Stochastic Resource Optimization over Heterogeneous Graph Neural Networks for Failure-Predictive Maintenance Scheduling

Zheyuan Wang, Matthew Gombolay

Georgia Institute of Technology, Atlanta, GA
pjohnwang@gatech.edu, matthew.gombolay@cc.gatech.edu

Abstract

Resource optimization for predictive maintenance is a challenging computational problem that requires inferring and reasoning over stochastic failure models and dynamically allocating repair resources. Predictive maintenance scheduling is typically performed with a combination of ad hoc, hand-crafted heuristics with manual scheduling corrections by human domain experts, which is a labor-intensive process that is hard to scale. In this paper, we develop an innovative heterogeneous graph neural network to automatically learn an end-to-end resource scheduling policy. Our approach is fully graph-based with the addition of state summary and decision value nodes that provides a computationally lightweight and nonparametric means to perform dynamic scheduling. We augment our policy optimization procedure to enable robust learning in highly stochastic environments for which typical actor-critic reinforcement learning methods are ill-suited. In consultation with aerospace industry partners, we develop a virtual predictive-maintenance environment for a heterogeneous fleet of aircraft, called AirME. Our approach sets a new state-of-the-art by outperforming conventional, hand-crafted heuristics and baseline learning methods across problem sizes and various objective functions.

1 Introduction

Resource optimization plays an important role in many real-world domains, including health care, manufacturing and services industries, and more (Zhou et al. 2021). Complicating the allocation and sequencing of workers and tasks in such 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 (Shah and Williams 2008). One of such case is aircraft maintenance scheduling, in which the inter-arrival times of part failures and the resulting service times are latent random variables (Dinis, Barbosa-Póvoa, and Teixeira 2019). 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 (Bajestani and Beck 2011). 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 (Haloui, Ponzoni Carvalho Chanel, and Haït 2020). Research suggests that predictive maintenance could reduce unscheduled work up to 33% (The Economist 2019), 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.

Recent advances in artificial intelligence (AI) have fostered the idea of leveraging deep neural networks (DNNs) to solve a plethora of problems in operations research (Bengio, Lodi, and Prouvost 2021). DNNs can be trained to automatically explore the problem structure and discover useful representations in high-dimensional data towards constructing high-quality solutions (LeCun, Bengio, and Hinton 2015) without hand-crafted feature engineering, which is required by typical operation research approaches (e.g., Genetic algorihtms, Branch-and-Bound solvers). Recent progress has been made in learning scalable solvers with graph neural networks (Khalil et al. 2017; Kool, van Hoof, and Welling 2019). However, these approaches typically consider only static, deterministic setting which limits applicability.

To overcome the limitations of prior work, 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 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 (IATA 2012) and 11% of revenue allocated for maintenance (Scott McCartney 2012).

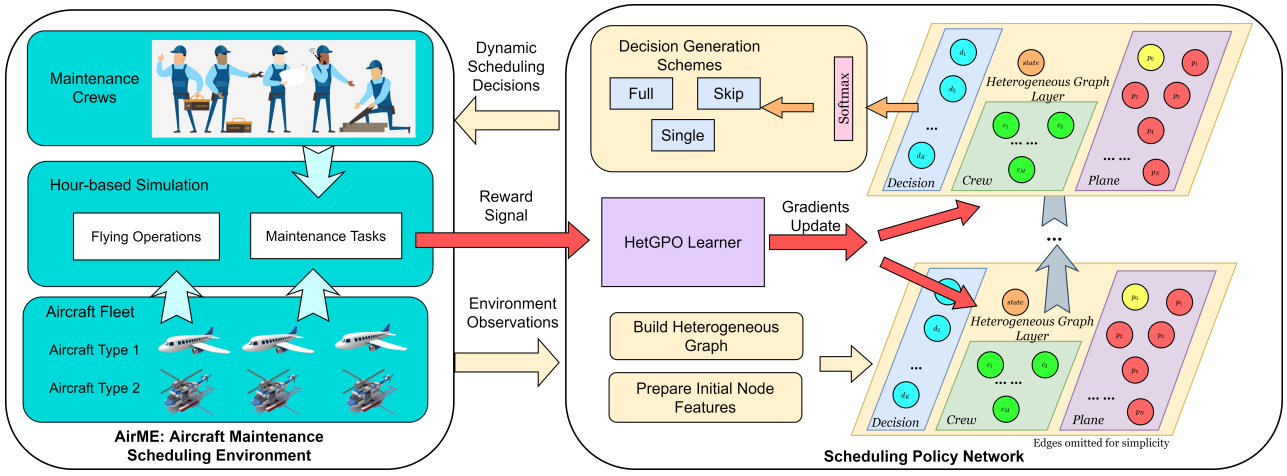

Figure 1: The figure depicts AirME, a virtual predictive-maintenance scheduling environment (Left), and our proposed scheduling policy network (Right). Left: AirME consisits of a team of maintenance crews and a heterogeneous fleet of aircraft and operates under hour-based simulation. Right: The scheduling policy network uses several heterogeneous graph layers to extract high level embeddings from the graph representation 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.

