Graph Neural Networks Based Meta-scheduling in Adaptive Time-Triggered Systems

Samer Alshaer, Carlos Lua, Pascal Muoka, Daniel Onwuchekwa, Roman Obermaisser
Department for Computer Science and Electrical Engineering
University of Siegen
Siegen, Germany

***Abstract*—Meta-scheduling algorithms are used for adaptation in time-triggered systems as they adapt to different scenarios such as failures or different environmental conditions. Most meta-scheduling algorithms demand a considerable amount of storage space from the host cyber-physical system due to the state-space explosion problem in covering a reasonable number of scenarios. This work deploys the Graph Neural Network (GNN) to learn the multi-schedules from the meta-scheduling algorithm required for adaptation. The GNN is used to learn the scheduling mechanism of a Genetic Algorithm (GA) so that at runtime, adaptation is achieved using the GNN model. We further investigate the impact of modifiable tasks during a meta-scheduling operation on the overall makespan. Finally, a comparison of the makespans is made between a List Scheduler (LS), GA and the proposed GNN-based technique to evaluate the impact of our approach. Our proposed GNN-based approach outperforms the LS scheduler as the number of modifiable tasks increases. The results show that the proposed GNN-based meta-scheduling can be suitable for real-time scenario adaptation in cyber-physical systems.**

***Keywords*—Meta-scheduling, Adaptive Scheduling, Real-Time systems, Adaptive-time triggered systems, Graph Neural Networks.**

I. Introduction

The process of computing numerous schedules, each of which is mapped to a different situation, is known as meta-scheduling in the context of distributed system architecture. The scenarios can involve downtime, failure, and various operational modes; they are all referred to as context events in this study. When certain context events take place, the meta-scheduler is capable of switching schedules. Additionally, the schedule adjustment is pre-calculated offline while taking these context events into account. Meta-scheduling is a notion that frequently appears in literature under the names quasi-static scheduling [1] and super schedulers [2].

Meta-scheduling is especially important for safety-critical systems that require adaption services. These services are designed to improve system dependability characteristics such as reliability and availability. Although static scheduling approaches are predictable, they are restricted to dataflow specifications with no option [3]. Meta-schedulers, on the other hand, expand static schedulers with data-dependent options at run-time.

This work has received funding from the ECSEL Joint Undertaking(JU) under grant agreement No 877056. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Austria, Germany, France, Finland, Switzerland.

An overall architecture known as the Adaptive Time-triggered Multicore Architecture (ATMA) for safety-critical Cber-Physical Systems (CPS) is presented in [4]. ATMA supports adaptation using Multi-Scheduled Graphs (MSG) while preserving the main properties of time-triggered systems. The key properties include implicit synchronization, temporal predictability and avoidance of resource conflict. Our work is motivated by the need to provide efficient handling of the large numbers of schedules generated by the meta-scheduler algorithm in [4].

Sorkhpour [5] designed a scenario-based meta-scheduling tool for an Adaptive Time-Triggered Multi-core Architecture which incorporated context events such as core failures and slack events. The design used Mixed-Integer Quadratic Programming (MIQP) to solve scheduling problems at design time. The resulting outcome was the MSG, which is utilized at run-time. There was no proposed algorithm to handle the volume of data resulting from the MSG size, which poses an increasing storage demand for CPS. Muoka et al. [6] proposed an algorithm based on path reconvergence and reconvergence horizon to handle the state-space explosion problem resulting from an increase in the number of scenarios required for adaptation. Nevertheless, the technique only controls the size of the MSG to be deployed, which is a trade-off as the target number of context events is limited within a pre-defined horizon.

To the best of our knowledge, no work exploits a machine learning-based approach to handle the MSG size. Particularly, Graph Neural Networks (GNN) so far not been exploited for meta-scheduling, even though they have proven helpful in solving graph representable applications [7]. The use of GNN for learning the MSG is explored in this work.

GNNs are neural network models used for learning graph representations [8]. GNN's strength is attributed to its capability to capture node relationships through their features and locations in the graph.

Although this work proposes the GNN for learning the MSG, it also addresses concerns regarding guaranteeing correct temporal and spatial allocation of tasks in safety-critical applications. It addresses such concerns by only using the GNN to predict tasks' temporal priority, which will then be used to perform online schedule reconstruction.

