Model Comparative Analysis of Neighborhood Aggregation Levels in Graph Neural Networks and Traditional Neural Networks for Metaschedulers

Samer Alshaer, Murali Piridi, Roman Obermaisser
Department of Embedded Systems
University of Siegen
Siegen, Germany

***Abstract*—In Time-Triggered Systems (TTS), the adaptation to various scenarios, including hardware failures, mode changes, and the accommodation of slack times, is facilitated by meta-scheduling algorithms. Nevertheless, significant storage demands on Cyber-Physical Systems (CPS) often arise due to the State-space explosion problem associated with handling multiple scenarios. This work builds upon prior research conducted in [1] [2], where Graph Neural Networks (GNNs) were examined in comparison to a list scheduler (LS) within constrained systems. In this extension, a deeper exploration of the application of GNNs in meta-scheduling is undertaken, with a particular focus on multi-neighbourhood aggregations and their influence on makespan results in process scheduling. Furthermore, a comprehensive comparative analysis of makespans is conducted between the Genetic Algorithm (GA) employed for generating scheduling scenarios, our newly proposed GNN-based approach, and traditional Artificial Neural Networks (ANNs) utilized in similar applications such as Random Forest classifiers (RFC), Artificial Neural Networks (ANN), and Encoder Decoder Neural Networks (E/D NN). Notably, the results highlight the significance of aiming for second neighbourhood aggregation as the optimal complexity/performance compromise in process scheduling for event-driven Multi-Schedule Graphs (MSG). And in comparison to different algorithms, GNNs provides one of the best machine learning solutions to the meta-scheduling problem in terms of makespan to model parameter cost of the inference.**

***Keywords*—Metascheduling, Process Scheduling, Graph Neural Networks, Graph Attention Networks, Multi-Neighborhood Aggregation, State-space explosion, Genetic Algorithm.**

I. Introduction

The purpose of meta-scheduling in distributed system architecture is to compute multiple schedules, each of which is mapped to a specific scenario. The scenarios can include slack time, failures, and different operating modes, collectively referred to in this work as context events. A dispatcher is equipped with the ability to switch schedules upon the occurrence of context events. Moreover, the schedules are pre-computed offline while considering these context events. The concept of meta-scheduling often appears in literature as quasi-static scheduling [3], or as super schedulers [4]. Meta-scheduling is of particular importance for safety-criticality systems where adaptation services are required. These services aim to increase system dependability attributes such as reliability and availability. Although static scheduling techniques are predictable, they are limited to specifications of dataflow without choice [5]. In contrast, meta-schedulers extend static schedulers with data-dependent choices at run-time. Sorkhpour [6] created a scenario-based meta-scheduling tool for an Adaptive Time-Triggered Multi-core Architecture that took into account context events like core failures and slack events. At design time, Mixed-Integer Quadratic Programming (MIQP) was employed to tackle scheduling challenges. The end product was the Multi-Schedule Graph (MSG), which is used at run-time. There was no solution provided to handle the volume of data generated by the MSG size, which creates a rising storage need for Cyber Physical System (CPS). Muoka et al [7] presented a path reconvergence and reconvergence horizon based technique to solve the state-space explosion problem caused by a rise in the number of scenarios necessary for adaptation. However, the approach only regulates the size of the MSG that will be deployed, which is a trade-off because the intended number of context events is restricted to a predefined horizon.

This paper extends previous work in to solve MSG state space explosion utilizing machine-learning based approaches [1] [2] by performing deeper exploration of the schedules varying the number of neighbourhood aggregation and comprehensively comparing to previous techniques used in literature for the exact same application as Graph Neural Networks (GNN) have proved helpful in handling graph applications [8] [9]. The strength of GNN's is attributable to their ability to capture node associations via their graph characteristics and placements. While this paper presents the GNN for learning the MSG, it also tackles problems about ensuring accurate temporal and spatial task allocation in safety-critical applications. It overcomes these problems by solely employing the GNN to forecast the temporal priority of jobs, which is subsequently utilised to do live schedule reconstruction. Furthermore, we investigate the influence of Neighborhood Aggregation levels in GNN on makespans of generated schedules and comparing it to previous work in literature such as: Random Forest Classification (RFC), Artificial Neural Networks (ANN), and Encoder/Decoder Neural Networks (E/D NN). The remainder of this work is structured as follows: Section II discusses the related works. Section III provides a background of the meta-scheduler and explains the proposed model. An experimental setup to evaluate our work is presented in section IV, while

the results are discussed in section V. Finally, the conclusion is discussed in Section VI.

