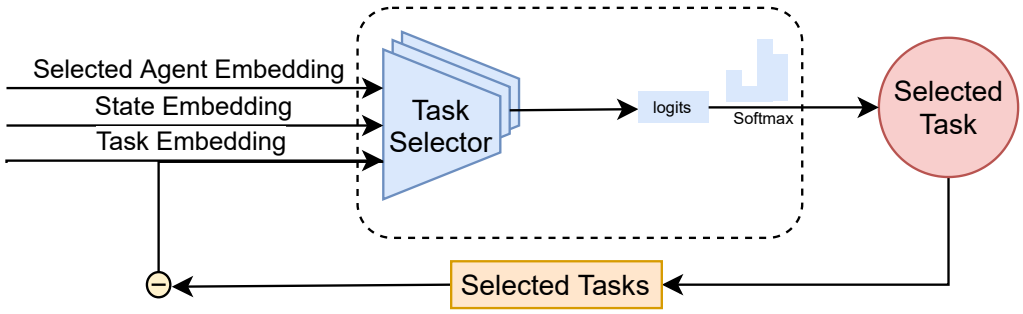


Figure 5.5: Task Selector Model using Softmax Sampling after filtering out the Previously Assigned Tasks.

tion 5.6, the Task Selector concatenates the state, selected agent and the unscheduled task embeddings and passes the combined information to a feedforward neural network, f_τ, to calculate the likelihood of the task being assigned to the selected agent. After assigning a task to an agent for execution, it is removed from the list of unscheduled tasks for this round. Since the calculation of likelihood of each task is independent of each other up to the last softmax operation, the model is scalable and can be used for different problem sizes.

$$\pi_{task}(\tau_i|a_j, s) = \mathit{softmax}_i(f_\tau([h_{\tau_i}||h_{a_j}||h_s])) \tag{5.6}$$

5.5.3 Ensemble-Based Schedule Boosting

We boost the performance of HybridNet by ultilizing multiple trained models to generate schedules for each round of the problem. These schedules are evaluated in the Estimated Environment using the task-completion times provided by the Learning Curve Estimator. The best schedule in the Esimated Environment is selected as the output of HybridNet and then sent to MuRSE for execution using actual task completion times for human agents. The Learning Curve Estimator is updated based on the chosen optimal schedule and the observed actual task-completion times for each round. This ensures that the scheduler interacts with the real world only once per round.

Leveraging Sample Space

During testing, we utilize an ensemble sampling strategy to further the performance gain. Specifically, we generate multiple schedules for the same task allocation problem every round. We select the best performing schedule by computing the estimated makespan utilizing the Learning Curve Estimator and provide it to the MuRSE instance. As the sampling of the task allocation is done after the generation of the output from the Heterogeneous Graph Encoder, the initial encoding for the problem is shared across all sampled schedules in the ensemble. We use the shared initial encoding and the Recurrent Schedule Propagator to generate the sequential task-agent allocations, creating a set of schedules. More sampling improves solution quality at increased computation that scales with the complexity of the Schedule Propagator.

Leveraging Multiple Policies

We utilize an ensemble of different models from the training to improve the overall performance. After the completion of the training, we validate the trained models to select top K performing policies. As different policies have different advantages, generating schedules using multiple policies provides a more diverse set of solutions to select from. We select the top K performing policies through a validation step and evaluate the models on a separate data set to the training set. Similar to 1), we leverage the simulated environment using the Learning Curve Estimator to select the best schedule from the solution sets. Diversity of the policies improves solution quality at the cost of increased computation.

Schedule from Previous Round

We also store the schedule used in the previous round in the MuRSE. This schedule is evaluated alongside the Ensemble-based schedules in the Estimated Environment using the task-proficiencies from the Learning Curve Estimator. This ensures that previous feasible schedules can be inherited between rounds to compare with newly generated schedules.

5.6 Learning Stochastic Scheduling Polices

HybridNet is end-to-end trainable in MuRSE using Policy Gradient methods that seek to directly optimize the model parameters based on rewards received from the environment [150]. We develop our policy learning framework from Proximal Policy Optimization (PPO) [94] and make several adaptations for better variance reduction in stochastic scheduling. In particular, we optimize the clipped surrogate objective shown in Equation 5.7, where $r_t(\theta)$ denote the probability ratio between current policy and old policy on collected rollout data, $r_t(\theta) = \frac{\pi_\theta(A|S)}{\pi_{\theta_{old}}(A|S)}$, Adv_t is the estimated advantage term, and ϵ is the clipping parameter.

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)Adv_t, \\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)Adv_t)] \tag{5.7}$$

In Equation 5.7, Adv_t, is estimated by subtracting a "baseline" from the total future reward (or "return"). PPO and Actor-Critic methods typically utilze a learned state-based value function as such baselinses. However, in a stochastic environment like MuRSE, due to the combinatorial nature of the task scheduling problem, plus the stochasticity in human proficiency, learning a helpful value function is non-trivial. Instead, we choose to use the Greedy Rollout Baseline (GRB) as a more accessible and efficient alternative. Specifically, GRB uses, $\pi_{greedy}(A|S)$, a deterministic greedy version of the HybridNet scheduler, to collect rewards in the environment. Its weights, θ_{greedy}, are updated periodically by copying the weights from the current learner, $\pi_\theta(A|S)$.

We present the pseudocode for HybridNet training in algorithm 4. Lines 5-14 details the process of rollout data collection on MuRSE instances generated on-the-fly. Line 6 ensures the same randomly-initialized environment instance is used by all episodes within one training epoch for variance reduction. In lines 15-22, GRB is used on the same MuRSE instance to collect baseline rewards. Lines 23-24 computes the advantage estimates with

Algorithm 4: HybridNet Training

Input: Number of training epochs K, Batch size N, learning rate η, number of gradient updates per epoch N_{iter}, total number of rounds T, greedy baseline update frequency K_{greedy}

Output: Trained policy π_θ

Initialize policy network parameters θ_0
$\theta_{greedy} \leftarrow \theta_0$
for $k = 1$ *to* K **do**
  Sample problem instance, MuRSE_k
  for $i = 1$ *to* N **do**
    Reset MuRSE_k to $t = 1$
    for $t = 1$ *to* T **do**
      Get observation o_t^i from MuRSE_k
      Build heterogeneous graph and input node features
      Sample $A_t^i = [\langle \tau_{i_1}, a_{j_1} \rangle, \langle \tau_{i_2}, a_{j_2} \rangle, ...]$ from π_θ
      Step through MuRSE_k and get reward r_t^i
      Store $\{(o_t^i, A_t^i, r_t^i)\}$ to trajectory buffer
    end
  end
  Reset MuRSE_k to $t = 1$
  for $t = 1$ *to* T **do**
    Get observation o_t^{greedy} from MuRSE_k
    Build heterogeneous graph and input node features
    $A_t^{greedy} = \arg\max_A p(A|o_t^{greedy})$, $p(A|\cdot) \sim \pi_{greedy}$
    Step through MuRSE_k and get reward r_t^{greedy}
    Store $\{(r_t^{greedy})\}$ to baseline buffer
  end
  Compute rewards-to-go: $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^i$, $R_t^{greedy} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^{greedy}$
  Compute advantage estimates: $A_t^i = R_t^i - R_t^{greedy}$
  for $j = 1$ *to* N_{iter} **do**
    Compute the clipped surrogate objective $L(\theta)$ using Equation 5.5-Equation 5.7
    Perform stochastic gradient ascent for $\nabla L(\theta)$ using Adam optimizer with learning rate η
  end
  Perform $\theta_{greedy} \leftarrow \theta$ every K_{greedy} epochs
end

GRB rewards. Lines 25-28 show the gradient update procedure by maximizing $L(\theta)$, which requires recomputing the action probability using updated policy network at each iteration. Line 29 updates the weights of GRB policy every certain training epochs.

5.7 Experimental Results

5.7.1 Data Generation

We generate scheduling problems with deadline, wait and spatial constraints under different scales to evaluate the performance of HybridNet. For all scales, the deadline constraints are randomly generated for approximately 25% of the tasks from a range of [1, $25N$] where N is the number of tasks. Wait constraints are generated such that 25% of Tasks have wait time constraints, and the duration of non-zero wait constraints is sampled from $U([1, 50])$. All agents' task durations are sampled and clamped to the range of [15, 100].

Small Scale

The small data set is uniformly sampled to have 9 to 11 tasks and 3 to 5 agents with at least 1 robots and 1 humans in a team. We generated 2000 training problems and 200 test problems.

Medium Scale

The medium data set is uniformly sampled to have 18 to 22 tasks and 3 to 5 agents with at least 1 robots and 1 humans in a team. We generated 200 test problems.

Large Scale

The large data set is uniformly sampled to have 36 to 44 tasks and 3 to 5 agents with at least 1 robots and 1 humans in a team. We generated 200 test Problems to evaluated the HybridNet performance on scalability.

We generate data sets for training, validating and testing our model against benchmarks:

- Training data set is generated to train the models in small data scale.
- Validation data Set are generated for the small data scale to validate and select the top $K = 3$ best performing policies for ensemble-based boosting.
- Test data set are generated for small, medium and large scales.

To simulate the stochastic learning of human agents, for each Data Set noise is introduced to the Human Agent models by simulating the natural distribution of the c, k, β parameters of Equation 5.1.

Agent performance of both humans and robots in the final round is uniformly sampled from a range of [15, 100]. For Human Performance, we achieve this by sampling the Human Mean Task Performance at starting round to have mean task durations within [30, 100] to account for human learning effects. The c and k values for the Human Performance are sampled from the mean task performance such that both value are at least 1. The β value is computed so that over 10 rounds the decrease in the task durations are within a range of 20% to 40%.

The standard deviations of c, k and β values for simulating stochastic human performance are generated using the following steps:

1. Sampling the combined standard deviation of c and k from a uniform distribution of [1, 28].
2. Sampling the standard deviation of c value based on the combined standard deviation such that both standard deviations of c and k are at least 1.
3. The standard deviation of β is calculated to ensure that 99.7% of sampled noise falls within the range of [0, 2β].

Deterministic Models are based on the mean values of k, c and β while the Stochastic Models sample from the generated standard deviations using Equation 5.1. Both Deterministic and Stochastic models are clipped to fall within the specified range of task durations.

5.7.2 Benchmark

We benchmark HybridNet 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 agent.

- *GeneticEDF*: An Evolutionary Optimization Algorithm that uses post-processing on the schedule generated by EDF [68]. The genetic algorithm creates new schedules based on the initial schedule through iterative randomized mutations by swapping task allocations and task orders [12]. Each generation selects the top performing schedules, sorted on feasibility and total schedule completion time, and used as the baseline for creating new mutations. The GeneticEDF was run for 10 generation with 90 baseline schedules, 10 task allocation and 10 task order swapping mutations.

Furthermore, we evaluate the functionality of the Recurrent Schedule Propagator by comparing it against the following HybridNet variant:

- *HetGAT*: We implement a HetGAT Scheduler based on the Graph Neural Network Encoder of HybridNet. After each task-agent pair assignment, instead of using the LSTM Cells to update the task, agent and state embeddings, HetGAT Policy directly interacts with the environment to model the consequences of action with a new heterogeneous graph and re-computes those information from it. We trained HetGAT using Policy Gradient with Greedy Rollout Baselines.

We evaluate HybridNet on three metrics: 1) Feasibility, that is, proportion of problems solved; 2) Adjusted makespan, which is computed using the average of a) the makespan of feasible schedules and b) the maximum possible makespan of the infeasible schedules; and 3) Runtime statistics. Runtime statistics for training and execution is compared for HybridNet and HetGAT Scheduler to investigate their computational complexity.