Furthermore, we study the impact of modifiable tasks during the meta-scheduler operation for the proposed GNN

approaches, and the existing approaches, such as List Scheduling (LS), and Genetic Algorithm (GA). The reason for the investigation is due to the insight gotten after visualizing the schedules in previous works [5], [6], which show a considerable schedule change within a meta-scheduler when more tasks can be rescheduled. For this reason, we study the offline consideration of modifiable tasks for different schedulers; List scheduling, Genetic Algorithm, and our proposed GNN-based scheduler.

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

II. Related works

A general survey of scheduling algorithms for network-on-chip is carried out by Kadri et al. in [1]. The survey studied fault-tolerant application remapping approaches and classified them based on static, quasi-static and dynamic techniques.

Pellerin et al. in [9] surveyed articles that targeted hybrid meta-heuristics scheduling algorithms such as the GA for resource-constrained project scheduling problems. The research emphasized 32 robust swarm-based scheduling algorithms, concluding that reduced complexity and reliability criteria are not considered. Having these criteria is essential to determine the time of which the algorithm consumes to deliver results.

Pirhoseinlo et al. in [10] proposed an AI-based framework for scheduling distributed systems. The work utilized neural networks and a genetic algorithm similar to the approach proposed in our work. It incorporates the use of AI to predict the completion time of tasks to ease up the process of scheduling for the GA. In contrast, our work implements a meta-scheduling algorithm that uses GA to produce schedules for the MSG, which is then used to train the GNN. The resulting GNN is used to predict the temporal priorities of tasks.

Prado et al. in [11] presented a flexible fuzzy rule based evolution with swarm intelligence for meta-scheduling in grid computing. In this work, a unique machine learning approach was used to lower setting stages.The simulation results of the proposed approach showed improved functionality of knowledge acquisition with a swarm intelligence approach and reduced computational effort.

Rai et al. in [12] describe a meta-scheduler for operating systems, which influences the process scheduling decisions in multicore processors-based computer systems. The model was generated using machine learning to predict L2 cache behaviour. The authors claimed a 12% speedup using the meta-scheduler compared to 'Completely Fair Scheduler' of the Linux kernel. The work is similar to our proposed research in using machine learning techniques for adaptive scheduling. In the work of Rai et al., adaptability context is targeted toward the processor load, whereas we consider additional context events that affect multicores, such as failure and mode changes. In addition, the machine learning based inference in this case was designed to assist the scheduler in its operation by predicting L2 cache behaviour. However, we predict the temporal priorities of tasks directly.

Zhao et al. in [13] proposed a graph convolutional network trained to perform scheduling operations in wireless networks, which further exposes the potential of GNNs in solving scheduling tasks. The scope did not account for schedule changes triggered by adaptation, unlike our work, which focuses on learning the MSG to perform run-time adaptation.

Ma et al. in [14] designed an online planner selection with GNNs and adaptive scheduling. The GNN learns a swarm-based algorithm offline. The GNN is used in selecting the optimal candidate planner that may switch during operation. In contrast, our work uses GNN to predict the temporal priorities of tasks.

Andy Auyeung et. al. in [15] proposed a GA algorithm in cooperation with four heuristics to generate a solution for the scheduling problem. The proposed solution uses GA to search for the best possible combination of the four heuristic scheduling algorithms to generate schedules with shortest execution time. The results showed that the proposed solution outperforms the four algorithms individually.

This research is based on the work in [4]; where a GA based adaptive meta-scheduling algorithm is used to generate schedules mapped to context events offline. We extend the capabilities of the previous design by training a GNN model that replaces GA-based offline precomputation of schedules by run-time inference on schedules.

III. Meta-scheduler

The metascheduler computes time-triggered schedules taking into account the temporal and spatial properties of the system. The generated schedules will define the succession of tasks for a particular context event at any given time.