II. RELATED WORKS

A. Related works on metascheduling

In prior work we proposed a meta-scheduling algorithm where a scheduler is invoked repeatedly to generate schedules for different scenarios. The proposed algorithm is not limited to a specific scheduler. For example, it can utilise schedulers that provide solutions based on Genetic Algorithms (GA), Mixed Integer Linear Programming (MILP), and Ant Colony Optimisation (ACO).

Babak in [10] proposed an optimisation method that utilises Mixed-Integer Quadratic Programming (MIQP) equations with which minimum consumption of energy for slack events is established. This work made use of the meta-scheduling algorithm that is proposed in [11], but the state explosion issue was not covered.

Muoka et al. in [7] implemented two algorithms as an attempt to solve this state explosion problem, one of them is a path reconvergence algorithm and the other being reconvergence horizon algorithm. The path reconvergence algorithm focuses on the adaptation, it establishes certain boundaries for the meta-scheduler in order to add schedules to the list of schedules. On the other hand the reconvergence horizon deals with eliminating the repeated schedules from the list of computed schedules. When implementing the reconvergence horizon in order to eliminate the need of firmly bounding the limits of adaptation, a solution is proposed int this work.

Alshaer et al. in [1] implemented the Message Passing Neural Network (MPNN) model from Pytorch Geometric (PyG) library to learn the scheduling mechanism of GA. In addition, a new design for the metascheduler that utilizes AI was proposed to facilitate run-time operation. A comparison is the made between the generated makespans for List Scheduler (LS), GA and GNN and is further investigated for the impact of modifiable jobs on the overall makespan during a multi-scheduling operation. The results showed that GNN models in concept prove to be an efficient scalable solution. Furthermore, work in [2] continued the investigation to study RFC, ANN, and E/D NN for the same application

H. Baniabdelghany et al. in [12], proposed a metascheduler for time triggered wireless sensor networks using discrete particle swarm optimization for reliable task allocation (DPSO-TA). This work particularly focuses on reducing the size of the generated MSG, as an attempt to tackle the state explosion issue, using a reconvergence algorithm. In contrary, this work focuses on AI inference using GNN's to solve the same issue.

B. AI-based Scheduling

Pirhoseinlo et al. in [13] proposed an AI framework based approach similar to this work, which utilises neural networks and a GA, the use of AI is incorporated to predict the end time of the jobs to reduce the required GA required conversion time to reach solutions. In contrast, our work produces schedules for an MSG using a meta-scheduling algorithm with help of GA. These schedules are then trained using a GNN model, to predict the temporal priorities of jobs.

Zang et al. in [14] introduced a Hybrid Deep Neural Network Scheduler (HDNNS) to solve the job-shop scheduling problem, this HDNNS architecture is a combination of a deep convolutional layer, a fully connected layer and a flatening layer. Also to convert the irregular scheduling information into regular data a Convolutional Two-dimensional Transformation (CTDT) is utilized conversely, in our research, the applicability of neural networks is explored where the data from a metascheduler is trained by a scheduler,which focuses more on the training and achieving high accuracies.

Melnik et al. in [15] introduced an innovative scheduling methodology that incorporates Artificial Neural Networks (ANN) and Reinforcement Learning (RL) in the context of a workflow system. The primary objective was to delve into the scheduling problem, utilizing RL to shape input states and consider the structural characteristics of the workflow. The approach, named Neural Networks Scheduling (NNS), is essentially rooted in RL, being trained to produce high-quality schedules in order to optimize the workflow makespan. Conversely, in our research, we employ a GNN algorithm for adaptive scheduling, which involves predicting task priorities to guide the subsequent schedule generation process. This priority prediction approach ensures the calculation of precise schedules while maintaining safety assurances. Scheduling in the realm of cloud computing attracts considerable attention from researchers and frequently leverages machine learning techniques.

Gondhi et al. [16], conducted an extensive examination of literature concerning scheduling in cloud computing. They highlighted the utilization of various advanced intelligent scheduling algorithms, some of which incorporate a swarm-based approach, a common practice within the cloud computing field. The primary objective of scheduling in cloud-based computing revolves around achieving load balancing, facilitating a scalable framework, and optimizing resource reallocation.