5.7.3 Model Details

We implement HybridNet and HetGAT using PyTorch [151] and Deep Graph Library [152]. The Kalman Filter's learning curve parameters are initialized based on 50 randomly generated human performance for each task during problem set construction. We use the same parameters for the Kalman filter updates as in [12]. The HybridNet Encoder used in training/testing is constructed by stacking three multi-head HetGAT layers (the first two use concatenation, and the last one uses averaging). The feature dimension of hidden layers = 64, and the number of heads = 8. The Recurrent Propagator utilizes a LSTMCell of size 32 followed by a fully-connected layer and a softmax layer. We set γ = 0.99, batch size = 8 and used Adam optimizer [153] with a learning rate of 2×10^{-3}, and a weight decay of 5×10^{-4}. We also added the entropy regularization in loss computation with a coefficient of 3×10^{-2}. We employed a learning rate decay of 0.5 every 4000 epochs. We evaluate the models using a batch size of 8 and 16. For MuRSE parameters, the infeasible reward coefficient $C_i = 2.0$ and total round number = 10. Both training and evaluation were conducted on a Quadro RTX 8000 GPU.

5.7.4 Evaluation with Deterministic Task Proficiency

Table 5.1 shows the evaluation performance with Deterministic Human Proficiency in different scales. The Deterministic Human Proficiency means that during training and evaluation, human learning curve is known and execution is deterministic for every agent based on the mean task completion times. In Table 5.1, "Small", "Medium" and "Large" denotes the data scale the method is tested. For ensemble-based schedule boosting, HybridNet samples 8 different schedules using the top 3 performing policies selected during the validation step. The results show that HybridNet outperforms both EDF and GeneticEDF in terms of adjusted makespan and feasibility percentage. HybridNet trained on Small scale problems is able to generalize to both Medium and Large scale problems, consistly outperforming other baselines. The performance of EDF and GeneticEDF dropped sharply from medium

Table 5.1: Evaluation Results: Adjusted Makespan and Feasibility with Deterministic Human Task Proficiency for the Final (10^{th}) Round

Methods	**Small**		**Medium**		**Large**	
	Total Makespan	**Feasibility (%)**	**Total Makespan**	**Feasibility (%)**	**Total Makespan**	**Feasibility (%)**
EDF	408.76 ± 0.00	61.50 ± 0.00	1022.70 ± 0.00	33.00 ± 0.00	2370.80 ± 0.00	12.00 ± 0.00
GeneticEDF	356.36 ± 9.68	72.55 ± 1.80	978.72 ± 9.27	38.20 ± 0.87	2385.10 ± 17.21	11.15 ± 0.67
HetGAT	572.81 ± 4.32	22.05 ± 1.23	1239.57 ± 4.57	4.00 ± 0.71	2518.65 ± 4.18	0.25 ± 0.25
HybridNet	$\mathbf{333.99 \pm 6.82}$	$\mathbf{82.05 \pm 1.72}$	$\mathbf{960.20 \pm 8.80}$	$\mathbf{43.35 \pm 1.43}$	$\mathbf{2289.66 \pm 8.01}$	$\mathbf{16.75 \pm 0.56}$

Table 5.2: Evaluation Results: Adjusted Makespan and Feasibility with Stochastic Human Task Proficiency for the Final (10^{th}) Round

Method	**Small**		**Medium**		**Large**	
	Total Makespan	**Feasibility (%)**	**Total Makespan**	**Feasibility (%)**	**Total Makespan**	**Feasibility (%)**
EDF	413.15 ± 6.72	62.25 ± 1.17	1059.40 ± 19.41	37.45 ± 1.52	2688.87 ± 28.62	14.25 ± 1.01
GeneticEDF	372.18 ± 7.81	69.35 ± 1.64	999.91 ± 7.82	31.85 ± 1.36	2218.91 ± 25.41	15.40 ± 1.50
HetGAT	581.84 ± 6.39	19.65 ± 2.34	1240.39 ± 5.46	4.15 ± 0.90	2521.87 ± 4.99	0.25 ± 0.34
HybridNet	$\mathbf{351.99 \pm 6.23}$	$\mathbf{78.40 \pm 1.97}$	$\mathbf{881.99 \pm 12.91}$	$\mathbf{52.05 \pm 1.56}$	$\mathbf{2211.54 \pm 13.60}$	$\mathbf{21.50 \pm 0.89}$

Table 5.3: Evaluation Results: Runtime (s) Performance on Single Problem

	Data Scale	EDF	GeneticEDF	HetGat	HybridNet
Training	**Small**	-	-	126.88 ± 63.76	**23.55 ± 5.59**
	Medium	-	-	394.11 ± 85.61	**33.84 ± 6.58**
Evaluation	**Small**	**0.88 ± 0.89**	4.22 ± 2.50	7.68 ± 5.66	**3.20 ± 0.41**
	Medium	**2.83 ± 3.50**	17.22 ± 14.06	14.32 ± 8.06	**6.28 ± 1.59**
	Large	**2.88 ± 11.03**	81.58 ± 96.67	42.88 ± 26.72	**15.33 ± 7.23**

to large problems.

HybridNet also outperforms HetGAT on all scales. This shows that HybridNet is capable of learning high performance policies by leveraging the Recurrent Schedule Propagator and effectively removes the interaction with the Environment that is required by HetGAT. This advantage makes HybridNet much less computationally expensive than a pure GNN-based framework. As seen in the lower performance of HetGAT model compared to other methods in Table 5.1 and Table 5.2, training GNN-only models like HetGAT using a single reward for each round, without interactive scheduilng to distribute the reward into each intermediate scheduilng decsion, leads to sub-optimal policies. The reason might be due to the difficulty of credit assignment and the vanishing gradients problem arising from long sequence assignments. Note that for pure-GNN based models, each task-agent assignment builds a new heterogeneous graph. Unlike complex and slow Graph Neural Network models to compute sequences of task allocations, LSTMs provide a faster method of predicting long sequences, making them more efficient for scalable task predictions. As a resuts, HybridNet only runs a single Graph Neural Network per schedule generated and ultlizes LSTM cells to address the vanishing gradient problem.

In Table 5.3, we provide the runtimes of training and evaluation for HetGAT and HybridNET. HybridNet is approximately 5 times faster in training compared to HetGAT Model in small scale and 10 times faster in the medium scale and at least 2 times faster during evaluation, under the same batch size. EDF and GeneticEDF were evaluated through the CPU without GPU acceleration, making it less appropriate to compare the runtime of deep learning models to the traditional methods. Here we only report their runtime in eval-

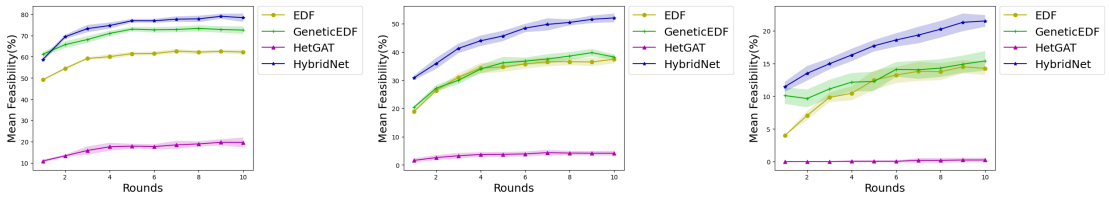
uation for completeness. When problem size increases, the computation time of HybridNet increases less than GeneticEDF. When the data scale increases by a factor of 4, GeneticEDF's solving duration increases by a factor of 20, while both HetGat and HybridNet scale by a factor of 5. The only advantage of EDF and GeneticEDF is that both methods do not require training.

5.7.5 Evaluation with Stochastic Task Proficiency

Table 5.2 shows the evaluation performance with Stochastic Human Proficiency in different problem scales. The Stochastic Human Proficiency is presented as uncertainty in the actual human performance within MuRSE, where the human task durations are sampled from Equation 5.1 with c, k and β values, plus initially generated standard deviation. The results show that HybridNet outperforms the EDF, GeneticEDF and HetGAT across different data scales. EDF outperforms GeneticEDF in Medium Scale for Stochastic Models as the GeneticEDF utilizes the Learning Curve Estimator to evaluate the best policy across multiple generations. The error between the Environment generated by the Learning Curve Estimator and the actual environment may lead to the selection of sub-optimal schedules for the GeneticEDF. Same as in the deterministic case, HybridNet trained on small problems is able to generalizes to larger problems with much less performance drop than EDF and GeneticEDF.

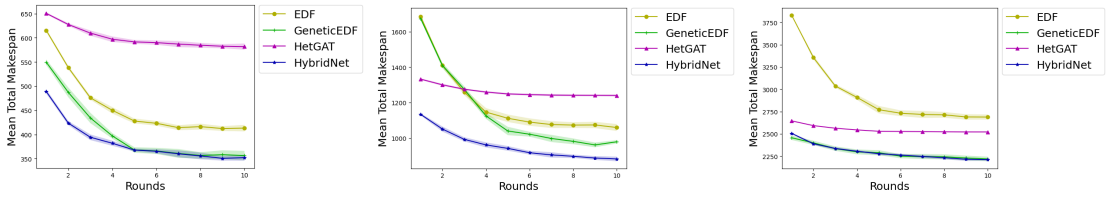
To make a closer comparison between different methods, in Figure 5.6 and Figure 5.7 we plot the performance curves evaluated at every round. It can be seen that the Human Performance Prediction made by the Kalman Filter helps the HybridNet scheduler to find better schedules for future rounds. We show that better scheduling policies also have a steeper learning curve, as more information for feasible task completion times are provided to the Kalman Filter. The increase in fidelity of the Learning Curve Estimator in turn improves the input of the policy model for the future rounds.

In Table 5.2 as well as in Figure 5.7a and Figure 5.7c, the GeneticEDF and HybridNet



(a) (b) (c)

Figure 5.6: Feasibility percentage results with stochastic human model over 10 rounds. The shaded regions represents 1 standard deviation of the mean values calculated over 10 repetitions of the evaluation. (a) Small-scale; (b) Medium-scale; (c) Large-scale.



(a) (b) (c)

Figure 5.7: Total makespan results with stochastic human model over 10 rounds. (a) Small-scale; (b) Medium-scale; (c) Large-scale.

have close adjusted makespans in the Small and Large problem scales, especially at later rounds, despite HybridNet having a higher feasibility percentage in both cases. This is attributed to the way GeneticEDF searches through schedules. As the GeneticEDF searches through schedules based on feasibility first and makespan second, it selects schedules with a lower adjusted makespan even if the generated schedule is not feasible. This approach results in GeneticEDF generating schedules with lower adjusted makespan even if the solution is not feasible. Moreover, HybridNet is trained to balance efficient schedules with feasible schedules during training leading to a slightly higher adjusted makespan compared to a model that specifically focuses on minimizing the makespan of feasible solutions. Since a similar Genetic extension to the EDF algorithm can also be applied to the results from the HybridNet model, the schedules produced by the HybridNet can be further optimized through a evolutionary approach similar to GeneticEDF, which we left as future work.

5.8 HybridNet Discussion

Our empirical results and analysis demonstrates that HybridNet establishes a state-of-the-art in autonomously learning policies for coordinating stochastic human-robot teams in a computationally efficient framework. In particular, we demonstrate that:

1. The Heterogeneous Graph Attention Model is able to leverage the relationships between individual units within the problem to generate more informed embeddings. The node feature updates can utilize different types of structural information efficiently to generate representations used by the selector model in Figure 5.4 and Figure 5.5 toward fast decision generation for the policy.

2. Our model is scalable in both data scale and sample size. This allows us to train HybridNet via policy optimization on small problems to provide high-quality schedules on much larger problems, as shown in Table 5.1 and Table 5.2.

3. Compared to pure GNN-based schedulers, the use of Recurrent Scheduler Propagator brings in much speedup. When leveraging sample space for schedule boosting, the Encoding generated by the Heterogeneous Graph Encoder is shared across multiple scheduler rollouts of the propagator. This allows for sequential task-allocation to be done without needing to rebuild the Graph Model after the initial construction in both training and testing. As only a single Graph Model is generated per training step, Proximal Policy Optimization only needs to store a single instance of the graph model to optimize the clipped surrogate objective in Equation 5.7.

We also note our algorithm's limitations. First, our environment only considers single-task agents and single-agent tasks. Therefore, it is not suitable to simulate tasks that require several robots at the same time to complete. Also, machine breakdowns are not considered during schedule execution. Addressing those points is necessary towards deploying HybridNet in real-world scenarios. Second, we use Reinforcement Learning and need Reward

Engineering to develop a reward scheme to enable learning with infeasible schedule explorations, using Equation 5.2. We propose in future work to investigate methods to learn from sub-optimal demonstrations. We also plan to evaluate off-policy RL methods to improve sample efficiency in training HybridNet.