In time-triggered multi-core architectures, adaptation is motivated by energy efficiency, lower cost for fault-tolerance and reliability under changing environmental conditions. In such architectures, an aligned schedule switch is used to achieve adaptation. The meta-scheduler considers decision variables [4] ensuring consistency between parent and child schedules which prevents system failure on the schedule change.
This work builds on the meta-scheduling algorithm proposed in [4]. Where an offline mode based metascheduler is used to generate the MSG.

The meta-scheduler computes the MSG using a context model, an application model and a platform model and context model as its input.

Context model: It holds information regarding the context events that are considered at design time. These context events include the description of failure scenarios, slack events, operation mode and environmental mode changes.
Platform model: This model holds information regarding the hardware resources, such as cores, memories, routers and links between the routers.

Application model: This model describes the computation tasks with their deadlines, Worst-Case Execution Times (WCETs), message exchange between resources and reflects the precedence constraints between the tasks and messages.
Meta-scheduler: This component gathers information from the application, platform and context models to generate suitable schedules in the form of JSON files using schedule algorithms such as the 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 directed acyclic graph of the time-triggered schedules that is generated at design time. A node in the MSG graph is used at any instant 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, tasks 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 event 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.

Context event features are stored as edge features in the dataset data object. The same data object also contains the temporal priorities that need to be learned by the GNN.
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 solution.

The generation of the MSG is described in [6] using the Algorithm 1. The meta-scheduler computes a root schedule from which a calendar of events is established. It then applies the earliest event to modify the application or platform model for each event in the calendar. Subsequently, the GA-scheduler is executed to generate a new schedule and is then added to the MSG and linked to the root node. Finally, the meta-scheduler is called again with the new schedule as the parent node.

The generated MSG in design mode from the previous design suffers from issues of state explosion where the size of the generated MSG increases exponentially rendering it unusable in online mode.

Algorithm 1: GA-based meta-scheduler

Input: Application, Platform and Context Models
Output: MSG
Initiate Application model as AM;
Initiate Platform model as PM;
Initiate Context model as Cal;
Initiate Multi Schedule Graph as MSG;
Function *AM, PM*{
  Construct initial population (AM);
  Selection;
  Crossover;
  Mutation;
  Fitness evaluation (Genome in population, PM);
  return *Schedule with best makespan*
}
Function `meta-scheduler` (*AM, PM, Cal, Schedule*) {
  Take earliest event in e in Cal.
  cal‘ = Remove e from cal.
  AM‘, PM‘ = Modify AM, PM according to e;
  New schedule = GA-scheduler(AM‘, PM‘)
  Add New schedule to MSG as node (Sn);
  Add e to MSG as edge (e);
  If cal is not empty :
  meta-scheduler(AM,PM,Cal,New schedule);
  end if;
  return *MSG*
}
begin
  base schedule = GA-scheduler(AM,PM);
  add base schedule to MSG as node (S_0);
  meta-scheduler(AM,PM,Cal,base schedule)
end

IV. Proposed model

This work builds on the meta-scheduling algorithm proposed in [4]. Our proposed model uses the MSG generated by the meta-scheduler to train a GNN-based model that will be deployed at run-time. Figure 1 illustrates the proposed architecture of the GNN-based meta-scheduler.

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 component of the proposed model in 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 output 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 as shown in Algorithm

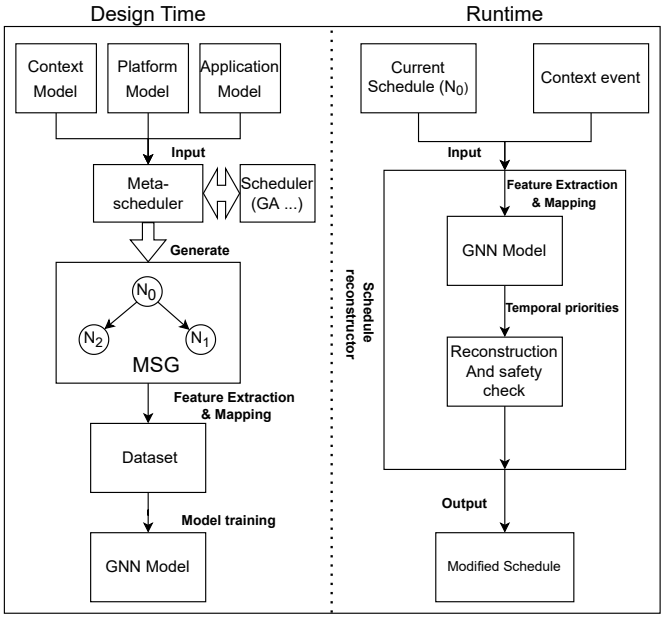

Figure 1. Meta-scheduler proposed model

(2) where temporal priorities are taken and assigned to the appropriate end system based on the communication costs covering spatial priorities.