Zhao et al. in [17], introduced a graph convolutional network specifically tailored for training in scheduling operations within wireless networks. This study underscores the potential of Graph Neural Networks (GNNs) in addressing scheduling tasks. It's worth noting that their research did not encompass schedule adjustments prompted by adaptive changes. In contrast, our work places its emphasis on acquiring knowledge about the MSG to enable real-time adaptation.

Ma et al. in [18] developed an online planner selection system utilizing Graph Neural Networks (GNNs) in conjunction with adaptive scheduling. Their approach involves training the GNN with a swarm-based algorithm offline, and the GNN is employed to choose the most suitable planner dynamically during system operation. In contrast, our research focuses on a distinct application of GNNs, specifically predicting the temporal priorities of jobs rather than in the context of online planner selection.

III. METASCHEDULER DESIGN

A metascheduler is used in Time Triggered Systems (TTS), where different schedules must be generated based on failures, slack or mode change events in order for the system to be able to adapt to these contexts. When generating these schedules, a metascheduler takes the system's contextual, spatial and temporal priorities into consideration. The generated schedules are basically a Directed Acyclic Graph (DAG) of linked schedules which is termed as MSG in this paper. Nodes represent schedules and links between the nodes or edges are the contextual events. The meta-scheduler considers decision variables ensuring consistency between parent and child schedules which prevents system failure upon a schedule change.

An Adaptive Time Triggered Architecture is designed to fulfill these issues as proposed in article [11]. In such architectures, an aligned schedule switch is used to achieve adaptation and an offline metascheduler for adaptive scheduling in TTS is proposed in [7], to generate the MSG. The meta-scheduler takes the computational and communication activities into consideration. These are further classified into 3 models, a context model, an application model and a platform model, which are given as input to compute an MSG. **Context model**: This model holds information regarding the context events that are taken into consideration at design time. These context events include the description of failure scenarios, slack events, operation mode and environmental mode changes.

Platform model: This model contains the information regarding the computation and communication resources, such as cores, memories, routers and links between the routers.

Application model: This model is basically a DAG where the vertex represents a job and the edge represents a message. This describes the computation jobs with their deadlines, Worst-Case Execution Times (WCETs), message exchange between resources and reflects the precedence constraints between the jobs and messages.

Meta-scheduler: This component generates the required schedules in the form of JSON files by gathering information from the application, platform and context models, using schedule algorithms such as GA. Each computed schedule is related to a given context event considered, and the output of the meta-scheduler is the MSG.

Multi-Schedule Graph (MSG): This component is the output of the meta-scheduler. It is a DAG of the time-triggered schedules that is generated at design time. A node in the MSG graph is a schedule used during run-time, and the node selected is based on the corresponding context event.

Dataset: This component is the result of the mapping process and feature extraction from the JSON file solutions generated by the meta-scheduler. The node features such as: WCET time, processor allocation, jobs precedence, messages size, and routes in the network are extracted and stored in a data object. Besides the node features, the dataset stores the context events that causes the schedule to change. The listing below describes the composition of the context vector inserted into the GNN as the edge features. The key features define each schedule in the MSG. In addition, key features are also extracted from the context event:

- Context type ID: Determines which context event occurred (slack, fault, battery level, user-defined).
- Context value: In the case of a slack or a battery level event, this element indicates the percentage change.
- Task ID: In which task the context event took place.
- Context time: At what time of the schedule the event occurred.
- Device ID: Device that was affected by the context event, for instance a fault in a particular component.

The dataset data object contains context events stored as edge features, and also the temporal priorities that the GNN should learn. The final dataset comprises the task features and context event features as the inputs and the temporal priorities as the outputs that will help to reconstruct the schedule. The generation of the MSG is described in [7] using a GA to search for an optimized solution. The GA injects different contexts into a main schedule and calculates different solutions for the optimal schedule in each case creating a MSG. The generation of MSGs from the previous designs suffers from issues of state explosion where the size of the generated MSG increases exponentially and requires significant use of resources to adapt to various situational contexts. An alternative model that targets online operation of the metascheduler was developed in [1]. Figure 1 illustrates the proposed architecture of the meta-scheduler.

