GNN Link Prediction for Time-Triggered Systems

Carlos Lua, Yeping Zhang, Omar Hekal, Daniel Onwuchekwa, Roman Obermaisser
Department of Embedded Systems
University of Siegen
Siegen, Germany
carlos.luamorales@uni-siegen.de

***Abstract*—Research on graph neural networks (GNNs) has increasingly gained popularity recently. GNN is considered a powerful tool for solving machine learning tasks that require dealing with irregular topologies such as graph data. Meanwhile, solving the scheduling problems for time-triggered systems has been debated for a long time. Even though several algorithms were proposed to solve this problem, none considered exploiting GNN partially or wholly, solving time-triggered scheduling. In this work, we propose an approach for dynamic adaptation in time-triggered systems using GNN. We use GNNs to solve scheduling problems for time-triggered systems by transforming job allocation problems to link prediction tasks. The preliminary results show that GNNs have a promising potential to perform job allocation problems in time-triggered systems.**

***Keywords*—Graph neural networks, link prediction, time-triggered systems, scheduling**

I. Introduction

Time-triggered systems are computer systems that execute one or more sets of jobs based on predetermined schedules. The implementation of time-triggered systems involves the deployment of a global time base that drives a job dispatcher to release jobs at predetermined points in time [1]. Time-triggered systems can support adaptive systems behaviour by deploying pre-computed schedules for relevant events at development time. This adaptive behaviour can improve energy efficiency, reliability, and context awareness. For example, information about faults can help recovery by re-distributing application services to the system's remaining resources. The predictability and fault tolerance of time-triggered systems has made its utilization dominant in many safety-critical applications that require their services to be maintained under all load and fault assumptions to minimize risk for people, property, and the environment [2, 3].

Scheduling is the process of efficient resource allocation of jobs to ensure job completion within deadlines [4]. Scheduling in time-triggered systems is mainly carried out offline. Several approaches were proposed in prior works trying to find feasible solutions to the scheduling problem utilizing genetic algorithms (GA) [5], heuristic list schedule techniques [6] or meta-scheduling techniques [7]. Time-triggered systems are widely used in safety-critical applications, but achieving these adaptivity requirements is restricted by the number of offline pre-computed schedules. The resulting computed schedules are usually stored within an embedded device that performs a look-up of a schedule triggered by context events. Generating schedules for different context events using any of the previously mentioned tools may lead to an exponential growth in the number of computed schedules causing a state-space explosion problem [8].

The fact that genetic scheduling algorithms require high computation time makes their deployment during run time inapplicable [5]. On the contrary, list scheduling techniques are fast in computation, but their makespan results are inferior compared to genetic algorithms [6]. A more efficient scheduling technique is needed to overcome the traditional scheduling algorithms' limitations concerning storage spaces, makespans, and computation time. The recent breakthrough in the artificial intelligence field turned researchers' efforts towards deploying artificial neural networks (ANNs) to solve scheduling problems. ANNs can provide the opportunity to learn and infer schedules at runtime with short delays.

Graph neural networks have been recently utilized in solving various scheduling problems for different applications such as job shop scheduling [9, 10], multi-robot task scheduling [11] and production scheduling [12]. Graph neural networks are deep learning-based methods that operate on graphs. The graphs are data structures that model a set of objects and their relations [13]. Many learning tasks require dealing with graph data due to their expressive power and ability to capture rich relational interactions between graph elements. Graphs can be used to denote systems from various areas, such as social networks [14], physical systems [15] and protein-protein interactions [16]. The unique non-euclidean topology of graphs allows applying different types of machine learning tasks such as node classification [17], link prediction [18, 19], and graph classification [20].