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.

V. Experiment Setup

The GA meta-scheduler was used to generate 16,384 samples for an application model that contains 15 tasks and 14 context events. 80% of the generated data was used for training the GNN model and 20% is used for testing and makespan comparison. Pytorch geometric library [16] was used to perform classication on the MSG's irregular data structure, train a continuous kernel-based convolutional operator GNN based model; a supervised learning Message Passing Neural Networks (MPNNs) [17] [18] as shown in Equation (1) which is a neural network that aggregates feature through passing information from each node to it's neighbour.

$$X_i^{'} = \Theta X_i + \sum_{j \in N(i)} X_j \cdot h_\Theta(e_{i,j}) \tag{1}$$

where h_Θ denotes a neural network designed to extract edge features. X denotes node features. e denotes edge features. Θ denotes adjustable weights.

Algorithm 2: Schedule reconstruction

Input: $tasks, msgs, ES$,Context event time
Output: schedule, makespan

Function `Allocation`$(tasks, msgs, ES)$**:**
  Remove tasks that started before context event time.
  Append tasks with satisfied precedence constrains in *tasks_ready*
  for *each* N *in* $tasks$ **do**
    Select from *tasks_ready* the tasks with the highest *temporal priority*
    for *each* m *in* $end\_systems$ **do**
      if $parents[j]$ *is* 0 **then**
        Append $endTime[m]$ to vector $start$
      else
        Append $max(endTime[m], DRT(m, j, msgs, tasks))$ to vector $start$
      end if
    end for
    $st\_time[j] = min(start)$
    $end\_time[j] = st\_time[j] + ex\_time[j]$
    $end\_sys[j] = ES[st\_time.idx(min(start))]$
    if $parents[j]$ **then**
      for *each* p *in* $parents[j]$ **do**
        Append $inj\_time + comm\_cost(p, j)$ in vector *arrrival*
      end for
      $st\_time[j] = max(max(arrival), min(start))$
      $end\_time[j] = st\_time[j] + ex\_time[j]$
      $endTime[idx.min(start)] = end\_time[j]$
    else
      $endTime[idx.min(start)] = end\_time[j]$
    end if
    Remove j from vector $tasks\_ready$
    Append $children[j]$ to vector $tasks\_ready$
  end for
  Append makespan
  return schedule, makespan

The model utilizes different Multi Layer Perceptrons (MLPs) that contains 2 layers for edge features, and 4 layers including the input and output layers with a total of 192,957 adjustable weights. Schedules generated from the GNN model were compared to the schedules from the GA meta-scheduler to determine the quality of schedules produced in terms of makespan. In addition, we compared our approach to schedules generated by the LS. The LS uses the bottom level to determine task temporal priorities while the GNN based scheduler uses priorities learned from the GA.

The schedules are categorized according to the number of tasks that occured after the context event time in each schedule of the MSG. We provide an insight into the rescheduling operation using Figure 2. In this example, we consider a context event occurrence with 50% slack. Tasks that started before the event occurrence remain the same. However, tasks that occurred after the event are re-ordered accordingly to adapt to the context event. Figure 2 shows two schedules used to illustrate the concept of modifiable tasks. The first schedule above is without the application of the context event. It shows

the base schedule. The schedule below, shows how a 50% slack on task 4 is handled. The 50% slack on task 4 means that task 4 finishes before its WCET. To take advantage of the slack event, we invoke the meta-scheduler to recompute a new schedule to adapt to the scenario. By so doing, we get a better makespan which can then be utilized for saving energy. The number of task that can be re-scheduled are referred to herein as modifiable tasks. These tasks in the example are shown in the shaded portion of the Figure 2 below.