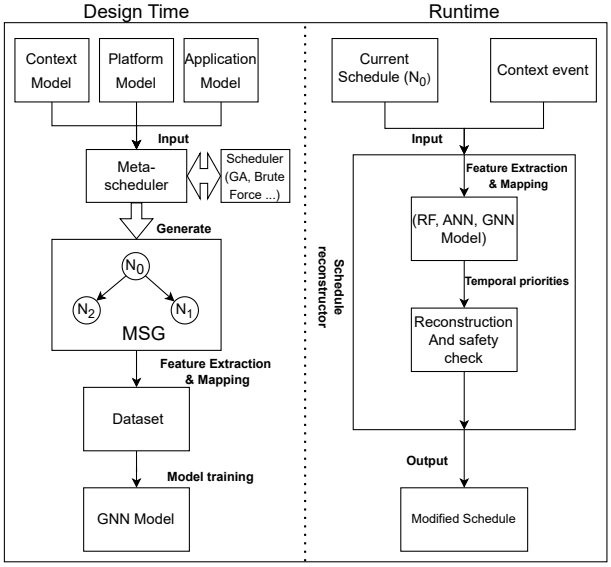

Figure 1: Proposed model of the meta-scheduler [1]

The meta-scheduler generates a MSG at design time, which is then used to train a GNN model predicting the temporal priorities of the next schedule. The GNN model is deployed at run-time as shown in Figure 1. The components of the proposed model at run-time are explained as follows:

GNN model: This component is the result of the model

training process. The GNN model is trained to predict the temporal priorities. After the temporal priorities of the valid schedule are generated they are sent to the reconstruction model.

Reconstruction and safety check: This component takes predicted temporal priorities and performs a schedule reconstruction. In addition, it performs safety checks to make sure precedence constrains are not violated by only allowing jobs that have their precedence constrains met in the first place to be allocated.

Modified schedule: The schedule is the output of the reconstruction block which consists of the schedule generated in response to the context event adaptation. During run-time whenever a context event occurs, the context event and the schedule itself go through a feature extraction process as shown in Figure 1 (run-time). The features are sent to the reconstruction model after the temporal priorities are predicted. The reconstructor then uses the generated temporal priorities and completes the spatial priorities of the schedule and performs a safety check to avoid message collisions.

IV. Experimental setup

The experiment in this paper studied multi-neibourhood aggregations in GNNs. This section explains the design steps of the experiment as follows:

- At the beginning, the GA was used to generate around 16,384 scenarios for various numbers of jobs (15, 30, 40, 60, and 100 jobs). Each scenario is represented by a schedule on its own and a context event. As previously explained, these scenarios are then converted into datasets of graph represented data to be used for training the GNN models.
- Four GNN models were designed for each job variation were they consider different numbers of neighbourhood aggregations. They consider the first neighbourhood, second neighbourhood, third neighbourhood and fourth neighbourhood in the graph data.
- Finally, these models are trained and evaluated for accuracy and model parameters. Then the results are used to be tested in the reconstruction algorithm to obtain the makespan.

A. Generating Data

The GA was used to create a MSG that schedules over a multi-core distributed memory network architecture that comprises of three routers (*R1 - R3*) and six processing elements *PE1 - PE6*. An application model is developed for each of the following job sizes: 15 jobs, 30 jobs, 40 jobs, 60 jobs, and 100 jobs. Using a single platform paradigm, the meta-scheduler method is deployed for various workloads. Different workloads result in a wide range of input feature sizes. Mainly because the features are connected to the number of jobs as each job has its worst-case execution time (WCET), precedence constrains and message transmission size. The data consisted of 16,384 samples.

B. Graph Attention Networks (GAT) Model Hyper Parameters

In the case of large graphs, like social networks or jobs involving Multi-Scheduled Graphs, understanding the relationships between neighboring nodes is essential. An indispensable tool in such scenarios is the "Attention mechanism," which uses stacked layers with nodes possessing features of neighboring nodes as displayed in Figure 2.

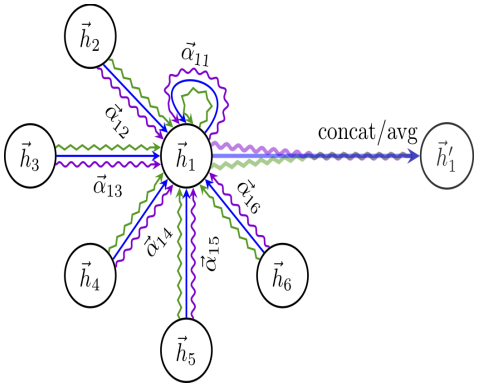

Figure 2: Multi-Head Attention [19]