This paper aims to deploy knowledge about graph neural networks to solve scheduling problems of time-triggered systems by transforming the job allocation problem into a link prediction task. A heterogeneous graph models a specific schedule, where a graph node represents a job, and the relations/dependencies between different jobs or between jobs and corresponding processors represent the graph links. The scenario generator component of the AI-based scheduling algorithm described in [21] is used to generate heterogeneous graphs. The resulting heterogeneous graphs are the input dataset to our model. We then use GNN to apply link prediction on the heterogeneous graph datasets. The goal is to inspect whether GNNs could successfully assign the jobs to their corresponding

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.

processors and provide schedules similar to the ones generated with the GA in the AI-based scheduling approach.

The remainder of this work is organized as follows. Section II discusses the related work. Section III presents the system model. The link prediction representation is presented in section IV. Results of the experiments are described in section V. The conclusion is discussed in section VI.

II. Related work

Graph neural networks have been recently employed in many different fields. Lilapati W. et al. [22] introduced a comprehensive overview of GNNs. They listed the possible tasks that can be performed over different graph levels, such as node-level tasks, which include node classification and clustering; edge-level tasks, under which lies edge prediction and classification; and graph-level tasks, which include graph classification. Authors categorized the variants of graphs based on the structure and the scale of graph data into directed, heterogeneous, dynamic, and attributed graphs. These different graph topologies enabled the real-world deployment of GNN in more applications.

Alex F. et al. [16] harnessed the GNN knowledge in drug discovery. They used GNNs to predict interfaces between proteins. They proposed a scheme that can make accurate predictions using either sequence information alone or in conjunction with structure-based features. The resulting algorithm showed better accuracy than the SVM approach [23].

Wenqi Fan et al. [24] utilized GNNs to carry out a social recommender system, which can be realized from the GNN perspective as a link prediction task. Furthermore, they introduced a novel GNN framework (GraphRec) for social recommendations. Results showed that their proposed method outperforms other methods, such as DeepSoR [25] and GCMC+SN [26].

Moreover, Shiwen Wu et al. [27] carried out a comprehensive review of the different contributions regarding GNN-based recommender systems. They categorized the existing works into user-item collaborative filtering, sequential recommendation, social recommendation, and knowledge graph-based recommendation. They argued that the proposed GNN-based recommendation frameworks could be utilized for recommendation tasks due to their representation of learning properties. In addition, they claimed that most of the data for the recommendation tasks could be modelled as a graph structure.

Machine learning algorithms have been recently deployed to tackle scheduling problems. For example, Carlos et al. [21] developed an adaptive artificial intelligence quasi-static scheduling approach for time-triggered scheduling problems. The model is trained offline to predict job priorities used to generate correct schedules. The priority of jobs is predicted by an AI-based model trained offline. At run time, adaptation is fulfilled by generating a new schedule for any new scenario. Results showed that while their approach was still far from the genetic algorithm (GA), it was superior to list heuristics in terms of makespan. However, using GA-based schedulers is not recommended for real-time applications due to their long computation time.

Chunmeng Z. et al. [28] proposed a deep reinforced learning-based schedule for time-triggered ethernet (DRLS). The DRL-based scheduling algorithm includes two phases. First, agents are trained using specific or random network topologies in an offline training phase. Then an online inference phase follows, where the trained agent computes the time-triggered schedule for the specific network topology and the flow requirements incrementally. Results showed that DRLS could adapt to specific network topologies and performs better than traditional heuristic-based methods.

All recent contributions deployed GNN for scheduling techniques to find feasible scheduling solutions offline. In contrast, we aim to use GNN to perform job assignments using link prediction. In this work, we apply the link prediction algorithm to a heterogeneous graph consisting of multiple edge and node categories. Our goal was to predict the edge category representing the job allocation. We do not predict the entire schedule due to safety considerations since a single error can cause the entire schedule to be invalid. Instead, we restrict the problem to predicting the job assignment to investigate the prospect of getting better makespans for a list scheduler. The goal of applying link prediction is assigning the jobs to their corresponding processors.