Similar to how the GeneticEDF Baseline improves upon the performance of the EDF as shown in experimental results, it is possible to use schedules found by HybridNet to warm-start the genetic algorithm's population. In future work, we propose to explore novel mechanisms for combining our framework with traditional planning techniques towards further performance gain.

Finally, we propose in future work to explore multi-task learning methods of joint training HybridNet under different objective functions beside minimizing overall makespan, and transfer learning between them.

5.9 Summary

We present a deep learning-based framework, called HybridNet, combining a heterogeneous graph-based encoder with a recurrent schedule propagator, for scheduling stochastic human-robot teams under temporal and spatial constraints. The resulting policy network provides a computationally lightweight yet highly expressive model that is end-to-end trainable via reinforcement learning algorithms. We developed MuRSE, a multi-round task scheduling environment for stochastic human-robot teams, and conducted extensive experiments, showing that HybridNet outperforms other human-robot scheduling solutions across various problem sizes.

CHAPTER 6
FAILURE-PREDICTIVE MAINTENANCE SCHEDULING USING HETEROGENEOUS GRAPH-BASED POLICY OPTIMIZATION

6.1 Introduction

Optimizing aircraft maintenance has drawn keen interest, due to the significant contribution of maintenance costs to overall operating expenses and aircraft availability [16]. One of the most promising strategies of reducing cost is by scheduling predictive maintenance, which entails deciding whether and when to preemptively service one or more of an aircraft's subsystems before the subsystem fails [17]. Research suggests that predictive maintenance could reduce unscheduled work up to 33% [18], which would result in an annual savings of $21.7 billion globally[1]. Currently, predictive maintenance scheduling is performed with ad hoc, hand-crafted heuristics and manual scheduling by human domain experts, which is a time-consuming and laborious process that is hard to scale. Because of these issues, researchers are becoming increasingly interested in developing automatic scheduling solutions that can not only provide high-quality schedules on large scale but also generalize to different application needs.

In this chapter, we propose an innovative design of the scheduling policy network operating on a heterogeneous graph representation of predictive-maintenance scheduling environment, as shown in Figure 6.1. Two keys to our approach are: 1) we directly model the dynamic scheduling decisions as nodes within a heterogeneous graph network, allowing for an end-to-end trainable resource scheduling policy that is capabale of reasoning over the various interactions within the environment, computationally lightweight and nonparametric to problem scales; 2) we develop an RL-based policy optimization procedure to enable

[1] Based upon 2012 figures for worldwide airline revenue of $598 Billion [19] and 11% of revenue allocated for maintenance [20].

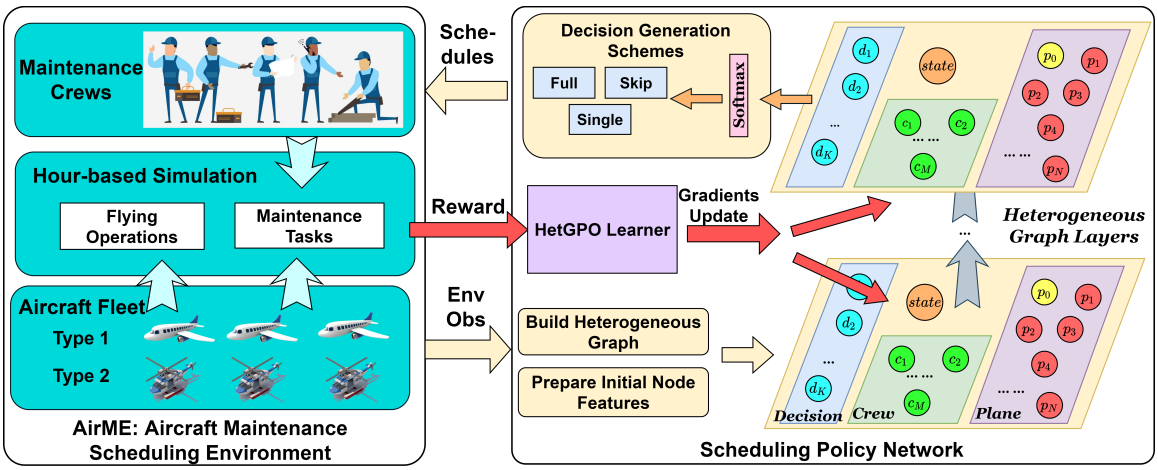


Figure 6.1: The figure depicts AirME, a virtual predictive-maintenance scheduling environment (Left), and our proposed scheduling policy network (Right). Left: AirME consisits of a team of maintenance crews and a heterogeneous fleet of aircraft and operates under hour-based simulation. Right: The scheduling policy network uses several heterogeneous graph layers (edges omitted for simplicity) stacked in series to extract high level embeddings from the graph built with environment observations. Different schemes are proposed and tested for generating dynamic scheduling decisions. We train our policy network via heterogeneous graph-based policy optimization, which we call HetGPO. HetGPO receives a reward signal from AirME and updates via gradient descent.

robust learning in highly stochastic environments for which typical actor-critic RL methods are ill-suited.

To evaluate our heterogeneous graph based policy optimization (HetGPO) approach, we worked in consultation with aerospace industry partners in developing a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, which we call AirME, as a testbed which will be released to public. The challenges for scheduling in AirME comes from the stochasticity in maintenance tasks and the uncertainty of potential components failure that greatly influences maintenance costs. We empirically validate HetGPO across a set of problem sizes and when optimizing for multiple objective functions. Results show HetGPO achieves a 29.1% improvement in airline profit over corrective scheduling and outperforms both heuristic and learning-based baselines.

6.2 Aircraft Maintenance Environment

Our research focuses on task scheduling for stochastic resource optimization, with application to failure-predictive aircraft maintenance. In a stochastic resource-constrained environment, both the resource allocation task and the outcome may be affected by latent stochastic processes (e.g., a plane may break randomly; the maintenance duration is non-deterministic). The scheduler must dynamically allocate resources to maximize application-specific objectives, given observations from the environment.

In consultation with aerospace industry partners along with their real-world experience on modeling approach for aircraft maintenance data, we develop a virtual predictive-maintenance scheduling environment for a heterogeneous fleet of aircraft, which we call AirME, as shown in Figure 6.1. In AirME, the stochasticity comes from the stochastic nature of aircraft maintenance work and the uncertainty of potential components failure that greatly influences maintenance costs. The AirME codebase and the supplementary metarials have been made publicly available[2].

An AirME instance consists of N_p planes, denoted as $\{p_i\}$, and N_c maintenance crews, denoted as $\{c_j\}$. We consider a heterogeneous fleet of aircraft including fixed-wing aircraft and helicopters and a homogeneous team of maintenance crews. Each airplane, p_i, has a set of observable parameters, $\{o_k^i\}$ such as operating time, total number of takeoffs, and engine status. A plane, p_i, is associated with a repeating maintenance task, m_i, a probabilistic failure model, $P_i(break|usage)$, and a repeating flying operation f_i, all affecting the plane's status during simulation. Each crew can be assigned a maintenance job, resulting in the crew becoming unavailable for further repairs until the current repair is complete. A maintenance decision, d, in AirME is specified by a 2-tuple $<p_i, c_j>$, consisting of an assignment of c_j to perform a specific maintenance operation (preemptive or otherwise) on p_i, starting at current time step.

AirME utilizes an hour-based time system and proceeds through the simulation in dis-

[2]https://github.com/CORE-Robotics-Lab/AirME

crete time steps. At the beginning of each hour, the environment receives maintenance decisions from a scheduler and updates the status of planes, crews and associated maintenance tasks and flying operations. Each plane's failure model is called to sample potential failures of its components. Then, AirME collects costs from all running maintenance tasks and hourly income from operating planes. Before stepping to the next hour, AirME releases completed maintenance tasks and flying operations.

6.2.1 Aircraft Failure Model

Each aircraft, p_i, has K_i number of components/parts depending on its type (i.e., airliner or helicopter). According to the MSG-3 document [154], failure refers to the inability of an item to perform within previously specified limits. For each component, its probability of failure is modeled using the Weibull distribution as a function of aircraft usage, such as flight hours or number of landings/takeoffs, as shown in Equation Equation 6.1, where x is the usage input, $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter. k and λ are randomly selected but hold constant across the same plane type.

$$p(x; \lambda, k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, x \geq 0, \tag{6.1}$$

A plane is grounded and changed to broken status when failure happens for at least one of its components. As a result, the plane failure model becomes a hybrid probabilistic model that jointly considers different plane parameters. Hyper-parameters of the failure model is not accessible to scheduling policies and instead must be inferred. Thus, HetGPO must implicitly learn a representation of this process in order to inform its decision-making policy.

Table 6.1 lists the hyper-parameters used for each aircraft type in our experiments. Those values are picked empirically so that the resulted probabilistic failure models have sufficiently different characteristics to test the heterogeneous graph neural networks ex-

Table 6.1: Hyper-parameters of Plane Failure Models

Aircraft Type	Number of Parts	Scales	Shapes	Hour Norm
Fixed-Wing Aircraft	4	[15, 12, 18, 16]	[5.0, 5.5, 6.0, 6.5]	20
Helicopter	3	[8, 7, 5]	[7, 6, 11]	15

pressiveness. For all types, the usage input of first part is number of landings, and the rest parts use flight hours as inputs. Additionally, we divide flight hours by the value specified under "Hour Norm" before being used as input.

6.2.2 Maintenance Task and Flying Operation

Each maintenance task is modeled as a stochastic process in which both the duration of the maintenance task and its cost are generated on-the-fly at the time when a crew is assigned to a plane. The duration of a maintenance task consists of a universal component drawn from a uniform distribution and a plane-specific part based on the plane's operation parameters. If one of the plane's subsystems is broken before a preemptive repair is performed, AirME labels the task as corrective maintenance and additional penalty time is added to its duration. The cost of a maintenance task includes: 1) a one-time, fixed cost proportional to the plane's hourly income; 2) the cost of labor proportional to maintenance time; 3) additional cost if part failure happens.

AirME assumes that each aircraft returns to the airbase where maintenance can be conducted after each operation. At the start of each time point, for every grounded available plane, the environment samples whether the plane will be used for an operation based upon a given plane type-specific usage rate. If so, the operation is enabled with a randomly sampled duration for this plane to execute. Different planes earn different hourly income, stored as their parameters, when flying.

We present the parameters of AirME instances used in our experiments. The values are picked empirically in consultation with aerospace industry partners. The usage rate for sampling a flying operation is 0.6 for fixed-wing aircraft and 0.3 for helicopters, assuming

helicopters are less often used. To support a heterogeneous fleet of aircraft, the hourly income of a plane is drawn randomly, from Uniform(1, 20) for fixed-wing aircraft and Uniform(1, 10) for helicopters. For each environment instance, at $t = 0$, random initial usage data are generated for each plane and we set $\sim$10% of the planes to be broken.

Maintenance Duration The universal component is drawn from Uniform(2, 8). The plane specific part is computed as $flight\_hours/24 + number\_of\_landings/6$. Penalty time for corrective maintenance is set to 12. Task duration is rounded to integer.

Maintenance Cost The one-time fixed part is computed as Uniform(0.1, 1)$\times hourly\_income$. The cost of labor is computed as $2 \times duration$. Additional failure cost is set to 48.

6.2.3 Scheduling Objectives

While a common objective for maintenance scheduling is maximizing the overall profit, other objectives exist to serve different needs. In AirME, we consider three objectives. **O1**: overall profit considering both hourly income and maintenance cost. **O2**: revenue only, similar to a situation where maintenance is provided at a fixed-price contract by a third-party. **O3**: fleet availability, a common objective in applications involving operation readiness and humanitarian crises [83].

6.2.4 POMDP Formulation

We formulate failure-predictive scheduling in AirME as a partially observable Markov decision process (POMDP) using a seven-tuple $<S, A, T, R, \Omega, O, \gamma>$below:

- States: The problem state S is a joint state consisting states of all planes and crews, plus the system hyper-parameters used in simulating flying operations and maintenance tasks.

