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

AirME: Parameters and Additional Details

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.

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.

Details of Baseline Methods

Heuristics Baselines

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 (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 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 (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.

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, $\beta_p = 0.004$.

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($\cdot$) and f($\cdot$) 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($\cdot$), f($\cdot$) and q($\cdot$) 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.

Generalizability of HetGPO on Stochastic Resource Optimization Domains

Our research aims to develop a scalable, non-parametric RL method for dynamic scheduling in stochastic resource optimization environments. While the experimental results are obtained in AirME, our HetGPO framework is 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.

Both the "state summary" and "decision value" nodes, and the HetGPO training process are developed with the mindset of a general graph-based policy learning algorithm to solve a broader class of problems. Next, we demonstrate how to use HetGPO on a similar domain: dynamic patient admission scheduling in health care.

Patient Admission Scheduling The patient admission scheduling (PAS) problem consists of assigning patients to hospital resources (beds, rooms, nurses, etc) in such a way as to maximize medical treatment effectiveness, management efficiency, and patient comfort, while respecting the capacity constraints and/or the gender policy (Gombolay et al. 2018). When all admission times and length-of-stay are known in advance, the problem can be solved just once for the whole planning period. However, solving with this static setting has little utility for most practical cases where patients may arrive at unknown times and the treatment duration is also stochastic. Such real world scenario can be formulated as a class of predictive scheduling under uncertainty, in which the schedule is adjusted in response to real-time events and HetGPO is suitable to be deployed to learn robust scheduling policies.

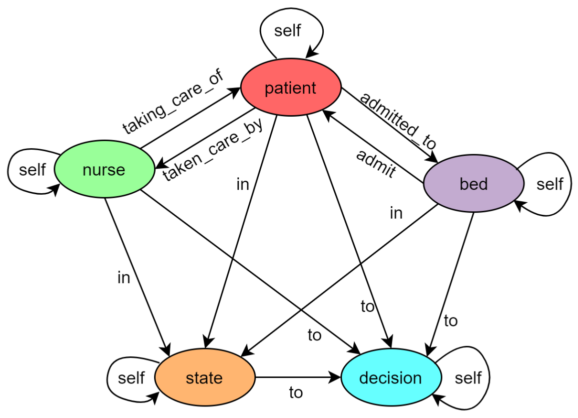

Figure S1: Metagraph of the heterogeneous graph built for patient addmission scheduling problems.

Take patient admission for the delivery room 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 S1. 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 1 can be used to learn scheduling policies under the objective functions defined for hospital scenarios.

Evaluation results with AirME instances showed the superiority of HetGPO across problem sizes and objective functions over popular heuristics and learning-based methods. Therefore, we expect HetGPO to continue to work well on similar stochastic resource optimization domains that require predictive scheduling efforts. In future work, we plan to develop our approach for such a healthcare application.