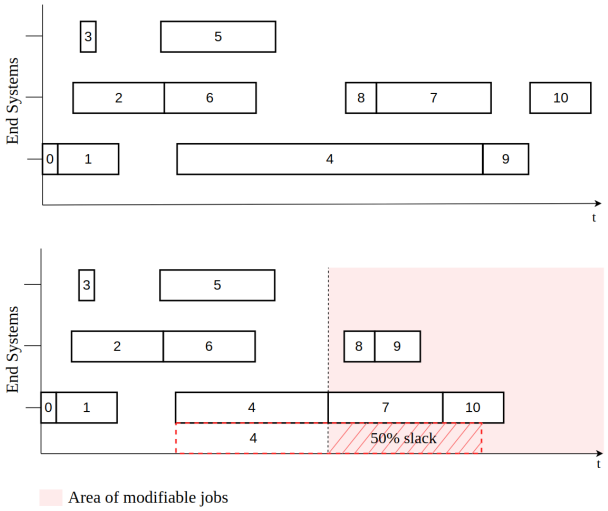

Figure 2. Rescheduling with context event.

Each sample in the generated 16,384 samples is categorized according to the number of modifiable tasks. And each category is tested separately on all the meta-scheduling algorithms.

VI. Results and Discussions

This section elaborates on the acquired results for training the GNN model to learn the prediction outputs of the GA meta-scheduling algorithm. After which we show the results of the makespans corresponding to the proposed GNN, GA, and LS.

Figure 3. Shows the train loss for each training epoch that iterates though the training sample set. The results show decline in the train loss throughout the training operation. Equation (2) shows the Cross Entropy Loss function used in the training the GNN:

$$Loss = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2}$$

where M - number of classes y - truth indicator (0 or 1) if class c is the correct for observation o. p - predicted probability observation o is of class c.

Throughout the training process, the accuracy of the GNN inference model training operation is calculated by validating 10% of the training dataset for each epoch as shown in figure 4. The results show a training accuracy of 97% at the end of training at 180 epochs. In addition, 20% of the entire dataset

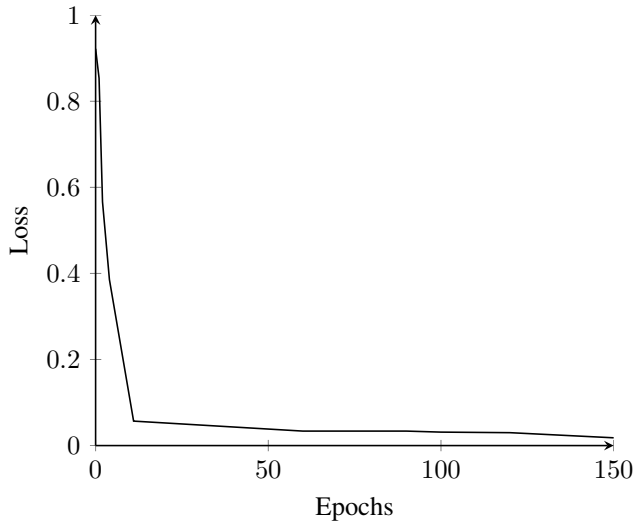

Figure 3. Train Loss.

was used for accuracy validation testing, an accuracy of 76% was obtained.

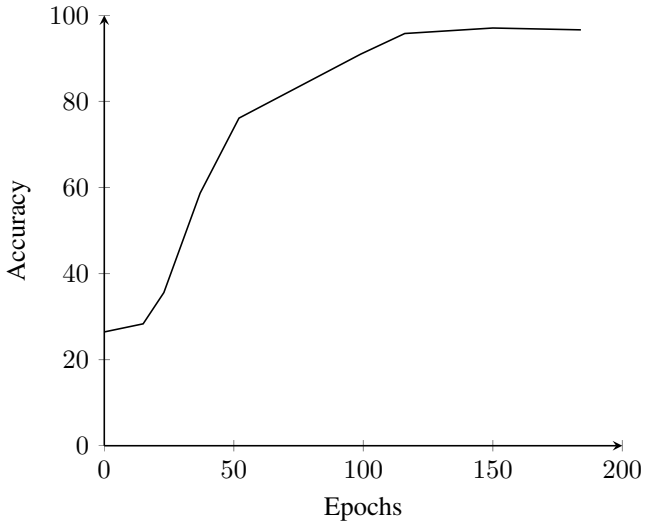

Figure 4. Train accuracy.