- Actions: Action at time t is denoted as a collection of maintenance decisions, $U_t = \{d_1, d_2, ..., d_n\}$. Action space in AirME is flexible as a scheduler may issue as many maintenance decisions as wanted for one time step.

- Transitions: T corresponds to executing the action in AirME and proceed to next time step.

- Rewards: R_t is set to the same value as the scheduling objective a user wants to maximize.

- Observation: Ω contains $\{o_k^i\}$ of all planes. Additionally, it includes the observable status of all crews, current progress of flying operations and maintenance tasks.

- Observation Functions: O is handled by AirME to update the observations based on problem state after taking the action.

- Discount factor, γ.

6.3 Stochastic Scheduling with Graphs

We develop heterogenous graph representation of the scheduling environment and propose a novel heterogeneous graph layer that learns per-edge-type message passing and per-node-type feature reduction mechanisms on this graph. We directly build our scheduler policy over it to obtain a fully graph convolutional structure that are nonoparametric on problems sizes and dynamic action space, and are end-to-end trainable. Our approach is not restricted to AirME and is capable of modeling the heterogeneity among entities and their interaction in most resource optimization environments (e.g., nurse-patient scheduling in health care; coordinating mixed human robot teams in manufacturing/assembly line). In this section, we first describe how the scheduling policy network operates under the composite, dynamic action space. Next, we explain each component of the heterogeneous graph constructed at

a given time step. Finally, we detail the computation flow within the building block layer used to assemble a policy network of arbitrary depth.

6.3.1 Scheduling Policy Network

We denote the policy learned by our scheduling policy network as $\pi_\theta(u|o)$, with θ representing the parameters of the heterogeneous graph neural network. In AirME, an action takes the form of a collection of maintenance decisions, $u_t = \{d_1, d_2, ..., d_n\}$, with n varying from 0 (no maintenance scheduled) to N_{avail} (every available crew is assigned a plane). To handle this flexible action space, we reformulate u_t as an ordered sequence of scheduling decisions, where a latter decision (e.g., d_i) is conditioned on a former one (e.g., d_{i-1}). Then, the policy can be factorized as Equation 6.2.

$$p_\theta(u_t|o_t) = \prod_{i=1}^{n} p_\theta(d_i|o_t, d_{1:i-1}) \tag{6.2}$$

The scheduling policy network recursively computes the conditional probability, $p_\theta(d_i|o_t, d_{1:i-1})$, for sampling a maintenance decision. The heterogenous graph is modified after every maintenance decision, before being used for computing the next decision. At the end, the network collects all the decisions and sends to AirME for execution.

We test different schemes for determining n, the number of total decisions in u_t, in the decision generation block shown in Figure 6.1. Scheme #1) **Full**: For every crew available at t, the policy assigns a plane to it to conduct maintenance. We include a placeholder plane with ID 0, when picked, denoting "no-op" for the assigned crew. Scheme #2) **Skip**: While the policy still recursively assigns planes to available crews as in **Full**, plane 0 now functions as a "skip" token. Picking plane 0 means that the policy does not wish to schedule further maintenance tasks and wants to step into $t + 1$. In both variants, plane 0 allows the policy to learn to balance between spending current resources and reserving for future needs, which is an important challenge to reason about in such scheduling problems.

Scheme #3) **Single**: During training, we restrict the policy to only issue one scheduling decision in any time step, i.e., $\pi(u|o) = \pi(d|o)$. During testing, multi-decision actions are allowed by repeatedly sampling from the same distribution, $p_\theta(d|o_t)$ for every available crew. While sacrificing some performance, **Single** only needs one forward pass over the network and is thus more computationally efficient.

6.3.2 Heterogeneous Graph Representation

When developing a heterogeneous graph representation for a given stochastic resource optimmization problem, our computational target is to model entities in the environment (i.e., planes, crews) and RL components (i.e., state, decisions) in the same graph to enable joint learning the problem representation and the policy.

To begin with, we directly model each entity class in the resource optimization problem as a unique node type and their interactions as directed edges to build a heterogeneous graph. We use a three-tuple, $<srcName, edgeName, dstName>$, to specify the edge type/relation that connects two nodes (from source to destination). In AirME, this leads to two node types: the plane nodes and crew nodes. If a crew is conducting maintenance work for a plane, two types of edges are established between them: $<crew, reparing, plane>$ and $<plane, repaired\_by, crew>$. The observable parameters of each entity are used as its input node features.

Next, a state summary node is added and is connected by all the task nodes and agent nodes, with edge types $<plane, in, state>$, $<crew, in, state>$, respectively. The addition of state node enables the policy network to explicitly learn a high-level global embedding for estimating the problem state regardless of the problem scale. The initial input features of the state node are the meta-data defining the problem (e.g., the type and number of each aircraft).

To obtain an end-to-end trainable, graph-based policy, we augment the heterogeneous graph by introducing decision value nodes to allow the policy to handle varying num-

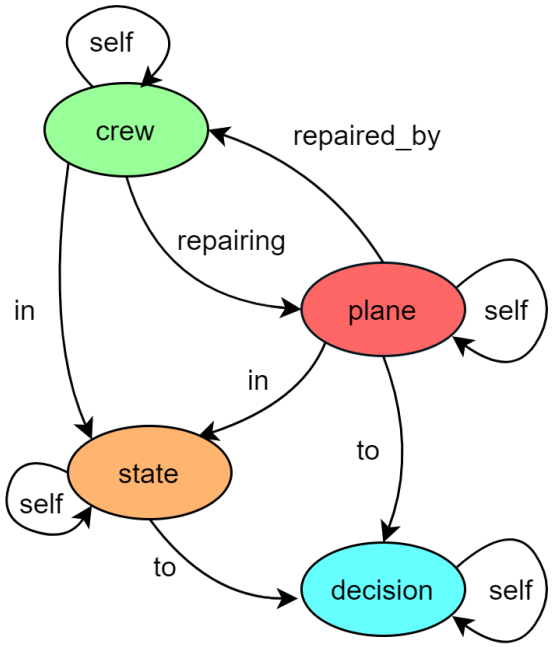


Figure 6.2: Metagraph of the heterogeneous graph built given an environment state in AirME.

ber of scheduling choices. Each scheduling decision is evaluated by a decision value node that is connected with the state node and associated entity nodes (e.g., the corresponding plane and crew in a maintenance task), using edge types $<state, to, decision>$, $<plane, to, decision>$, and $<crew, to, decision>$, respectively. The initial feature of a decision node is set to 0. As shown in Figure Figure 6.1, after the last heterogeneous graph layer, the scheduling policy network performs a softmax operation over all decision value nodes to obtain a probability distribution for picking each decision. As AirME involves a homogeneous team of maintenance crews, we simplify the graph by removing edges from crew nodes to decision nodes. We leave scheduling with heterogeneous crews as future work.

The metagraph for HetGPO applied to AirME is shown in Figure 6.2, which summarizes all the node types and edge types. For all nodes, self-loops are added so that their own features from previous layers are considered in current layer's computation.

6.3.3 Computation Flow of Graph Layers

We propose and implement a novel heterogeneous graph layer that operates on the heterogeneous graph structure and serves as the building block of our scheduling policy network. The feature update process in a heterogeneous graph layer is conducted in two steps: 1) per-edge-type message passing and then 2) 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 process the input feature from the source node, N_{src}, and sends the computation result to the destination node, N_{dst}. S is the input feature dimension of N_{src}, and D is the output feature dimension of N_{dst}. In the case that several edge types share names, we use $W_{srcName,edgeName}$ to distinguish between weight matrices.

Feature reduction is performed for each node type by aggregating received messages to compute a node's output features. The feature update formulas of different node types are listed in Equation 6.3-Equation 6.6, where $\sigma()$ represents the ReLU nonlinearity, and $N_{edgeType}(s)$ is the set of incoming neighbors of the state node s along the specified edge type.

$$\text{Plane } \vec{h}'_p = \sigma\Big(W_{plane,self}\vec{h}_p + W_{repairing}\vec{h}_c\Big) \tag{6.3}$$

$$\text{Crew } \vec{h}'_c = \sigma\Big(W_{repaired\_by}\vec{h}_p + W_{crew,self}\vec{h}_c\Big) \tag{6.4}$$

$$\begin{aligned}\text{State } \vec{h}'_s = \sigma\Big(&\sum_{p \in N_{plane,in}(s)} \alpha_{s,p}^{plane,in} W_{plane,in}\vec{h}_p \\ &+ \sum_{c \in N_{crew,in}(s)} \alpha_{s,c}^{crew,in} W_{crew,in}\vec{h}_c \\ &+ W_{state,self}\vec{h}_s\Big)\end{aligned} \tag{6.5}$$

$$\text{Decision } \vec{h}'_d = \sigma\Big(W_{plane,to}\vec{h}_p + W_{state,to}\vec{h}_s + W_{decision,self}\vec{h}_d\Big) \tag{6.6}$$

When computing output features of state summary node using Equation 4.6, we implement attention mechanisms adapted from [32] to weigh incoming messages for each edge type in a feature-dependent and structure-free manner. The per-edge-type attention coefficient, $\alpha_{s,i}^{edgeName}$, is calculated based on source node features and destination node features using Equation 4.10, where $\vec{a}_{edgeName}^T$ is the learnable weights, $||$ is the concatenation operation, and $\sigma'()$ is the LeakyReLU. The softmax function is used to normalize the coefficients across all choices of i.

$$\alpha_{s,i}^{edgeName} = \text{softmax}_i\Big(\sigma'\Big(\vec{a}_{edgeName}^T \Big[W_{state,self}\vec{h}_s || W_{edgeName}\vec{h}_i\Big]\Big)\Big) \tag{6.7}$$

To stabilize the learning process of self-attention, we utilize the multi-head mechanism that has been shown beneficial in homogeneous graphs [32], adapting it to fit the heterogeneous case. We use K independent heterogeneous graph (sub-)layers to compute node features in parallel and then merge the results as the multi-headed output either by concatenation or by averaging.

By stacking several heterogeneous graph layers sequentially (i.e., output from previous layer is directly used as input to the next layer), we construct the scheduling policy network that utilizes multi-layer structure to extract high-level embeddings of each node as shown in Figure 6.1. Note that all graph layers operate on the same heterogeneous graph built on current observation and share the same computation flow. However, the weight matrices (e.g., $W_{reparing}$, $W_{repaired\_by}$, $W_{plane,in}$) differ with each layer).

Algorithm 5: HetGPO Training

Input: Number of training epochs K, Number of episodes per epoch N, learning rate, η, number of gradient updates per epoch N_{iter}, episode length parameters: $T_{min}, T_{step}, T_{range}$

Output: Trained policy π_θ

Initialize policy network parameters θ_0
for $t = 1$ *to* K **do**
  Sample episode length $T \sim$ Uniform$(T_{min}, T_{min} + T_{range})$
  Sample a random environment instance, AirME_k
  for $i = 1$ *to* N **do**
    Reset AirME_k to $t = 0$
    for $t = 0$ *to* $T - 1$ **do**
      Get observation o_t^i from AirME_k
      Build heterogeneous graph and input node features
      Sample $u_t = \{d_1, d_2, ..., d_n\}$ from π_θ
      Step through AirME_k and get intermediate reward r_t^i
      Store $\{(o_t^i, u_t^i, r_t^i)\}$ to trajectory buffer
      if *broken planes* $\geq 80\%$ **then**
        Terminate current episode and zero-pad future rewards
      end
    end
  end
  Compute rewards-to-go: $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^i$
  Compute advantage estimates: $A_t^i = R_t^i - \frac{1}{N} \sum_{i'=1}^{N} R_t^{i'}$
  for $j = 1$ *to* N_{iter} **do**
    Compute the clipped surrogate objective $L(\theta)$ using Equation 7
    Perform stochastic gradient ascent for $\nabla L(\theta)$ using Adam optimizer with learning rate η
  end
  if $T_{min} \leq T_{max}$ **then**
    $T_{min} = T_{min} + T_{step}$
  end
end

6.4 Stochastic Policy Learning Methods

Our scheduling policy network is end-to-end trainable via Policy Gradient methods that seek to directly optimize the network's parameters based on rewards received from the environment. We develop our heterogeneous graph-based policy learning framework, which we call HetGPO, from Proximal Policy Optimization (PPO) [94] and make several adaptations for better variance reduction. In particular, we optimize the clipped surrogate objective shown in Equation 6.8, where $r_t(\theta)$ denote the probability ratio between current policy and old policy on collected rollout data, $r_t(\theta) = \frac{\pi_\theta(u|o)}{\pi_{\theta_{old}}(u|o)}$, A_t is the estimated advantage term, and ϵ is the clipping parameter.

$$L(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t] \tag{6.8}$$

In Equation 6.8, the advantage term, A_t is estimated by subtracting a "baseline" from the total future reward (or "return"). PPO and Actor-Critic methods typically utilze a learned state-based value function as such baselinses. However, in a stochastic environment, such as AirME, learning a helpful value function is non-trivial. This difficulty is due to the fact that the state dynamics and rewards are closely affected by an exogenous, random process (e.g., plane's failure model). Thus, the state alone provides limited information for predicting future expected return, resulting in high variance when learning a state-based value function. Instead, we choose to use a step-based baseline [96] as a more accessible and efficient alternative. Specifically, during gradient estimation, the baseline value being subtracted is set as the average of the return values, where the average is taken at the same time step across all training episodes.