When applying the attention mechanism to these nodes, the network assigns different weights to neighboring nodes, prioritizing more influential nodes during neighborhood aggregation. Now let's dive into the steps to achieve the attention mechanism these equations are based on [19], in which the authors utilized a particular attentional setup which is similar to the work of Bahdanau et al. (2016) in [20]

- Step 1: The first step is to apply the Linear Transformation $[W * X_i]$, which represents the schedule features mentioned previously.
- Step 2: The next step is to compute the un-normalized attention coefficients across the pair nodes i, j , based on their node features. This basically specifies the importance of the features of node j on node i.Here the co-relation between the schedules in the MSG is identified using these attention coefficients.

$$e_{ij} = a(W * \overrightarrow{X_i}, W * \overrightarrow{X_j}) \tag{1}$$

  a is the function that is determined by the user, subject to the restriction.
- Step 3: Now in order to have common scaling across all the neighborhoods, the attention coefficients should be normalized. We use either *Softmax* or *LeakyRelu* in order to achieve this.

$$\alpha_{ij} = \frac{\exp(LeakyRelu(e_{ij}))}{\sum_{k \in N} \exp(LeakyRelu(e_{ij})} \tag{2}$$

  N is the neighborhood of node i
- Step 4: Now the final output features are computed using $X_i^{'}$,in our case the output features are the attention coefficient vector. All the nodes (schedule features), edges (context events) and the attention coefficients are given together to the model as input and the final feature

vector,also known as attention coefficient vector for every schedule node is calculated.

$$\vec{X_i^{'}} = \sigma(\sum_{j \in N} \alpha_{ij} * W * X_i^{'}) \tag{3}$$

In Figure 2 the node feature vector is represented with $\vec{h_1^i}$, and the notation $\vec{X_1^i}$ is used.

- Step 5: Finally to stabilize the process of attention, the other neighborhood attentions are included to perform transformation and aggregation of output features this basically is called multi-head attention which is clearly shown in Figure 2 in which every neighborhood node i of node 1 sends its own vector of attention coefficients,i.e., α_{1i}, one per each attention head α_{1i}^k. The attention heads are used to compute K separate combinations of neighbouring node features $\vec{X^i}$. These are then aggregated through concatenation or taking average to get the next level features of $\vec{X_1^i}$.

$$\vec{X_i^{'}} = \sigma(\frac{1}{k} * \sum_{k=1}^{k} \sum_{j \in N} \alpha_{ij}^k W^k \vec{X_i^{'}}) \tag{4}$$

This step allows us to aggregate the information of neighborhood multiple times which is basically referred to number of heads. Here once the attention coefficient vectors of all the neighbouring schedules are calculated(in step 4), multiple combinations are made and aggregated using simple addition or mean to calculate the final output which is a temporal priority vector. k represents the number of independent attention maps used.

With the help of the attention mechanism, attention networks extract useful information from large data which makes the network more reliable.

The GAT model has been imported from Pytorch Geometric [21] Library and for training the MSG the optimized values for the hyper parameters are chosen after several observations, Learning rate = 0.0001, weight_decay = 0.0005, drop out probability is chosen only in first layer as '0.25' in order to not lose much information and at the same time preventing the model from over fitting. These values remain constant for training all the four neighborhood models, Whereas the complexity parameter is adjusted through out the process by manipulating the number of hidden channels thus increasing or decreasing the number of model parameters.The main agenda of training is to minimize the model complexity parameter to avoid the state explosion problem, which has been achieved in this work.

C. Prediction and Performance

The model outputs are the temporal priorities and they vary depending on the number of jobs. So if there are 30 jobs, then the priority for each job needs to be predicted. So the variation in model parameters not only stems from neighbourhood aggregations but also from the number of jobs. Each model is studied for its training accuracy on the 80% training dataset, and validation accuracy where 20% of the data was stored without training. Eventually, 5 models will be produced for each neighbourhood, for each of the data samples (15, 30, 40, 60, and 100 jobs). The temporal priorities generated by the GNN are fed into the reconstruction algorithm, which in turns generates makespans for each schedule.

Both training and validation accuracies are calculated through equation 5:

$$Acc = \frac{(Cp)}{(Cp + Fp)} * 100\% \tag{5}$$

Where, Acc represents the accuracy, Cp is the number of classes that are correctly predicted, and Fp is the number of classes that are falsly predicted.