2 Related Work

Aircraft maintenance is performed to prevent or reduce the adverse effect of failures and to maximize aircraft availability at a minimum cost, and has been a popular application area for operations research studies. Dekker and Scarf (1998) stated that the problem underlying aircraft maintenance scheduling is a job scheduling problem on parallel machines with temporal and availability constraints. Gavranis and Kozanidis (2015) proposed an exact solution method to maximize feet availability by deciding which aircraft to assign to each fight under deterministic setting. Liu et al. (2019) designed and implemented an autonomous system that fuses aircraft's condition, strategy, planning and cost to improve the operational support for aircraft maintenance scheduling. A detailed overview on papers addressing the aircraft maintenance scheduling problem through operations research techniques can be found in Dinis, Barbosa-Póvoa, and Teixeira (2019).

Graph neural networks (GNNs), extending DNNs to learn from graph-structured data, were introduced by Scarselli et al. (2008). Research in GNNs either work with a spectral representation of the graphs directly (Bruna et al. 2014), or define convolutions on the graph and operate on groups of spatially close neighbors (Hamilton, Ying, and Leskovec 2017). Graph Attention Networks (GATs) (Veličković et al. 2018) were proposed to learn the importance between nodes and its neighbors and fuse the neighbors by normalized attention coefficients. Recently, researchers have proposed heterogeneous GNNs, allowing for learning with different types of nodes and edges, showing superior performance and model expressiveness (Wang et al. 2019; Seraj et al. 2021).

Leveraging these advances in deep learning, researchers have turned their attention to tackling a variety of problems in operations research (Bengio, Lodi, and Prouvost 2021). The fact that many combinatorial optimization problems (e.g., Minimum Vertex Cover, Travelling Salesman Problem, and Vehicle Routing Problem) can be expressed in graphs makes GNNs a top choice for representation learning in such scenario. Specifically, GNNs have shown success in addressing resource optimization and scheduling problems, such as scheduling data processing clusters (Mao et al. 2019), multi-robot systems (Wang and Gombolay 2020), wireless networks (Eisen and Ribeiro 2020), and more (Ma et al. 2020). However, most of these works relay on homogeneous or bipartite graphs to learn a universal message passing scheme, with limited expressiveness and generalizability. While Wang, Liu, and Gombolay (2022) combined heterogeneous graph networks with imitation learning, the policy requires known task information beforehand, operates in a deterministic environment, and requires examples of optimal schedules from an oracle for training. We are the first to

utilize a heterogeneous graph neural networks-based policy for stochastic scheduling scenarios.

3 Problem Overview

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.

3.1 Aircraft Maintenance 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 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[2] will be made publicly available upon paper publication.

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 discrete 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.

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 (ATA 2007), 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 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.

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

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, Het-GPO must implicitly learn a representation of this process in order to inform its decision-making policy.

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.

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 (Cho 2011).

3.2 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.
- 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.

[2] Attached as supplementary material

- 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.

4 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.

4.1 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 2.

$$p_\theta(u_t|o_t) = \prod_{i=1}^{n} p_\theta(d_i|o_t, d_{1:i-1}) \tag{2}$$

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 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.

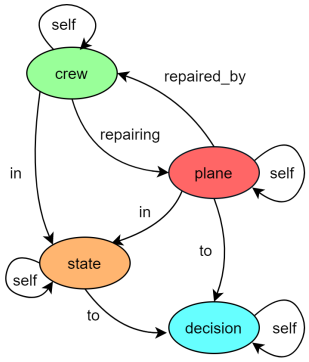

Figure 2: Metagraph of the heterogeneous graph built given an environment state in AirME.

4.2 Heterogeneous Graph Representation

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 nods (from source node to destination node). 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 number 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 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 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.