We present the pseudocode for HetGPO training in algorithm 5. Lines 3-17 details the process of rollout data collection on AirME instances generated on-the-fly. Line 6 ensures the same randomly-initialized environment instance is used by all episodes within one training epoch for further variance reduction. Early termination based on percentage

of plane failures are enabled in lines 13-15 to penalize poor explorations. In line 19, a step-based baseline is computed by taking the average of rewards-to-go on all episodes at the same step. Lines 20-23 shows the gradient update procedure by maximizing $L(\theta)$, which requires recomputing the action probability using updated policy network at each iteration. Lines 24-26 implements a curriculum-based procedure in which initial training episodes are shorter, and the duration of episodes gradually increases, to avoid ineffective learning at the initial training phase.

Generalizability of HetGPO We develop HetGPO with the mindset of a general, graph-based policy learning algorithm to solve a broader class of stochastic resource optimization problems that are not restricted to aircraft maintenance scheduling. Both the heterogeneous graph formulation techniques (e.g., the useage of "state summary" and "decision value" nodes) and the HetGPO training process can be used in similar stochastic scheduling domains as they require little hand-engineering. We leave it as future work to apply our framework on similar domains such as patient admission scheduling in health care [155].

6.5 Experimental Results

In this section, we evaluate the utility of HetGPO against baselines under various application needs in AirME.

6.5.1 Baseline Methods

We benchmark HetGPO against a set of relevant baselines (i.e., heuristics) commonly employed for scheduling maintenance operations as well as modern machine learning-based approaches we adapt to the task. Further details for all baselines are provided in Supplementary.

Heuristics Baselines

We employ the following heuristics:

Random Scheduler At time t, the random baseline assigns each available crew a plane to start maintenance work on that is randomly picked from all planes that are not under maintenance (including placeholder plane #0 for "no op") to build u_t.

Corrective Scheduler [156] The corrective scheduler only schedules corrective maintenance tasks which address component failures that have occurred. When there are multiple planes with component failures at t, these planes are ranked into a priority queue based on hourly income.

Condition-Based Scheduler [157] Condition-based maintenance (CBM) has been shown to improve system efficiency by reducing the number of needed corrective maintenance tasks. Besides addressing all planes with component failures, the condition-based scheduler ranks the non-failure planes into another priority queue (e.g., based the flight hours or number of landings) and assigns the rest of crew for conducting CBM for them. A threshold, β_c, is set for planes without a failure to be eligible to enter the priority queue. Being a measure to balance plane availability and future failure risk, the choice of β_c greatly affects the scheduler's performance. To decide the choice of β_c, we test values from [0, 10, 20, 30, ..., 110, 120] for flight hours and [0, 1, 2, 3, ..., 11, 12] for number of landings on small environment instances and pick the best performing one. As a result, $\beta_c = 40$ for flight hours is used to generate evaluation results for all objectives.

Periodic Scheduler [74] The periodic scheduler schedules regular occurring maintenance tasks using a prescribed time interval, β_p. After β_p amount of time has passed, the periodic scheduler assign every available crew a plane for conducting maintenance. Planes are ranked first by failure status and then by flight hours. A periodic scheduler's perfor-

mance is closely related to the choice of β_p. Note that a periodic scheduler with $\beta_p = 1$ is equivalent to a condition-based scheduler with $\beta_c = 0$. We test values from [1, 2, 3, ..., 11, 12] for β_p on small environment instances and pick the best performing one. As a result, $\beta_p = 6$ is used to generate results for all objectives.

Model-based Planning

To construct a model-based scheduler, we augment the condition-based scheduler by giving it access to the plane failure model used in the environment. In this case, non-failure planes are ranked based on their truth failure probability used in the environment sampling process for next time step. A threshold, β_p, on failure probability is set for planes to be eligible to enter the priority queue. We test various values for β_p on small environment instances and pick the best performing one, $\beta_p = 0.004$.

Machine Learning-Based Methods

We consider two methods in prior works for resource scheduling and adapt them to AirME:

DeepRM [95] DeepRM represents the current allocation of resources as fixed-size tensors and uses feed-forward neural network to learn a policy of fixed output dimension. The input to the policy network is constructed by concatenating the flattened features of all planes, crews and state. To enable DeepRM to handle variable problem sizes, we zero-pad the flattened plane- and crew-feature tensor to contain the maximum number of planes and crews. The output dimension of policy network is also set to the maximum number of planes. When generating the decision probability distribution, we mask out the unvalid planes (i.e., planes already under repair) from the network output. We train separate DeepRM models for small, medium and large environments in AirME. The policy network consists of four fully-connected layers, with hidden dimension of 64 for small, 128 for medium and large environments. DeepRM learns by REINFORCE with step-based

baselines.

Decima [96] Decima utilizes a scalable architecture that combines a graph neural network to process jobs/tasks and a separate policy network that makes decisions triggered by scheduling events. In AirME, the graph neural network used by Decima is a bipartite graph containing maintenance task nodes as children nodes and a global state summary node as the parent node. Considering homogeneous crews, we model maintenance task nodes similarly as the plane nodes used in HetGPO. Message passing in Decima is conducted as Equation Equation 6.9.

$$h_s = g\big(\sum_{m \in N(s)} f(h_m)\big), \tag{6.9}$$

where g(·) and f(·) are non-linear transformations implemented as neural networks. h_m are the input features of maintenance tasks and h_s are the global state embeddings. In Decima, a scheduling event is triggered when a worker (maintenance crew) becomes available. Decima's separate policy network computes a score $q_m = q(f(h_m), h_s)$ for each candidate maintenance task m. $q(\cdot)$ is a score function that takes as input the global state embeddings and transformed task embeddings. Decima then uses a softmax operation over all the scores to compute the probability of selecting each task as Equation Equation 6.10.

$$p(m) = \frac{\exp(q_m)}{\sum_{m' \in M} \exp(q_{m'})}, \tag{6.10}$$

where M is the set of all candiate tasks. Keeping consistent with the original paper, we implement g(·), f(·) and q(·) as separate neural networks in AirME, each consisting of three fully-connected layers with ReLU activation and hidden dimension of 64. Decima is trained by REINFORCE with step-based baselines.

6.5.2 Evaluation Settings

Evaluation Dataset Environment instances with various problem sizes and random initialization are generated and saved as test dataset. Three environment scales are considered: 1) Small: the ranges of fixed-wing aircraft, helicopters and crews are chosen from the ranges [16, 24], [8, 12], and [6, 8], respectively, using uniform distributions. 2) Medium: the corresponding ranges are [32, 48], [16, 24], and [12, 16]. 3) Large: the corresponding ranges are [64, 96], [32, 48], and [24, 32]. For each testing environment instance, we evaluate a method for ten episodes and record the overall performance. Each evaluation episode starts by loading the test environment instance and runs the scheduler for a fix length of duration (30 days used).

Evaluation Metrics For each objective discussed in Section section 6.2, we use two evaluation metrics. **M1**: normalized objective value. Normalization is performed w.r.t the objective value obtained when assuming all planes are flying without failure; **M2**: % improvement over the Corrective Scheduler.

Model Details We implement HetGPO[3] using PyTorch [144] and Deep Graph Library [152]. The policy network used in training/testing is constructed by stacking three multi-head heterogeneous graph layers (the first two use concatenation, and the last one uses averaging). The feature dimension of hidden layers = 32, and the number of heads = 4. We set γ = 0.99, and used Adam optimizer with a learning rate of 1e-3. All variants of HetGPO are trained with small environment instances generated on-the-fly. In algorithm 5, we set $K = 2000$, $N = 8$, $\eta = 10^{-3}$, $N_{iter} = 3$, $T_{min} = 50$, $T_{step} = 0.8$, $T_{max} = 200$, $T_{min} = 30$. The clipping parameter ϵ in Equation 6.8 is set to 0.2. Models are trained and evaluated on a Nvidia A40 Data Center GPU and a AMD EPYC 7452 32-Core CPU.

[3] https://github.com/CORE-Robotics-Lab/AirME

Table 6.2: Evaluation results on O1: profit.

Methods	Small		Medium		Large	
	M1	**M2 (%)**	**M1**	**M2 (%)**	**M1**	**M2 (%)**
Random	0.522 ± 0.025	-2.87 ± 5.65	0.532 ± 0.021	-2.65 ± 4.22	0.533 ± 0.016	-2.23 ± 3.51
Corrective	0.539 ± 0.023	0.0 ± 0.0	0.547 ± 0.016	0.0 ± 0.0	0.546 ± 0.016	0.0 ± 0.0
Condition-based	0.656 ± 0.050	21.7 ± 6.41	0.661 ± 0.041	20.8 ± 5.87	0.648 ± 0.051	18.6 ± 7.03
Periodic	0.599 ± 0.051	11.1 ± 6.96	0.598 ± 0.047	9.38 ± 7.00	0.587 ± 0.048	7.52 ± 6.83
Model-based	0.669 ± 0.052	24.0 ± 6.99	0.671 ± 0.044	22.8 ± 6.45	0.658 ± 0.054	20.5 ± 7.68
DeepRM	0.533 ± 0.015	-0.88 ± 2.57	0.538 ± 0.011	-1.47 ± 1.77	0.539 ± 0.013	-1.11 ± 0.97
Decima	0.651 ± 0.021	21.1 ± 6.42	0.660 ± 0.017	20.9 ± 4.49	0.663 ± 0.014	21.6 ± 4.51
HetGPO-Single	0.680 ± 0.012	26.4 ± 4.32	0.676 ± 0.011	23.7 ± 3.17	0.666 ± 0.011	22.3 ± 3.77
HetGPO-Skip	**0.695 ± 0.010**	**29.1 ± 4.09**	**0.697 ± 0.009**	**27.5 ± 2.70**	**0.695 ± 0.008**	**27.5 ± 2.72**
HetGPO-Full	0.693 ± 0.011	28.8 ± 4.01	0.694 ± 0.009	27.1 ± 2.62	0.693 ± 0.008	27.1 ± 2.68

Table 6.3: Evaluation results on O2: total revenue. Note that for each 1% of improvement for O2, we would get a $0.6578 Billion revenue increase. e.g., for Large-O2, HetGPO-Full would achieve a $9.27 Billion increase in revenue.