V. Results

The outcomes of the contrasting GNN with conventional artificial intelligence algorithms for similar applications as reported in [1] and [2]. Specifically, the comparison involves GNN, RFC, ANN, and E/D NN. Since the experiments done in this paper extend to reach for 100 jobs while the research in [2], only investigated for 60 jobs, curve fitting techniques were used to extrapolate the remaining data by identifying the nature of increase/decrease in each graph and predicting the results reliably to further understand scalability for each AI models in the meta-scheduler. The results of the multi-neighborhood analysis are displayed in Figures 3, 4.

In Figure 3, the results for GAT models that take into consideration one neighbourhood aggregation (1-N) ... to four neighbourhood aggregations (4-N) show that (2-N) was the best solution since it can predict the temporal priorties more accurately than the rest of the models. Thus, (2-N) was chosen for generating makespans in this work. It can be observed that for the small jobs with 15 and 30 jobs 1st neighborhood aggregation is enough for the model to learn properly, but when going for jobs with 40, 60 and 100 it can be observed from Figure 3 that the model performs better with 2nd neighborhood aggregation and beyond 2nd neighborhood aggregation the model looses the information due to large amounts of information transferring between high number of available model parameters.

Figure 4 shows the number of model parameters required for attaining accuracies shown in Figure 3. It is also important to study the increase in the model parameters after considering increasing the number of multi-neighbourhood aggregations. It is noticed that while going for 3-N and 4-N, the number of model parameters is increasing although the accuracy dropped as shown previously in Figure 3. A slight increase is noticed going from 1-N to 2-N which also adds to the validity of choosing 2-N models in solving the meta-scheduling problem.

The accuracy of the 2-N GAT model is compared to accuracies of models designed for the same specific task in literature studied across multiple numbers of jobs in Figure 5. The figure shows the accuracy loss in data while complicating the scheduling task. GAT showed minimal to no loss in processing data which shows that the algorithm is more scalable in

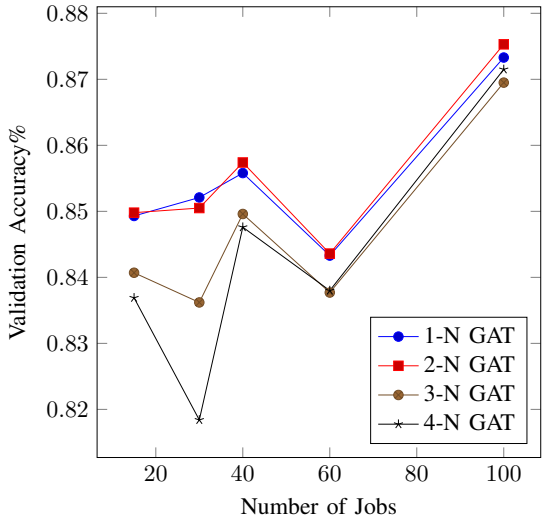

Figure 3: Results after GAT model training for different neighbourhood aggregations

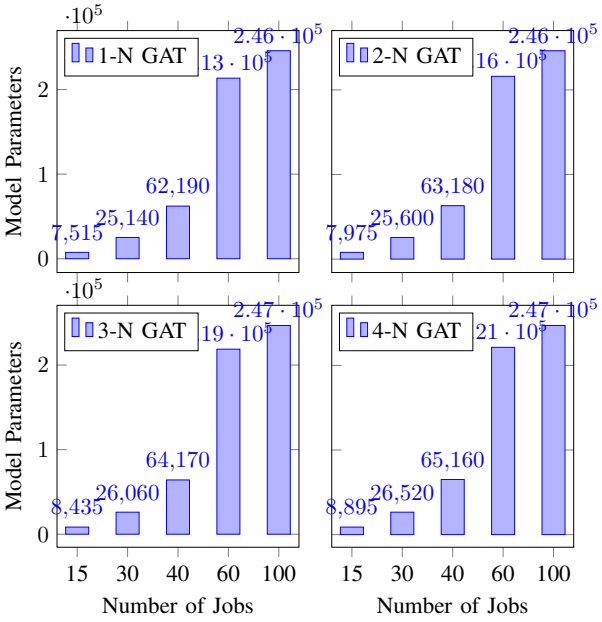

Figure 4: Difference in model parameters across all job datasamples for all Neighbourhood aggregations

comparison to the previous algorithms tested in literature. However, it is also important to understand the nature of the increase on model parameters as well. The more complicated the presented task the more parameters are used to process information and generate output. The more model parameters are used the more power is consumed and time spent in processing data.