4.3 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 equations 3-6, 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.

$$\text{Plane } \vec{h}'_p = \sigma\Big(W_{plane,self}\vec{h}_p + W_{repairing}\vec{h}_c\Big) \quad (3)$$

$$\text{Crew } \vec{h}'_c = \sigma\Big(W_{repaired\_by}\vec{h}_p + W_{crew,self}\vec{h}_c\Big) \quad (4)$$

$$\text{State } \vec{h}'_s = \sigma\Big(\sum_{p \in N_{plane,in}(s)} \alpha_{s,p}^{plane,in} W_{plane,in}\vec{h}_p + \sum_{c \in N_{crew,in}(s)} \alpha_{s,c}^{crew,in} W_{crew,in}\vec{h}_c + W_{state,self}\vec{h}_s\Big) \quad (5)$$

$$\text{Decision } \vec{h}'_d = \sigma\Big(W_{plane,to}\vec{h}_p + W_{state,to}\vec{h}_s + W_{decision,self}\vec{h}_d\Big) \quad (6)$$

When computing output features of state summary node using equation 5, we implement attention mechanisms adapted from (Veličković et al. 2018) 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 7, 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.

$$\alpha_{s,i}^{edgeName} = \text{softmax}_i\Big(\sigma'\Big(\vec{a}_{edgeName}^T \Big[W_{state,self}\vec{h}_s || W_{edgeName}\vec{h}_i\Big]\Big)\Big) \quad (7)$$

To stabilize the learning process of self-attention, we utilize the multi-head mechanism that has been shown beneficial in homogeneous graphs (Veličković et al. 2018), 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, we construct the scheduling policy network that utilizes multi-layer structure to extract high-level embeddings of each node as shown in Figure 1.

5 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) (Schulman et al. 2017) and make several adaptations for better variance reduction. In particular, we optimize the clipped surrogate objective shown in Equation 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.

$$L(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t] \quad (8)$$

In Equation 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 (Mao et al. 2019) as a more accessible and efficient alternative. Specifically, during gradient estimation, the baseline value being subtracted

Algorithm 1: 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

1 Initialize policy network parameters θ_0
2 **for** $t = 1$ *to* K **do**
3 Sample episode length $T \sim$ Uniform$(T_{min}, T_{min} + T_{range})$
4 Sample a random environment instance, AirME_k
5 **for** $i = 1$ *to* N **do**
6 Reset AirME_k to $t = 0$
7 **for** $t = 0$ *to* $T - 1$ **do**
8 Get observation o_t^i from AirME_k
9 Build heterogeneous graph and input node features
10 Sample $u_t = \{d_1, d_2, ..., d_n\}$ from π_θ
11 Step through AirME_k and get intermediate reward r_t^i
12 Store $\{(o_t^i, u_t^i, r_t^i)\}$ to trajectory buffer
13 **if** *broken planes* $\geq 80\%$ **then**
14 Terminate current episode and zero-pad future rewards
15 **end**
16 **end**
17 **end**
18 Compute rewards-to-go: $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^i$
19 Compute advantage estimates: $A_t^i = R_t^i - \frac{1}{N} \sum_{i'=1}^{N} R_t^{i'}$
20 **for** $j = 1$ *to* N_{iter} **do**
21 Compute the clipped surrogate objective $L(\theta)$ using Equation 7
22 Perform stochastic gradient ascent for $\nabla L(\theta)$ using Adam optimizer with learning rate η
23 **end**
24 **if** $T_{min} \leq T_{max}$ **then**
25 $T_{min} = T_{min} + T_{step}$
26 **end**
27 **end**

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 1. Lines 3-17 details the process of rollout data collection on AirME instances generated on-the-fly. Here, we use a curriculum-based procedure in which initial training episodes are shorter, and the duration of episodes gradually increases, to avoid ineffective learning at the beginning. 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. At the end of each training episode, the range for sampling episode length is increased.

6 Experimental results

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

6.1 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 We employ the following heuristics: Corrective Scheduler (Stenström et al. 2016); Condition-based Scheduler (Yam et al. 2001); Periodic Scheduler (Ahmad and Kamaruddin 2012); and a random scheduler baseline.

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 and plan according to the actual failure probability.

Deep Learning-based Methods We compare against DeepRM (Mao et al. 2016) and Decima (Mao et al. 2019).

6.2 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 3.1, 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 using PyTorch (Paszke et al. 2019) and Deep Graph Library (Wang et al. 2020). 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

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 1: Evaluation results on O1: profit.

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 2: 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.

generated on-the-fly. In Algorithm 1, 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 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.

6.3 Evaluation Results

We present the evaluation results under three objectives in Tables 1-3, 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 random scheduler. This shows that the fix-sized 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

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**

Table 3: Evaluation results on O3: fleet availability.

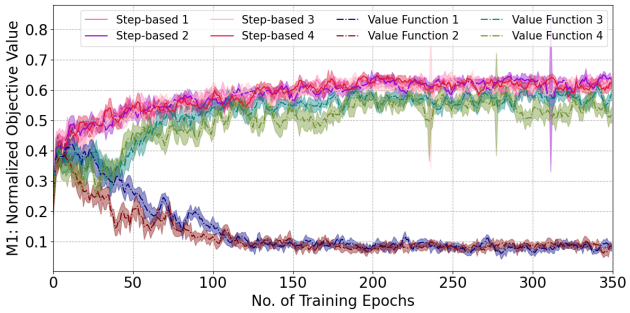

Figure 3: HetGPO-Single training on **O1** with a step-based baseline vs. state-based value function. Numbers in the legend denote the random seeds used.