Methods	**Small**		**Medium**		**Large**	
	M1	**M2 (%)**	**M1**	**M2 (%)**	**M1**	**M2 (%)**
Random	0.611 ± 0.016	-5.63 ± 3.85	0.618 ± 0.014	-5.74 ± 2.87	0.618 ± 0.012	-5.68 ± 2.41
Corrective	0.648 ± 0.022	0.0 ± 0.0	0.656 ± 0.015	0.0 ± 0.0	0.655 ± 0.018	0.0 ± 0.0
Condition-based	0.724 ± 0.035	11.6 ± 2.71	0.727 ± 0.030	10.7 ± 2.61	0.718 ± 0.036	9.45 ± 2.87
Periodic	0.675 ± 0.038	4.06 ± 3.47	0.677 ± 0.033	3.05 ± 3.39	0.669 ± 0.035	2.05 ± 3.17
Model-based	0.728 ± 0.037	12.3 ± 2.91	0.732 ± 0.030	11.5 ± 2.66	0.723 ± 0.037	10.3 ± 2.98
DeepRM	0.625 ± 0.014	-3.54 ± 1.62	0.626 ± 0.011	-4.66 ± 1.38	0.622 ± 0.012	-5.02 ± 1.24
Decima	0.725 ± 0.011	11.9 ± 4.57	0.730 ± 0.010	11.3 ± 3.17	0.732 ± 0.007	11.8 ± 3.55
HetGPO-Single	0.736 ± 0.008	13.7 ± 3.72	0.735 ± 0.008	12.0 ± 2.67	0.730 ± 0.007	11.5 ± 3.24
HetGPO-Skip	0.747 ± 0.008	15.3 ± 3.43	0.748 ± 0.007	14.0 ± 2.29	0.747 ± 0.006	14.0 ± 2.72
HetGPO-Full	**0.747±0.008**	**15.4 ± 3.41**	**0.749 ± 0.006**	**14.1 ± 2.26**	**0.747 ± 0.006**	**14.1 ± 2.73**

Table 6.4: Evaluation results on O3: fleet availability.

Methods	Small		Medium		Large	
	M1	**M2 (%)**	**M1**	**M2 (%)**	**M1**	**M2 (%)**
Random	0.699 ± 0.015	-2.97 ± 4.27	0.706 ± 0.012	-3.26 ± 3.43	0.705 ± 0.012	-3.00 ± 3.39
Corrective	0.722 ± 0.033	0.0 ± 0.0	0.730 ± 0.027	0.0 ± 0.0	0.728 ± 0.032	0.0 ± 0.0
Condition-based	0.810 ± 0.057	12.0 ± 4.18	0.812 ± 0.052	11.1 ± 3.82	0.796 ± 0.065	9.16 ± 4.68
Periodic	0.747 ± 0.080	3.19 ± 7.20	0.743 ± 0.074	1.48 ± 7.05	0.725 ± 0.081	-0.67 ± 7.33
Model-based	0.814 ± 0.058	12.6 ± 4.27	0.817 ± 0.051	11.8 ± 3.76	0.803 ± 0.065	10.0 ± 4.63
DeepRM	0.708 ± 0.017	-1.82 ± 2.30	0.709 ± 0.013	-2.86 ± 2.45	0.706 ± 0.013	-2.90 ± 3.22
Decima	0.748 ± 0.031	3.98 ± 8.85	0.755 ± 0.025	3.69 ± 7.10	0.760 ± 0.022	4.76 ± 7.63
HetGPO-Single	0.842 ± 0.007	16.8± 4.84	0.840 ± 0.005	15.2 ± 3.99	0.837 ± 0.004	15.1 ± 4.94
HetGPO-Skip	0.847 ± 0.007	17.6 ± 4.89	0.848 ± 0.005	16.3 ± 3.99	0.847 ± 0.005	16.6 ± 4.86
HetGPO-Full	**0.849 ± 0.007**	**17.7 ± 4.90**	**0.849 ± 0.005**	**16.5 ± 3.98**	**0.849 ± 0.005**	**16.8 ± 4.84**

6.5.3 Evaluation Results

We present the evaluation results under three objectives in Table 6.2-Table 6.4, where both mean and standard deviation are listed. The corrective scheduler is used as the baseline method when computing **M2**.

HetGPO outperforms all baselines across all objectives. HetGPO-Skip and HetGPO-Full performs similarly, with HetGPO-Single achieving slightly worse. HetGPO-Skip and HetGPO-Full's performance remains consistent from small scale to large scale, while a performance drop is observed for HetGPO-Single on large scale. This is due to the difference in training and testing for HetGPO-Single, which trades some performance for better computation efficiency.

The biggest improvement in **M2** of HetGPO is observed in **O1**, up to 29.1%. HetGPO outperforms the condition-based scheduler, which is the best performing heuristic method, by $\sim 8\%$ in **O1**, $\sim 4\%$ in **O2**, and $\sim 6\%$ in **O3**. The condition-based scheduler benefits from the priority queue as a mechanism to estimate the likelihood of plane failures. Both the condition-based scheduler and the periodic scheduler rely on hand-picked threshold values, and their performance drops quickly if β_c or β_p deviates from optimal.

With access to the plane failure model, the model-based scheduler improves over condition-based scheduler, but is still outperformed by HetGPO. As we have heterogeneous fleet and stochastic maintenance task, it is not trivial to design effective heuristics even with access to truth failure sampling probability. On the other hand, thanks to the heterogeneous graph formulation, HetGPO is capable of automatically learning to implicitly reason about the plane failure and maintenance specifics toward optimizing scheduling objectives, without the help of expert domain knowledge. Note that HetGPO can be complementary to symbolic and model-based methods, and we leave it as future work to explore novel mechanisms for combining HetGPO with model-based planning techniques towards further performance gain.

DeepRM fails to learn useful scheduling policies and the performance is close to the

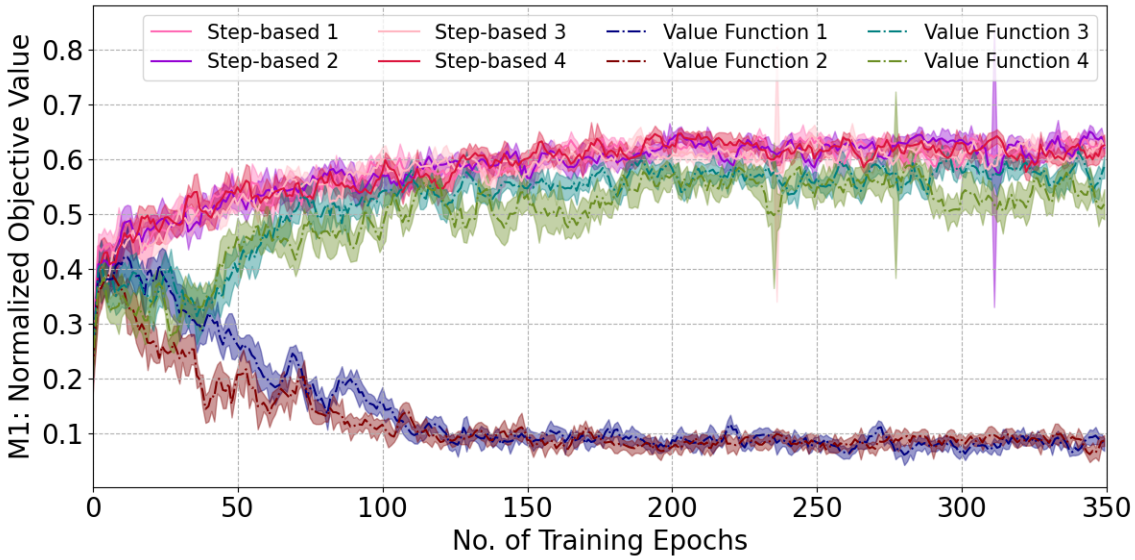


Figure 6.3: HetGPO-Single training on **O1** with a step-based baseline vs. state-based value function. Numbers in the legend denote the random seeds used.

random scheduler. This shows that the fixed-size tensor representation DeepRM uses is inefficient in modeling heterogeneous scheduling environments. On the other hand, Decima uses graph neural networks to learn from the structure information with the help of a global summary node. Decima is able to learn scheduling policies that are comparable with the condition-based scheduler under **O1** and **O2**. However, the graph structure in Decima only allows for one round of message passing among its nodes, and the summary node does not utilize attention. The limitation in model expressiveness makes Decima perform worse than HetGPO. In addition to achieving superior performance across problem sizes and various objective functions, HetGPO policies are more robust and consistent than other methods, with **M1** standard deviation up to 5x smaller than heuristics and 2x smaller than Decima.

6.5.4 Ablation Studies

Here, we investigate the effectiveness of step-based baselines over standard PPO training with a state-based value function. We add a critic head to process the output node feature of state node for value function prediction. We include the amount of time left in an episode as

separate input to the critic head during training, because the time information affects value estimation. Figure 6.3 shows the learning curves of different baseline choices on 4 random seeds under **O1**, using **Single** variant. Due to the stochasticity from both the maintenance work and the plane failure process, learning a state-based value function made the policy learning nosier than using step-based baselines. As shown in Figure 6.3, two seeds failed to learn a helpful value function, in which the policy performance decreased as training continued. When a value function was learned, the policy performance was still inferior than policies trained with step-based baselines.

6.6 Summary

Inspired by recent advances in leveraging deep learning to solve operations research problems, we propose an innovative design of heterogeneous graph neural networks-based policy for automatically learning the decision-making for failure-predictive maintenance scheduling. We directly build the scheduling policy into a heterogeneous graph representation of the environment to obtain a fully convolutional structure, providing a computationally lightweight and nonparametric means to perform dynamic scheduling. Furthermore, we develop an RL-based policy optimization procedure, called HetGPO, to enable robust learning in highly stochastic environments. AirME, a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, is designed and implemented as a testbed. Experimental results across various problem scales and objective functions (e.g., profit- and availability-based) show the effectiveness of our proposed framework over conventional, hand-crafted heuristics and baseline learning methods.

CHAPTER 7
CONCLUSION AND FUTURE WORK

In this dissertation, we explore deep-learning based methods for solving resource optimization problems and how to best leverage graph neural networks for effective policy learning. This chapter concludes our work, discusses limitations, and gives potential future work plans.

7.1 Conclusion

We build a unified framework of learning scalable scheduling policies for effectively solving resource optimization problems. We validate this framework in different scheduling scenarios ranging from multi-robot task scheduling, human-robot coordination to aircraft maintenance scheduling.

Our investigation begins with homogeneous robot teams. In Chapter 2, 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 networks in a non-parametric framework, we were able to obtain a policy that generated fast, near-optimal scheduling of robot teams. We demonstrated that our network-based policy found significantly more solutions than prior state-of-the-art methods in all testing scenarios.

To extend our work to allow scheduling robots with different capabilities, in Chapter 3, we presented a novel heterogeneous graph attention network model, called ScheduleNet, to learn a 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, and generalizes well to large, unseen problems. Empirically, we showed that our method outperformed existing state-of-the-art methods in a variety of testing scenarios involving both homogeneous robot teams and heterogeneous robot teams.

One of the big challenges of ScheduleNet is that the training requires optimal expert demonstrations and task durations are assumed to be deterministic and known a priori. To overcome this issue, ways of learning stochastic scheduling policies are being explored. Chapter 4 introduces a deep learning-based hybrid framework, called HybridNet, combining a heterogeneous graph-based encoder with a recurrent schedule propagator, for scheduling stochastic human-robot teams under temporal and spatial constraints. The resulting policy network provides a computationally lightweight yet highly expressive model that is end-to-end trainable via reinforcement learning algorithms. We developed MuRSE, a multi-round task scheduling environment for stochastic human-robot teams, and conducted extensive experiments, showing that HybridNet outperforms other human-robot scheduling solutions across various problem sizes.

Finally, in Chapter 5, we focus on learning the decision-making for failure-predictive maintenance scheduling. We directly build the scheduling policy into a heterogeneous graph representation of the environment to obtain a fully convolutional structure, providing a computationally lightweight and nonparametric means to perform dynamic scheduling. We developed an RL-based policy optimization procedure, called HetGPO, to enable robust learning in highly stochastic environments. AirME, a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, is designed and implemented as a testbed. Experimental results across various problem scales and objective functions (e.g., profit- and availability-based) show the effectiveness of our proposed framework over conventional, hand-crafted heuristics and baseline learning methods.

7.2 Limitations and Future Work

We note the limitations of our research, as listed below.

- First, we only address high-level task planning while ignoring task execution details. For multi-robot and human-robot teams, motion planning and path planning of each agent after task-agent assignment are not considered. We also assume the task duration is considerably larger than agent travel time. In scenarios where such assumption no longer holds, the schedule provided by our learner may become infeasible.

- Second, the training of both the RoboGNN and ScheduleNet models requires high-quality expert data as the loss function assumes the experts choose the optimal scheduling action at each time step. Therefore, sub-optimal demonstrations would lead to degraded model performance or even the wrong direction of gradient updates. Also, the proposed imitation learning method no longer works in stochastic scenarios. Although we can train HybridNet with reinforcement learning methods in such cases, the training requires delicate reward engineering and takes much longer time compared to imitation learning. Therefore, it is beneficial to investigate policy learning methods that can explicitly reason about sup-optimality in scheduling demonstrations.