Figure 6 shows the difference in model parameters generated from complicating the scheduling task by increasing the number of jobs. It is shows that GAT had a moderate amount of model parameters ranking better than 'E/D NN' and slightly worse than 'ANN'. However, it is worth mentioning

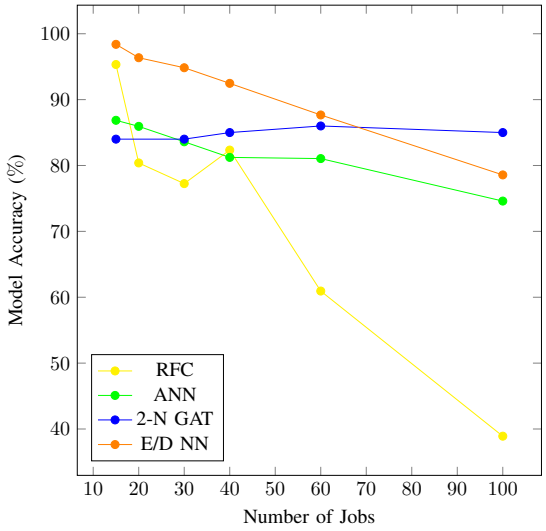

Figure 5: Accuracy comparison of AI models for the meta-scheduling problem

that the growth in the number of parameters becomes less after increasing the number of jobs from 60 to 100. This indicates a potential decrease in growth for increasing the number of jobs, which adds to the scalability factor of GNNs.

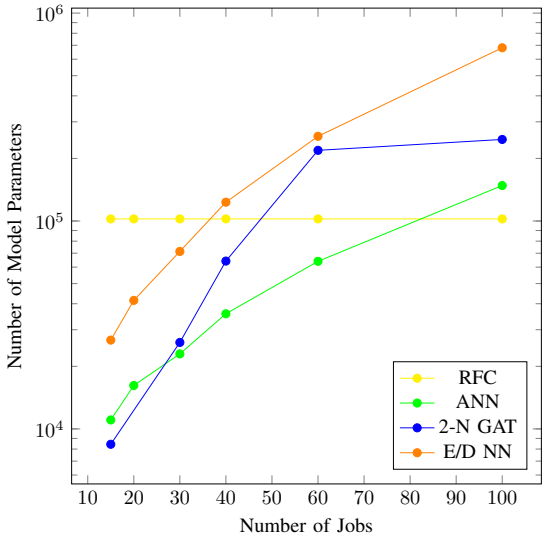

Figure 6: Complexity comparison of AI models for the meta-scheduling problem

Finally, the GNN model is run through the reconstruction algorithm to produce schedules that have specific makespans, and Figure 7 shows the results after running the reconstruction algorithm. It is noticed that the generated schedules are quite comparable to the quality of the schedules generated from the GA. However, it is expected for the GNN algorithm to behave slightly worse since the same schedules generated from the GA were used to train the model. Work in [22] shows the makespan results of running an ANN, and a LS algorithm through the reconstruction algorithm. Running the GAT GNN

algorithm through the reconstructor to generate makespans results in almost identical behaviour to that of the GA making it the top performer directly after the GA not to forget that it takes much less time to execute and does not suffer from state-space explosion.

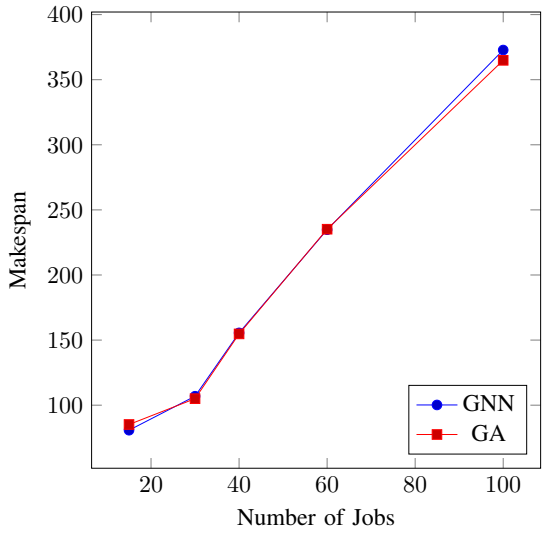

Figure 7: Makespan comparison between GA and GNN

VI. Conclusion and Future Work

In conclusion, this paper presented a detailed work of a metascheduler's inference design, GNN Neighborhood Aggregation levels analysis, and a Comparative study between GNN and Traditional Neural Networks used in literature for the same application. Results highlight the significance of aiming for second neighbourhood aggregation as the optimal complexity/performance compromise in process scheduling for MSGs. The results showed that GNN is quite feasible in comparison to other traditional neural networks in literature, offering best performance while keeping reasonable scalable model parameters.