Figure 5. Shows the output results for the makespans for each schedule generate from the GA, GNN and LS. Each sample schedule is classified based on the number of modifiable tasks in each schedule as previously explained in the experimental setup section. Throughout the dataset, each schedule (node) in the MSG is given a label that would define the number of modifiable tasks in the schedule. This is determined by the occurrence time of the context event itself. Schedules with the same labels are grouped together and compared with different meta-scheduling algorithms. The results show that when the scheduling problem is simple with less than 10 modifiable tasks, all meta-scheduling algorithms perform almost the same and the GNN performs the worst

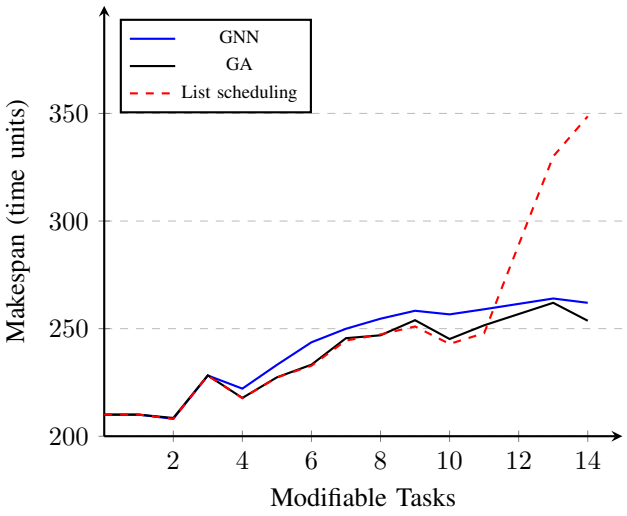

Figure 5. Makespan comparison.

with an average ranging from 10 to 20 time units difference from the GA and LS. However, when the complexity of the scheduling problem is increased with more number of tasks, the quality of makespan for the list scheduling falls behind GNN and GA techniques. The results show that in terms of makespan quality, the GA ranks first, followed by the GNN at rank 2, and finally the LS.

GA cannot be used in run-time due to its demanding computational requirements. GA is unsuitable for hard real-time systems due to its unpredictable high timing and computational requirement. The LS is desirable for hard real-time application due to its computational speed compared to the GA. Nevertheless, since the makespan quality is no where compared to the GA, the proposed GNN technique provides a way to utilize the benefits of the GA in application with stringent timing requirements.

VII. Conclusion and Future Work

The results generated from the GNN based system model showed great approximation to the GA schedules. The GNN model is used to learn the scheduling mechanism of the GA, therefore it is highly improbable for the GNN to outperform the GA. Nevertheless, the quality of schedules in terms of makespan generated by the proposed GNN approach shows great potential to be deployed for real-time operations. This works studies the impact of the different scheduling algorithms based on the number of modifiable tasks. Future plans for this research project include performing further investigations on GNN meta-scheduling algorithims; There exists several GNN algorithms that are capable of exploiting the use of edge attributes to be explored.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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

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

[9] R. Pellerin, N. Perrier, and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 280, no. 2, pp. 395–416, 2020.

[10] 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.

[11] R. P. Prado, J. E. M. Expósito, and S. Garcia-Galan, "Flexible fuzzy rule bases evolution with swarm intelligence for meta-scheduling in grid computing," *Computing and Informatics*, vol. 33, no. 4, pp. 810–830, 2014.

[12] J. K. Rai, A. Negi, R. Wankar, and K. Nayak, "A machine learning based meta-scheduler for multi-core processors," *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, vol. 1, no. 4, pp. 46–59, 2010.

[13] 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.

[14] 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.

[15] A. Auyeung, I. Gondra, and H. Dai, "Multi-heuristic list scheduling genetic algorithm for task scheduling," in *Proceedings of the 2003 ACM symposium on applied computing*, 2003, pp. 721–724.

[16] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[18] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.