- Third, although our GNN-based schedulers scale to different problem and team sizes, we observe a performance drop when the problem scale increases. Considering that our approach only trains on small scale problems, applying the learned representation to large scale problems, it remains an important open problem to investigate fine-tuning and transfer learning methods for improving the performance as problem sizes increases.

In future work, besides addressing the above-mentioned limitations, there are also several directions to extend our work, which are listed below.

Learning across Different Objective Functions

In ScheduleNet and HetGPO in AirME, we show that the same model structure can be trained to learn policies with different objective functions depending on application needs. However, those training processes are independent of each other. While different objectives require different scheduling strategies, they operate in the same problem state space. We plan to investigate if the learned high-level representation of one objective can facilitate the learning process when optimizing a different but related objective. Therefore, we propose in future work to explore multi-task learning methods of joint learning under different objective functions and transfer learning between them.

Combining Deep Learning Models with Heuristics

Our graph neural network-based models are developed to be complementary to symbolic and heuristic methods. For example, in multi-robot coordination, one could leverage ScheduleNet's task selector as a branching or selection policy for model-based search methods. It is also possible to use schedules found by HybridNet to warm-start the Genetic Algorithm's population. In future work, we propose to explore novel mechanisms for combining our framework with traditional planning techniques towards further performance gain.

Generalizing HetGPO to Broader Domains

We develop HetGPO with the mindset of a general, graph-based policy learning algorithm to solve a broader class of stochastic resource optimization problems that are not restricted to aircraft maintenance scheduling. The heterogenous graph is built by first modeling each entity class of the domain as a unique node type and their interactions as directed edges (i.e., the base graph) and then adding "state summary" node and "decision value" nodes. While constructing the base graph depends on the specific domain, the modeling is relatively straightforward and requires little hand-engineering. The input node features are merely the observables from the environment and do not require feature engineering. In

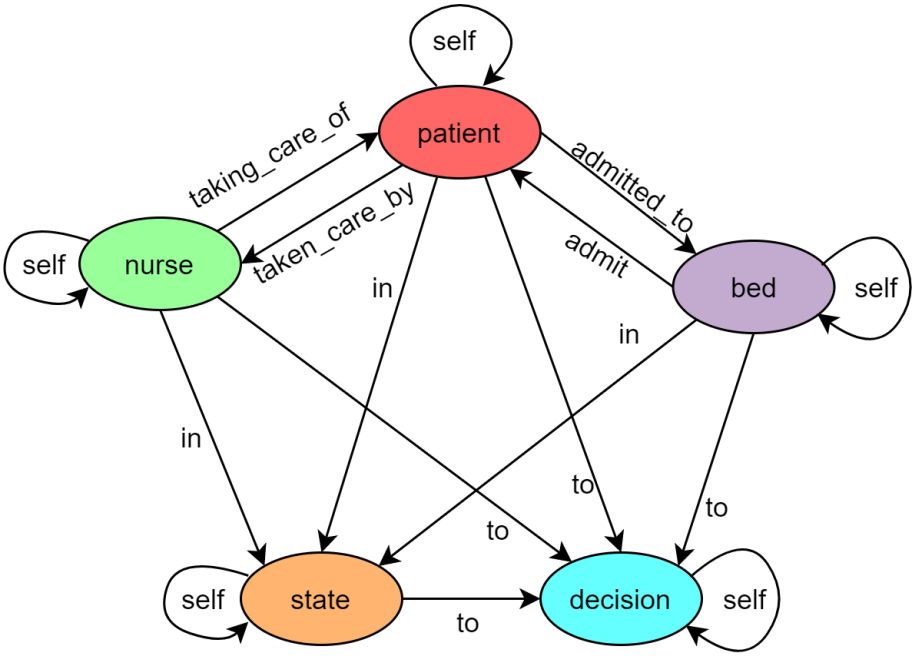


Figure 7.1: Metagraph of the heterogeneous graph built for patient admission scheduling problems.

future, we plan to apply HetGPO on similar stochastic resource optimization domains that require predictive scheduling efforts, including patient admission scheduling in health care. Recent research has shown success in training GNN-based models with imitation learning for solving the staff rerostering problem [158].

Take patient admission for the delivery room from [159] as an example, the base graph of the scenario can be built by modeling nurses, beds and patients as different types of nodes, with edges denoting their interaction. By adding the "state summary" and "decision value" nodes to the base graph, we obtain the heterogeneous graph used by the scheduling policy network of HetGPO, as shown in Figure 7.1. In addition to the state summary node, a decision node now also connects with a patient, a nurse and a bed to estimate the outcome of admitting the selected patient. Then, Algorithm 5 can be used to learn scheduling policies under the objective functions defined for hospital scenarios.

REFERENCES

[1] B. Zhou, J. Bao, J. Li, Y. Lu, T. Liu, and Q. Zhang, "A novel knowledge graph-based optimization approach for resource allocation in discrete manufacturing workshops," *Robotics and Computer-Integrated Manufacturing*, vol. 71, p. 102 160, 2021.

[2] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[3] Y. Kantaros and M. M. Zavlanos, "Global planning for multi-robot communication networks in complex environments," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1045–1061, 2016.

[4] J. Alonso-Mora, S. Baker, and D. Rus, "Multi-robot formation control and object transport in dynamic environments via constrained optimization," *The International Journal of Robotics Research*, vol. 36, no. 9, pp. 1000–1021, 2017.

[5] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial intelligence*, vol. 219, pp. 1–24, 2015.

[6] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[7] C. Sarkar, H. S. Paul, and A. Pal, "A scalable multi-robot task allocation algorithm," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 5022–5027.

[8] E. F. Flushing, L. M. Gambardella, and G. 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)*, IEEE, 2017, pp. 1861–1868.

[9] 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.

[10] A. Vysocky and P. Novak, "Human-robot collaboration in industry," *MM Science Journal*, vol. 9, no. 2, pp. 903–906, 2016.

[11] A. Ajoudani, A. M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, and O. Khatib, "Progress and prospects of the human–robot collaboration," *Autonomous Robots*, vol. 42, no. 5, pp. 957–975, 2018.

[12] R. Liu, M. Natarajan, and M. C. Gombolay, "Coordinating human-robot teams with dynamic and stochastic task proficiencies," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 11, no. 1, pp. 1–42, 2021.

[13] M. C. Gombolay, C. Huang, and J. Shah, "Coordination of human-robot teaming with human task preferences," in *2015 AAAI Fall Symposium Series*, 2015.

[14] J. A. Shah and B. C. Williams, "Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation.," in *ICAPS*, 2008, pp. 322–329.

[15] D. Dinis, A. Barbosa-Póvoa, and Â. P. Teixeira, "A supporting framework for maintenance capacity planning and scheduling: Development and application in the aircraft mro industry," *International Journal of Production Economics*, vol. 218, pp. 1–15, 2019.

[16] M. A. Bajestani and J. C. Beck, "Scheduling an aircraft repair shop," in *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[17] I. Haloui, C. Ponzoni Carvalho Chanel, and A. Haït, "Towards a hierarchical modelling approach for planning aircraft tail assignment and predictive maintenance," in *the 13th International Scheduling and Planning Applications woRKshop (SPARK)*, 2020.

[18] The Economist, *Artificial intelligence is changing every aspect of war*, Accessed: 2021-12-15, 2019.

[19] IATA, *International Air Transport Association Annual Report 2012*. 2012.

[20] Scott McCartney, *How airlines spend your airfare*, Accessed: 2021-12-15, 2012.

[21] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, "Event-based milp models for resource-constrained project scheduling problems," *Computers & Operations Research*, vol. 38, no. 1, pp. 3–13, 2011.

[22] 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.

[23] V. Tereshchuk, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "A scheduling method for multi-robot assembly of aircraft structures with soft task precedence

constraints," *Robotics and Computer-Integrated Manufacturing*, vol. 71, p. 102 154, 2021.

[24] E. R. López-Santana and G. A. Méndez-Giraldo, "A knowledge-based expert system for scheduling in services systems," in *Workshop on Engineering Applications*, Springer, 2016, pp. 212–224.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[26] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[27] 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.

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

[29] T. Ma, P. Ferber, S. Huo, J. Chen, and M. Katz, "Online planner selection with graph neural networks and adaptive scheduling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5077–5084.

[30] 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.

[31] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

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

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

[34] 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.

[35] M. M. Solomon, "On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints," *Networks*, vol. 16, no. 2, pp. 161–174, 1986.

[36] M. Caridi and S. Cavalieri, "Multi-agent systems in production planning and control: An overview," *Production Planning & Control*, vol. 15, no. 2, pp. 106–118, 2004.

[37] Y. N. Sotskov and N. V. Shakhlevich, "Np-hardness of shop-scheduling problems with three jobs," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 237–266, 1995.

[38] C. Le Pape, "A combination of centralized and distributed methods for multi-agent planning and scheduling," in *Proceedings., IEEE International Conference on Robotics and Automation*, IEEE, 1990, pp. 488–493.

[39] R. J. Rabelo and L. Camarinha-Matos, "Negotiation in multi-agent based dynamic scheduling," *Robotics and computer-integrated manufacturing*, vol. 11, no. 4, pp. 303–309, 1994.

[40] X. Zheng and S. Koenig, "K-swaps: Cooperative negotiation for solving task-allocation problems," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

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

[42] 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.

[43] 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.

[44] M. Koes *et al.*, "Heterogeneous multirobot coordination with spatial and temporal constraints," in *AAAI*, vol. 5, 2005, pp. 1292–1297.

[45] A. Prorok, M. A. Hsieh, and V. Kumar, "Fast redistribution of a swarm of heterogeneous robots," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2016, pp. 249–255.

[46] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 758–768, 2002.

[47] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," *Journal of Intelligent & Robotic Systems*, vol. 80, no. 1, pp. 33–58, 2015.

[48] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar, "Grstaps: Graphically recursive simultaneous task allocation, planning, and scheduling," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 232–256, 2022.

[49] H. Hanna, "Decentralized approach for multi-robot task allocation problem with uncertain task execution," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005, pp. 535–540.

[50] C. Ferreira, G. Figueira, and P. Amorim, "Scheduling human-robot teams in collaborative working cells," *International Journal of Production Economics*, vol. 235, p. 108 094, 2021.

[51] Y. Qu, X. Ming, Z. Liu, X. Zhang, and Z. Hou, "Smart manufacturing systems: State of the art and future trends," *The International Journal of Advanced Manufacturing Technology*, vol. 103, no. 9, pp. 3751–3768, 2019.

[52] S.-D. Lee, M.-C. Kim, and J.-B. Song, "Sensorless collision detection for safe human-robot collaboration," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 2392–2397.

[53] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tolio, "Motion planning and scheduling for human and industrial-robot collaboration," *CIRP Annals*, vol. 66, no. 1, pp. 1–4, 2017.

[54] P. A. Hancock, D. R. Billings, K. E. Schaefer, J. Y. Chen, E. J. De Visser, and R. Parasuraman, "A meta-analysis of factors affecting trust in human-robot interaction," *Human factors*, vol. 53, no. 5, pp. 517–527, 2011.

[55] M. Natarajan and M. Gombolay, "Effects of anthropomorphism and accountability on trust in human robot interaction," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 33–42.

[56] H. Liu and L. Wang, "Gesture recognition for human-robot collaboration: A review," *International Journal of Industrial Ergonomics*, vol. 68, pp. 355–367, 2018.

[57] A. M. Zanchettin, A. Casalino, L. Piroddi, and P. Rocco, "Prediction of human activity patterns for human–robot collaborative assembly tasks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 3934–3942, 2018.

[58] J. Kolb, M. Kishore, K. Shaw, H. Ravichandar, and S. Chernova, "Predicting individual human performance in human-robot teaming," in *2021 30th IEEE Inter-*

national Conference on Robot & Human Interactive Communication (RO-MAN), IEEE, 2021, pp. 45–50.