III. System Model

Scheduling problems can be abstractly defined as follows: given a set of jobs $\mathcal{J} := \{J_1, ..., J_n\}$, which have to be processed on a set of processors $\mathcal{P} := \{P_1, ..., P_m\}$, the goal is to assign each job a corresponding processor depending on predefined constraints such as [29]:

1) Each processor can handle one job at a time.
2) Each job can only be allocated to one processor at any time.

The time-triggered scheduling problem can be represented by application and platform models. The application model, shown in Figure 1, also known as the task graph, is a directed acyclic graph carrying information about the jobs and messages. The task graph can be modeled as $G = (\mathbf{V}, \mathbf{E}, w, c)$, where $\mathbf{V}$ are graph nodes representing the jobs, $\mathbf{E}$ represents the connection between jobs, The computation cost w is a positive weight associated with a given node $n \in V$. The communication cost c is a non-negative weight associated with members of the edge $\mathbf{E}$. The platform model, shown in Figure 2, is a separate undirected graph that includes the hardware information, such as switches/routers, end systems/processors and the interconnection pattern between them.

Traditional scheduling approaches aim to deploy knowledge about application and platform models to find a solution for the scheduling problem. The scheduling approach should maximize a fitness function, including small makespan and computation periods. The current techniques tend to find solutions to the job allocation problem offline. Different schedules are generated at development time, each corresponding to a certain context event. Different techniques can generate schedules, for example, heuristic scheduling techniques, genetic algorithms (GA), or deploying artificial neural networks, which is used in our case. We use the resulting schedules to generate heterogeneous

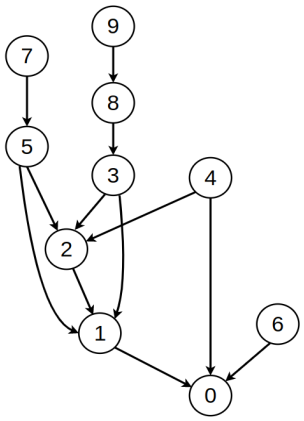

Figure 1. Shows the application model for ten jobs

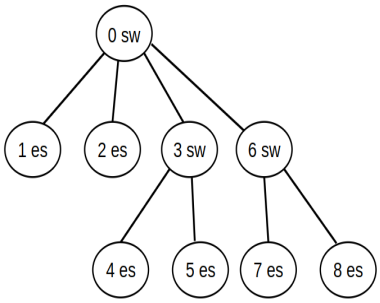

Figure 2. Platform model conformed by switches (sw) and end systems (es).

graphs, which are the inputs to our Graph neural network (GNN) link prediction model.

This work aims to motivate the idea of solving scheduling problems at run time. At runtime, application and platform models represent heterogeneous graphs with missing links between jobs and processors. The GNN link prediction model is deployed to find the missing links that correspond to the solution to the scheduling problem. Figure 3 shows two flow charts, which illustrate the offline and runtime processes. In the offline process a scheduling tool is used to provide the require solutions for the different scenarios. The solutions together with the input application and platform models are transformed into a heterogeneous graph using the PyTorch Geometric library. Before inserting the graphs into the training model links need to be erased from the graphs depending on the prediction objective. For the runtime scheduling the models are transformed into a graph to subsequently insert it into the model for the link prediction. A schedule reconstructor block ensures that the schedule is valid.

IV. Link Prediction Representation

This section introduces the technique for transforming the job allocation problem into link prediction. The goal of link prediction is to predict the existence of a connection (edge) between two nodes. GNNs can learn graph structures and node features, which make them superior to traditional methods [30] that rely on calculating neighborhood overlap between two nodes so that the node similarity score acts as the likelihood of a link existing between these nodes.