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.

6.4 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 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 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.

7 Conclusion

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.

References

Ahmad, R.; and Kamaruddin, S. 2012. An overview of time-based and condition-based maintenance in industrial application. *Computers & industrial engineering*, 63(1): 135–149.

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

Bajestani, M. A.; and Beck, J. C. 2011. Scheduling an aircraft repair shop. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

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

Cho, P. Y. 2011. *Optimal scheduling of fighter aircraft maintenance*. Ph.D. thesis, Massachusetts Institute of Technology.

Dekker, R.; and Scarf, P. A. 1998. On the impact of optimisation models in maintenance decision making: the state of the art. *Reliability Engineering & System Safety*, 60(2): 111–119.

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

Eisen, M.; and Ribeiro, A. 2020. Optimal wireless resource allocation with random edge graph neural networks. *IEEE Transactions on Signal Processing*, 68: 2977–2991.

Gavranis, A.; and Kozanidis, G. 2015. 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*, 242(2): 631–643.

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

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

IATA. 2012. *International Air Transport Association Annual Report 2012*.

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

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*, 521(7553): 436–444.

Liu, Y.; Wang, T.; Zhang, H.; Cheutet, V.; and Shen, G. 2019. The design and simulation of an autonomous system for aircraft maintenance scheduling. *Computers & Industrial Engineering*, 137: 106041.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04): 5077–5084.

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

Mao, H.; Schwarzkopf, M.; Venkatakrishnan, S. B.; Meng, Z.; and Alizadeh, M. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, 270–288.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Scott McCartney. 2012. How airlines spend your airfare. Accessed: 2021-12-15.

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

Shah, J. A.; and Williams, B. C. 2008. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *ICAPS*, 322–329.

Stenström, C.; Norrbin, P.; Parida, A.; and Kumar, U. 2016. Preventive and corrective maintenance–cost comparison and cost–benefit analysis. *Structure and Infrastructure Engineering*, 12(5): 603–617.

The Economist. 2019. Artificial intelligence is changing every aspect of war. Accessed: 2021-12-15.

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

Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; Xiao, T.; He, T.; Karypis, G.; Li, J.; and Zhang, Z. 2020. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. arXiv:1909.01315.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference*, 2022–2032. ACM.

Wang, Z.; and Gombolay, M. 2020. Learning scheduling policies for multi-robot coordination with graph attention networks. *IEEE Robotics and Automation Letters*, 5(3): 4509–4516.

Wang, Z.; Liu, C.; and Gombolay, M. 2022. Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints. *Autonomous Robots*, 46(1): 249–268.

Yam, R.; Tse, P.; Li, L.; and Tu, P. 2001. Intelligent predictive decision support system for condition-based maintenance. *The International Journal of Advanced Manufacturing Technology*, 17(5): 383–391.

Zhou, B.; Bao, J.; Li, J.; Lu, Y.; Liu, T.; and Zhang, Q. 2021. A novel knowledge graph-based optimization approach for resource allocation in discrete manufacturing workshops. *Robotics and Computer-Integrated Manufacturing*, 71: 102160.

Stochastic Resource Optimization over Heterogeneous Graph Neural Networks for Failure-Predictive Maintenance Scheduling Supplementary Materials

1 AirME: Parameters and Additional Details

1.1 Aircraft Failure Model

Table 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 expressiveness. 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.

1.2 Additional Details

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.

2 Details of Baseline Methods

2.1 Heuristics Baselines

Random Scheduler At time t, the random baseline assigns each available crew a plane to start maintance 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 (Stenström et al., 2016) 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 (Yam et al., 2001) 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 anther priority queue (e.g., based the flight hours or number of landings) and assigns the rest of crew for conducting CBMs 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 (Ahmad and Kamaruddin, 2012) 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 performance 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.

2.2 Model-based Scheduler

To construct a model-based scheduler, we augment the condition-based scheduler by giving it access to the plane

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

Table 1: Hyper-parameters of Plane Failure Models

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, $\beta_p = 0.004$.

2.3 Learning-based Schedulers

To include RL-based baselines, we consider two methods in prior works for resource scheduling and adapt them to AirME.

DeepRM (Mao et al., 2016) DeepRM represents the current allocation of resources as fix-sized 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 (Mao et al., 2019) 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 S1.

$$h_s = g\big(\sum_{m \in N(s)} f(h_m)\big), \tag{S1}$$

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 S2.

$$p(m) = \frac{\exp(q_m)}{\sum_{m' \in M} \exp(q_{m'})}, \tag{S2}$$

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.