[59] J. C. Mateus, D. Claeys, V. Limère, J. Cottyn, and E.-H. Aghezzaf, "A structured methodology for the design of a human-robot collaborative assembly workplace," *The International Journal of Advanced Manufacturing Technology*, vol. 102, no. 5, pp. 2663–2681, 2019.

[60] R. Paleja, M. Ghuy, N. Ranawaka Arachchige, R. Jensen, and M. Gombolay, "The utility of explainable ai in ad hoc human-machine teaming," *Advances in Neural Information Processing Systems*, vol. 34, pp. 610–623, 2021.

[61] W. Xu, Q. Tang, J. Liu, Z. Liu, Z. Zhou, and D. T. Pham, "Disassembly sequence planning using discrete bees algorithm for human-robot collaboration in remanufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 62, p. 101 860, 2020.

[62] M. Rizwan, V. Patoglu, and E. Erdem, "Human-robot collaborative assembly planning using hybrid conditional planning," in *Proc. FAIM/ISCA Workshop on Artificial Intelligence for Multimodal HRI*, 2018, pp. 23–26.

[63] A. Cherubini, R. Passama, A. Crosnier, A. Lasnier, and P. Fraisse, "Collaborative manufacturing with physical human–robot interaction," *Robotics and Computer-Integrated Manufacturing*, vol. 40, pp. 1–13, 2016.

[64] L. Wang, B. Schmidt, and A. Y. Nee, "Vision-guided active collision avoidance for human-robot collaborations," *Manufacturing Letters*, vol. 1, no. 1, pp. 5–8, 2013.

[65] H. Ding, M. Schipper, and B. Matthias, "Optimized task distribution for industrial assembly in mixed human-robot environments-case study on io module assembly," in *2014 IEEE international conference on automation science and engineering (CASE)*, IEEE, 2014, pp. 19–24.

[66] N. Nikolakis, N. Kousi, G. Michalos, and S. Makris, "Dynamic scheduling of shared human-robot manufacturing operations," *Procedia CIRP*, vol. 72, pp. 9–14, 2018.

[67] A. Casalino, A. M. Zanchettin, L. Piroddi, and P. Rocco, "Optimal scheduling of human–robot collaborative assembly operations with time petri nets," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 70–84, 2019.

[68] S. Zhang, Y. Chen, J. Zhang, and Y. Jia, "Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability*," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3860–3866, 2020.

[69] Y. Ran, X. Zhou, P. Lin, Y. Wen, and R. Deng, "A survey of predictive maintenance: Systems, purposes and approaches," *arXiv preprint arXiv:1912.07383*, 2019.

[70] L. Swanson, "Linking maintenance strategies to performance," *International journal of production economics*, vol. 70, no. 3, pp. 237–244, 2001.

[71] I. Gertsbakh and I. B. Gertsbakh, *Reliability theory: with applications to preventive maintenance*. Springer Science & Business Media, 2000.

[72] J. Wan *et al.*, "A manufacturing big data solution for active preventive maintenance," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2039–2047, 2017.

[73] J. H. Williams, A. Davies, and P. R. Drake, *Condition-based maintenance and machine diagnostics*. Springer Science & Business Media, 1994.

[74] R. Ahmad and S. Kamaruddin, "An overview of time-based and condition-based maintenance in industrial application," *Computers & industrial engineering*, vol. 63, no. 1, pp. 135–149, 2012.

[75] K.-A. Nguyen, P. Do, and A. Grall, "Multi-level predictive maintenance for multi-component systems," *Reliability engineering & system safety*, vol. 144, pp. 83–94, 2015.

[76] J. Wang, L. Zhang, L. Duan, and R. X. Gao, "A new paradigm of cloud-based predictive maintenance for intelligent manufacturing," *Journal of Intelligent Manufacturing*, vol. 28, no. 5, pp. 1125–1137, 2017.

[77] H. Löfsten, "Measuring maintenance performance–in search for a maintenance productivity index," *International Journal of Production Economics*, vol. 63, no. 1, pp. 47–58, 2000.

[78] T. A. Feo and J. F. Bard, "Flight scheduling and maintenance base planning," *Management Science*, vol. 35, no. 12, pp. 1415–1432, 1989.

[79] M. Biró, I. Simon, and C. Tánczos, "Aircraft and maintenance scheduling support, mathematical insights and a proposed interactive system," *Journal of Advanced Transportation*, vol. 26, no. 2, pp. 121–130, 1992.

[80] C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi, "The fleet assignment problem: Solving a large-scale integer program," *Mathematical Programming*, vol. 70, no. 1, pp. 211–232, 1995.

[81] R. Dekker and P. A. Scarf, "On the impact of optimisation models in maintenance decision making: The state of the art," *Reliability Engineering & System Safety*, vol. 60, no. 2, pp. 111–119, 1998.

[82] C. Sriram and A. Haghani, "An optimization model for aircraft maintenance scheduling and re-assignment," *Transportation Research Part A: Policy and Practice*, vol. 37, no. 1, pp. 29–48, 2003.

[83] P. Y. Cho, "Optimal scheduling of fighter aircraft maintenance," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[84] A. Gavranis and G. Kozanidis, "An exact solution algorithm for maximizing the fleet availability of a unit of aircraft subject to flight and maintenance requirements," *European Journal of Operational Research*, vol. 242, no. 2, pp. 631–643, 2015.

[85] Y. Liu, T. Wang, H. Zhang, V. Cheutet, and G. Shen, "The design and simulation of an autonomous system for aircraft maintenance scheduling," *Computers & Industrial Engineering*, vol. 137, p. 106 041, 2019.

[86] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[87] M. Pinedo, "Stochastic scheduling with release dates and due dates," *Operations Research*, vol. 31, no. 3, pp. 559–572, 1983.

[88] Z. Li and M. Ierapetritou, "Process scheduling under uncertainty: Review and challenges," *Computers & Chemical Engineering*, vol. 32, no. 4-5, pp. 715–727, 2008.

[89] R. K. Chakrabortty, R. A. Sarker, and D. L. Essam, "Resource constrained project scheduling with uncertain activity durations," *Computers & Industrial Engineering*, vol. 112, pp. 537–550, 2017.

[90] X. Cai, X. Wu, and X. Zhou, "Stochastic scheduling on parallel machines to minimize discounted holding costs," *Journal of Scheduling*, vol. 12, no. 4, pp. 375–388, 2009.

[91] R. Ramírez-Velarde, A. Tchernykh, C. Barba-Jimenez, A. Hirales-Carbajal, and J. Nolazco-Flores, "Adaptive resource allocation with job runtime uncertainty," *Journal of Grid Computing*, vol. 15, no. 4, pp. 415–434, 2017.

[92] P. Donti, B. Amos, and J. Z. Kolter, "Task-based end-to-end model learning in stochastic optimization," in *Advances in Neural Information Processing Systems*, 2017, pp. 5484–5494.

[93] K. M. Sallam, R. K. Chakrabortty, and M. J. Ryan, "A reinforcement learning based multi-method approach for stochastic resource constrained project scheduling problems," *Expert Systems with Applications*, vol. 169, p. 114 479, 2021.

[94] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[95] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[96] 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*, ser. SIGCOMM '19, 2019, pp. 270–288.

[97] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, 2020.

[98] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, vol. 29, no. 1, pp. 185–195, 2017.

[99] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[100] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *International Conference on Machine Learning*, 2018, pp. 344–353.

[101] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," *arXiv preprint arXiv:2002.05120*, 2020.

[102] A. Lodi and G. Zarpellon, "On learning and branching: A survey," *Top*, vol. 25, no. 2, pp. 207–236, 2017.

[103] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[104] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[105] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations*, 2019.

[106] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[107] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[108] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[109] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[110] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[111] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[112] X. Wang *et al.*, "Heterogeneous graph attention network," in *The World Wide Web Conference*, ACM, 2019, pp. 2022–2032.

[113] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 793–803.

[114] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proceedings of The Web Conference 2020*, 2020, pp. 2331–2341.

[115] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 2077–2085.

[116] E. Seraj, Z. Wang, R. Paleja, M. Sklar, A. Patel, and M. Gombolay, "Heterogeneous graph attention networks for learning diverse communication," *arXiv preprint arXiv:2108.09568*, 2021.

[117] E. Seraj *et al.*, "Learning efficient diverse communication for cooperative heterogeneous teaming," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1173–1182.

[118] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *CoRR*, vol. abs/1808.03314, 2018. arXiv: 1808.03314.

[119] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[120] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *International conference on artificial neural networks*, Springer, 2005, pp. 799–804.

[121] A. Bérard, O. Pietquin, C. Servan, and L. Besacier, "Listen and translate: A proof of concept for end-to-end speech-to-text translation," *arXiv preprint arXiv:1612.01744*, 2016.

[122] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[123] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[124] A. Ycart and E. Benetos, "A study on lstm networks for polyphonic music sequence modelling," in *ISMIR*, 2017.

[125] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[126] Z. Lu, W. Lv, Z. Xie, B. Du, and R. Huang, "Leveraging graph neural network with lstm for traffic speed prediction," in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2019, pp. 74–81.

[127] C. Si, W. Chen, W. Wang, L. Wang, and T. Tan, "An attention enhanced graph convolutional LSTM network for skeleton-based action recognition," *CoRR*, vol. abs/1902.09130, 2019. arXiv: 1902.09130.

[128] N. Sesti, J. J. G. Luis, E. F. Crawley, and B. G. Cameron, "Integrating lstms and gnns for COVID-19 forecasting," *CoRR*, vol. abs/2108.10052, 2021. arXiv: 2108.10052.

[129] 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.

[130] 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.

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

[132] 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.

[133] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

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

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

[136] S. Wilson *et al.*, "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.

[137] G. Tang and P. Webb, "Human-robot shared workspace in aerospace factories," *Human-robot interaction: safety, standardization, and benchmarking*, pp. 71–80, 2019.

[138] I. Essafi, Y. Mati, and S. Dauzère-Pérès, "A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 8, pp. 2599–2616, 2008.

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

[140] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 1–13, 1977.

[141] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[142] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," in *Robotics: Science and System*, 2013, pp. 49–56.

[143] Z. Wang and M. Gombolay, "Learning to dynamically coordinate multi-robot teams in graph attention networks," *arXiv preprint arXiv:1912.02059*, 2019.

[144] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[145] M. Wang *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[146] M. C. Gombolay, R. A. Gutierrez, S. G. Clarke, G. F. Sturla, and J. A. Shah, "Decision-making authority, team efficiency and human worker satisfaction in mixed human–robot teams," *Autonomous Robots*, vol. 39, no. 3, pp. 293–312, 2015.

[147] K. Kreeger, "The learning curve," *Nature Biotechnology*, vol. 21, no. 8, pp. 951–952, 2003.

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

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

[150] R. S. Sutton, S. Singh, and D. McAllester, "Comparing policy-gradient algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, 2000.

[151] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019.

[152] M. Wang *et al.*, *Deep graph library: A graph-centric, highly-performant package for graph neural networks*, 2020. arXiv: 1909.01315 [cs.LG].

[153] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[154] ATA, *ATA MSG-3 Operator/Manufacturer Scheduled Maintenance Development*. Air Transport Association of America Inc., 2007.

[155] M. Gombolay *et al.*, "Robotic assistance in the coordination of patient care," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1300–1316, 2018.

[156] C. Stenström, P. Norrbin, A. Parida, and U. Kumar, "Preventive and corrective maintenance–cost comparison and cost–benefit analysis," *Structure and Infrastructure Engineering*, vol. 12, no. 5, pp. 603–617, 2016.

[157] R. Yam, P. Tse, L. Li, and P. Tu, "Intelligent predictive decision support system for condition-based maintenance," *The International Journal of Advanced Manufacturing Technology*, vol. 17, no. 5, pp. 383–391, 2001.

[158] F. F. Oberweger, G. R. Raidl, E. Rönnberg, and M. Huber, "A learning large neighborhood search for the staff rerostering problem," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2022, pp. 300–317.

[159] M. Gombolay, T. Golen, N. Shah, and J. Shah, "Queueing theoretic analysis of labor and delivery," *Health Care Management Science*, vol. 22, no. 1, pp. 16–33, 2019.