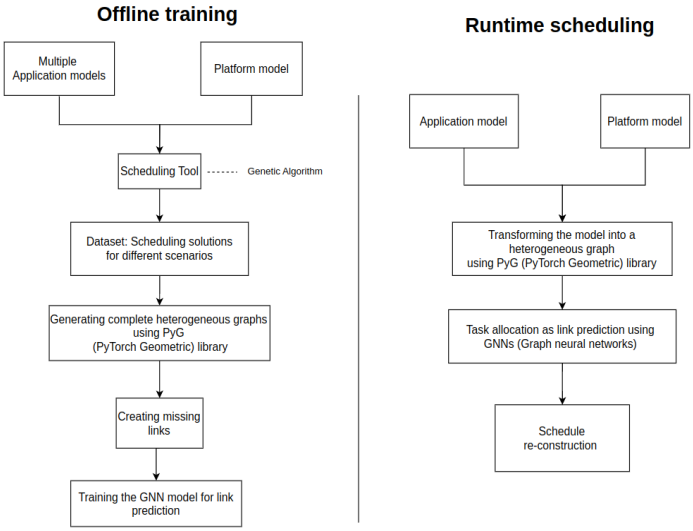

Figure 3. Flow diagrams showing the process of job allocation for the offline training (left diagram) and the runtime scheduling (Right diagram).

The previous knowledge about the potential application of GNNs motivated us towards using link prediction for scheduling. In our experiments, we used one platform model and different application models consisting of 10 jobs. We used the scenario generator block described in [21] to generate the input data. The scheduler block is then used to generate feasible schedules for each input data fulfilling the pre-defined constraints and minimizing the makespans.

The solution is recorded as a JSON file specifying which processor each job should run on. An example of the job allocation to different processors is shown in the following table:

Table I
Jobs allocation to different processors

Job nodes	0	1	2	3	4	5	6	7	8	9
Processor nodes	4es	5es	4es	5es	2es	5es	5es	2es	2es	4es

Based on the solution files and using the PyTorch geometric library (PYG), we combine the different application models with the platform models to represent the solutions of the scheduling problems by constructing multiple heterogeneous graphs. Each heterogeneous graph represents the solution for a specific application model.

The heterogeneous graph, shown in Figure 4 consists of three sets of nodes $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \mathcal{V}_3$. $\mathcal{V}_1$ represents the job set marked by red in the graph, $\mathcal{V}_2$ represents the switches marked by the blue color, and $\mathcal{V}_3$ represents the processors having the green color in the graph. The edges satisfy specific constraints based on the type of nodes they connect. The undirected dotted lines join different jobs with different processors meaning this job is running on that specific processor. For example, job (4) should run on processor number (5) based on the scheduling solution.

The heterogeneous graphs are used as input to our GNN link prediction model so that the model can learn different node embeddings. The objective is that after training the model, it should be able to predict whether an edge exists between

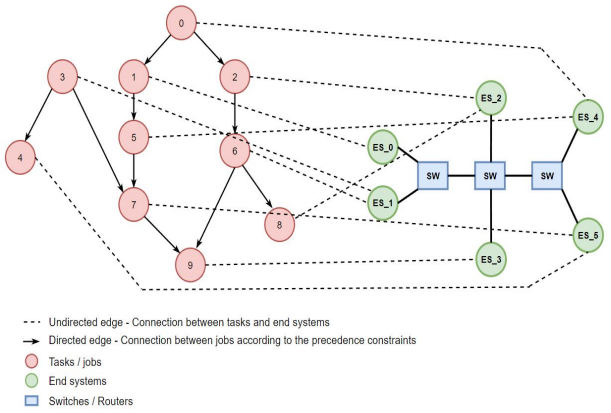

Figure 4. Heterogeneous graph representing a solution to a scheduling problem for an application model of 10 jobs.

a specific job node and a processor node. Edges that need to be predicted whether they exist are called positive edges. Meanwhile, the remaining processor nodes not edge-connected to this job node will have artificial links created during the training process, called non-existent or negative edges [18, 31].

The trained model should be able to rank the positive edges higher than the negative ones to make a distinction between them. During the prediction phase (testing), a certain number of edges joining the jobs with their corresponding processors (dotted edges) is masked and the prediction is determined based on the propability computed by the inference model.