VII. Acknowledgment

One of the authors; S.Alshaer would like to thank the German Academic Exchange Service (DAAD) for funding his Phd studies allowing his contribution to this paper. In addition, the authors would like to acknowledge the contributions of Uni siegen's Omni Cluster for the allowing the data generation process.

References

[1] S. Alshaer, C. Lua, P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Graph neural networks based meta-scheduling in adaptive time-triggered systems," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–6.

[2] D. Onwuchekwa, M. Dasandhi, S. Alshaer, and R. Obermaisser, "Evaluation of ai-based meta-scheduling approaches for adaptive time-triggered system," in *2023 International Conference on Smart Computing and Application (ICSCA)*, 2023, pp. 1–8.

[3] N. Kadri and M. Koudil, "A survey on fault-tolerant application mapping techniques for network-on-chip," *Journal of Systems Architecture*, vol. 92, pp. 39–52, 2019.

[4] M. L. M. Peixoto, M. J. Santana, J. C. Estrella, T. C. Tavares, B. T. Kuehne, and R. H. C. Santana, "A metascheduler architecture to provide qos on the cloud computing," in *2010 17th International Conference on Telecommunications*, 2010, pp. 650–657.

[5] J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin, and Y. Watanabe, "Quasi-static scheduling for concurrent architectures," *Fundamenta Informaticae*, vol. 62, no. 2, pp. 171–196, 2004.

[6] B. Sorkhpour, "Scenario-based meta-scheduling for energy-efficient, robust and adaptive time-triggered multi-core architectures," 2019.

[7] P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Adaptive scheduling for time-triggered network-on-chip-based multi-core architecture using genetic algorithm," *Electronics*, vol. 11, no. 1, p. 49, 2021.

[8] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[9] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[10] B. Sorkhpour, "Scenario-based meta-scheduling for energy-efficient, robust and adaptive time-triggered multi-core architectures," Ph.D. dissertation, Universität Siegen, 2019. [Online]. Available: https://dspace.ub.uni-siegen.de/handle/ubsi/1471

[11] R. Obermaisser, H. Ahmadian, A. Maleki, Y. Bebawy, A. Lenz, and B. Sorkhpour, "Adaptive time-triggered multi-core architecture," *Designs*, vol. 3, no. 1, p. 7, 2019.

[12] H. Baniabdelghany, R. Obermaisser, A. Khalifeh, and P. Muoka, "Metascheduling using discrete particle swarm optimization for fault tolerance in time-triggered iot-wsn," *IEEE Internet of Things Journal*, vol. 10, no. 14, pp. 12 666–12 675, 2023.

[13] A. Pirhoseinlo, N. Osati Eraghi, and J. Akbari Torkestani, "Artificial intelligence-based framework for scheduling distributed systems using a combination of neural networks and genetic algorithms," *Mobile Information Systems*, vol. 2022, 2022.

[14] Z. Zang, W. Wang, Y. Song, L. Lu, W. Li, Y. Wang, and Y. Zhao, "Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation," *Computational intelligence and neuroscience*, vol. 2019, 2019.

[15] M. Melnik and D. Nasonov, "Workflow scheduling using neural networks and reinforcement learning," *Procedia Computer Science*, vol. 156, pp. 29–36, 2019.

[16] N. K. Gondhi and A. Gupta, "Survey on machine learning based scheduling in cloud computing," in *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, 2017, pp. 57–61.

[17] Z. Zhao, G. Verma, C. Rao, A. Swami, and S. Segarra, "Distributed scheduling using graph neural networks," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4720–4724.

[18] 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, no. 04, 2020, pp. 5077–5084.

[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *arXiv (Cornell University)*, 10 2017. [Online]. Available: http://arxiv.org/pdf/1710.10903

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," May 2016. [Online]. Available: https://arxiv.org/abs/1409.0473

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, vol. abs/1903.02428, 2019. [Online]. Available: http://arxiv.org/abs/1903.02428

[22] C. Lua, D. Onwuchekwa, and R. Obermaisser, "Ai-based scheduling for adaptive time-triggered networks," in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, 2022, pp. 1–7.