V. Experiments and Results

In this section, we decribe the experimental procedures together with the results. The experiments were carried out in the OMNI cluster of the University of Siegen. The dataset for this experiment consists of 40,000 heterogeneous graphs representing the solutions for the scheduling problem from the GA for each of the 10-job and 40-job application models. For model training, 39,000 heterogeneous graphs were used wheras the remaining 1000 are used to evalute the model performance. Data prepartion procedure is shown in Figure 5. From the application models node and edge features were extracted:

- Node features
  - Execution time: Time needed for the job to be executed.
  - Node degree: Number of edges adjacent to the node.
  - Clustering coefficient: It is a measure that defines how closely connected the nodes in a graph are.
  - Node centrality: It measures the importance of a node in the overall graph based on its location.
- Edge features
  - Message size

The resulting heterogeneous graphs from the data processsing procedure are used as inputs for the GNN-Link prediction model for training and testing purposes. For the training model, we defined a function to delete a specific number of edges between jobs and processors.

The 1000 graphs used for prediction to evaluate the model performance were split into three groups depending on the

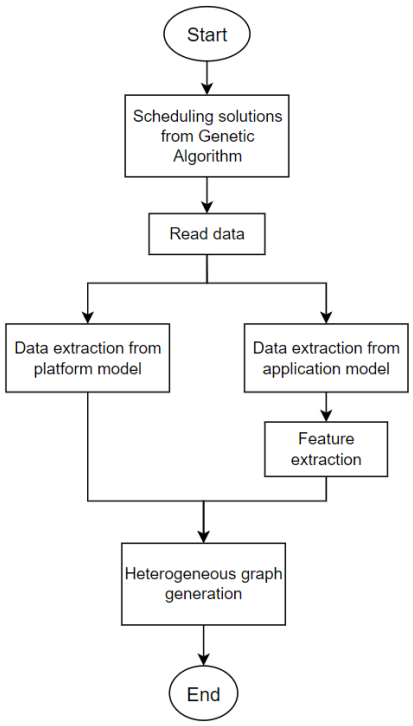

Figure 5. Data processing steps

number of erased/predicted links. In the first group, only one link per graph was deleted. In the second group, half of the links were deleted. Finally, in the last group, all the links were deleted. In every group, the final goal is to predict the missing links.

The accuracy of training, testing and prediction is defined based on the number of the correct positive edges the model can output. The predicted results are compared with the original deleted value to get the accuracy of the prediction of the model. The results for training and testing the job datasets are presented in Table II.

Table II
Prediction accuracy of different models for the two datasets

Dataset	Predicted links	Accuracy using different models (%) 1	2	3	4
10 jobs	One link	36.06	32.33	31.33	31
	Half links	34.33	32.93	32.28	32.66
	All links	31.9	30.1	28.66	29
40 jobs	One link	41	35	33	34
	Half links	35.53	34.76	32.6	32.82
	All links	35.05	35.82	32.51	31.9

We recorded the prediction accuracy of 4 different models, each of them with different training parameters. The prediction accuracy reflects the results of the model operation. The more edges we erase during the training process, thus more edges to be predicted, the less accurate the final prediction result will be. The highest recorded prediction accuracy was 41%. There is a difference in the accuracy from the 10 and the 40 jobs model. In the 40 jobs model, there is more information from

which the GNN can learn due to the size of the graph, which leads to better prediction accuracy.

VI. Conclusion

This work investigates the applicability of deploying GNNs to solve scheduling problems. It expresses the scheduling problem as a link prediction task, where the goal is to map each job with its corresponding processor by learning the behaviour of the genetic algorithm. We used heterogeneous graph representations to model the scheduling solutions obtained from the genetic algorithm to prove the capability of our assumption deploying the GNN link prediction algorithm in solving scheduling problems. Several factors can affect the obtained accuracy, one of them being the relatively small amount of data used during this experiment, unlike other datasets typically used to apply link prediction with GNNs such as in [32], where all the extracted data comes from an enourmous single graph. Although the accuracy of the predictions could be better, it should be kept in mind that these predictions can also lead to correct solutions, i.e. makespans that fulfil the intended deadlines. The results show the potential of applying GNN-Link prediction in solving scheduling problems. Future work is expected to improve the overall accuracy. Furthermore, applying GNN-link prediction would enable finding feasible scheduling solutions at run-time, thus saving storage space usually occupied by the pre-computed schedules at the development time resulting from the traditional time-triggered scheduling techniques.

References

[1] M. J. Pont, *Patterns for time-triggered embedded systems*. TTE System, Ltd, 2008.

[2] F. Heilmann, A. Syed, and G. Fohler, "Mode-changes in cots time-triggered network hardware without online reconfiguration," *ACM SIGBED Review*, vol. 13, no. 4, pp. 55–60, 2016.

[3] 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.

[4] T. R. Ramanan, R. Sridharan, K. S. Shashikant, and A. N. Haq, "An artificial neural network based heuristic for flow shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 22, no. 2, pp. 279–288, 2011.

[5] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, vol. 1. IEEE, 2018, pp. 337–344.

[6] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," *ACM Sigbed Review*, vol. 16, no. 1, pp. 15–20, 2019.

[7] B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 0143–0150.

[8] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *LASER Summer School on Software Engineering*. Springer, 2011, pp. 1–30.

[9] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning," *International Journal of Production Research*, vol. 59, no. 11, pp. 3360–3377, 2021.

[10] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.

[11] H. Kang, A. Mynbay, J. R. Morrison, and J. Park, "Scalable and transferable learning of algorithms via graph embedding for multi-robot reward collection." *CoRR*, 2019.

[12] T. Seito and S. Munakata, "Production scheduling based on deep reinforcement learning using graph convolutional neural network." in *ICAART (2)*, 2020, pp. 766–772.

[13] A. Menzli, "Graph neural networks: Libraries, tools, and learning resources," Jul 2022. [Online]. Available: https://neptune.ai/blog/graph-neural-networks-libraries-tools-learning-resources

[14] Y. Wu, D. Lian, Y. Xu, L. Wu, and E. Chen, "Graph convolutional networks with markov random field reasoning for social spammer detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 1054–1061.

[15] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4470–4479.

[16] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *Advances in neural information processing systems*, vol. 30, 2017.

[17] S. Xiao, S. Wang, Y. Dai, and W. Guo, "Graph neural networks in node classification: survey and evaluation," *Machine Vision and Applications*, vol. 33, no. 1, pp. 1–19, 2022.

[18] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[19] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Revisiting graph neural networks for link prediction," *International Conference on Learning Representations*, 2021.

[20] Y. Wang, Y. Zhao, N. Shah, and T. Derr, "Imbalanced graph classification via graph-of-graph neural networks," 2022, pp. 2067–2076.

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

[22] L. Waikhom and R. Patgiri, "Graph neural networks: Methods, applications, and opportunities," *arXiv preprint*

arXiv:2108.10733, 2021.

[23] F. u. A. Afsar Minhas, B. J. Geiss, and A. Ben-Hur, "Pairpred: partner-specific prediction of interacting residues from sequence and structure," *Proteins: Structure, Function, and Bioinformatics*, vol. 82, no. 7, pp. 1142–1155, 2014.

[24] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[25] W. Fan, Q. Li, and M. Cheng, "Deep modeling of social relations for recommendation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[26] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[27] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[28] C. Zhong, H. Jia, H. Wan, and X. Zhao, "Drls: A deep reinforcement learning based scheduler for time-triggered ethernet," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–11.

[29] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[30] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[31] J. Tang, S. Chang, C. Aggarwal, and H. Liu, "Negative link prediction in social media," in *Proceedings of the eighth ACM international conference on web search and data mining*, 2015, pp. 87–96.

[32] X. Song, R. Ma, J. Li, M. Zhang, and D. P. Wipf, "Network in graph neural network," *arXiv preprint arXiv:2111.11638*, 2021.