Choosing a Classical Planner with Graph Neural Networks

Jana Vatter[1], Ruben Mayer[2], Hans-Arno Jacobsen[3], Horst Samulowitz[4], Michael Katz[4]

[1]Technical University of Munich
[2]University of Bayreuth
[3]University of Toronto
[4]IBM Research

jana.vatter@tum.de, ruben.mayer@uni-bayreuth.de, jacobsen@eecg.toronto.edu, samulowitz@us.ibm.com, michael.katz1@ibm.com

Abstract

Online planner selection is the task of choosing a solver out of a predefined set for a given planning problem. As planning is computationally hard, the performance of solvers varies greatly on planning problems. Thus, the ability to predict their performance on a given problem is of great importance. While a variety of learning methods have been employed, for classical cost-optimal planning the prevailing approach uses Graph Neural Networks (GNNs). In this work, we continue the line of work on using GNNs for online planner selection. We perform a thorough investigation of the impact of the chosen GNN model, graph representation and node features, as well as prediction task. Going further, we propose using the graph representation obtained by a GNN as an input to the Extreme Gradient Boosting (XGBoost) model, resulting in a more resource-efficient yet accurate approach. We show the effectiveness of a variety of GNN-based online planner selection methods, opening up new exciting avenues for research on online planner selection.

1 Introduction

Automated planning is a foundational discipline within the field of Artificial Intelligence (AI) research (Russell and Norvig 1995). The focus of planning lies in formulating goal-oriented policies, or, in the deterministic case, sequences of actions to achieve predefined goals. Applications range from robotics and autonomous vehicles to industrial automation (Ghallab, Nau, and Traverso 2004; Rogalla, Fay, and Niggemann 2018; Karpas and Magazzeni 2020; Chakraborti et al. 2020).

Given the inherent complexity of classical planning, as described by (Bylander 1994), there is unlikely to be a single planning algorithm that can work well across diverse planning problems. Over the years, a variety of planners were developed, tackling various aspects that make planning problems challenging. Consequently, there is a growing interest in the development of portfolio-based approaches (Seipp et al. 2012; Vallati 2012; Cenamor, De La Rosa, and Fernández 2013; Seipp et al. 2015) where multiple planners are aggregated and out of this portfolio, a single planner or a schedule of planners are selected. Portfolio-based planning can be divided into *offline* and *online* methods. While *offline* methods (Helmert, Röger, and Karpas 2011; Seipp et al. 2012) construct a single invocation schedule ahead of time which is anticipated to perform well across many domains, *online* methods (Cenamor, De La Rosa, and Fernández 2013; Katz et al. 2018; Ma et al. 2020; Ferber and Seipp 2022) adapt by learning to select the most suitable planner for each specific task.

Focusing our attention on the online methods, a variety of deep learning based approaches were recently developed. There are Convolutional Neural Networks (CNNs) based approaches (Katz et al. 2018; Sievers et al. 2019), with the planner *Delfi* winning the cost-optimal track of the International Planning Competition (IPC) 2018, where each planning problem is represented by an image. However, these images are obtained by first converting the planning problem to a graph, for instance as a problem description graph (PDG) (Pochter, Zohar, and Rosenschein 2011), or as an abstract structure graph (ASG) (Sievers et al. 2017). Therefore, following the success of *Delfi*, a first attempt was made to use Graph Neural Networks to directly learn on the graphs and capture structural information (Ma et al. 2020).

In our work, we continue the line of work on using GNNs directly for online planner selection. We focus on exploring four GNN architectures, two graph representations, various dataset features and multiple prediction tasks. We also combine the advantages of GNNs and Extreme Gradient Boosting (XGBoost) (Chen and Guestrin 2016). Our main contributions are as follows: We explore the strengths and weaknesses of the GNN architectures for online planner selection. We investigate two graph representations, namely the lifted and grounded representation, in combination with additional node features. We focus on different ways to pick a planner including predicting the probability that a planner solves a task and the time it takes a planner for solving the task. Additionally, we experiment with the use of graph representations obtained by the GNNs for other ML-based methods like XGBoost.

2 Related Work

In this section, we embed our work into related approaches by highlighting and comparing them to ours. We provide an overview of recent ML-based approaches as well as approaches using Graph Neural Networks (GNNs). We incorporate related methods solving various tasks in the planning domain, including approaches using the IPC dataset for automatic planner selection.

Cenamor, De La Rosa, and Fernández (2016) focus on creating a configurable portfolio which adapts to a given planning task. First, a number of candidate planners is selected with a filtering method. After that, predictive models estimate the time it takes to solve a task. Based on the combined results of the predictive models, it is decided which planners to include in the portfolio. Instead of creating portfolios, our work focuses on the task of directly choosing a planner for a given planning problem. In addition, we use GNNs to perform the prediction task.

The Delfi planner (Katz et al. 2018) makes use of graphical representations of a planning task in combination with deep learning techniques. Each planning problem is converted to an image and a CNN is used to make predictions. By applying it to new benchmarks, the authors show a good generalization of the proposed model. We use the same data, but instead of CNNs, experiment with GNNs.

Sievers et al. (2019) further pursue the method of converting planning problems to images and applying a CNN model. They analyze the shortcomings of the current methods and present possible solutions. Their main findings include the need for methods working well with non-IID (independent and identically distributed) data and exploring alternative network architectures. They especially analyze the shortcomings and possible solutions of existing CNN-based approaches. However, we mainly analyze how we can use GNNs to solve the automatic planner selection task.

Another related work is by Ferber and Seipp (2022), who explore graph features which are used during training simple machine learning models, for instance, linear regression or random forests. The authors analyze the features and their importance for the training process. We distinguish ourselves by exploring different features when using different GNN models.

The closest to our work is the previous work on using GNNs for Automatic Planner Selection is by Ma et al. (2020). The authors propose using two GNN architectures, namely GCN (Graph Convolutional Network) and GGNN (Gated Graph Neural Network), to select candidate planners. Their experiments with the IPC dataset show that their graph-based approach outperforms previous image-based ones. They highlight the ability of GNNs to capture structural information of a planning graph and address the lack of node-level information in previous approaches. The authors show that the lifted representation is favored over the grounded one as it produces more consistent results. However, it should be noted that the lifted representation contains much more nodes in the graphs and therefore scalable training approaches are needed. We distinguish ourselves by not only including two GNN architectures, but a set of four representative architectures with different focuses. In addition, we explore the use of node features for GNN training and combine GNNs with a simpler ML-based method.

Chen, Trevizan, and Thiébaux (2023) propose an approach based on the Weisfeiler-Lehman graph isomorphism test for learning heuristics for planning. Compared to GNN-based methods, they show a better performance in terms of coverage and evaluation time. Our task is different, we aim to predict the planner performance and not heuristic values for states.

Another work experiments with the representation of planning tasks (Chen, Thiébaux, and Trevizan 2024). After thoroughly analyzing the grounded and lifted representation, they propose novel representations to overcome shortcomings of the previous ones by augmenting the graphs. Our work is tangential to the work on new representations.

3 Preliminaries

The following covers important preliminaries our experiments are based on. This includes GNNs and its variants in Section 3.1. , as well as XGBoost in Section 3.2.

3.1 Graph Neural Networks

Graph Neural Networks are used in numerous domains and capture the given graph structure. Node-level, edge-level or graph-level tasks can be solved. The training incorporates two main steps, namely `aggregate` and `update`. First, the node representations of all neighboring nodes are aggregated according to

$$a_v^{(t+1)} = AGGREGATE^{(t+1)}(h_u^t : u \in N_{(v)}) \quad (1)$$

with the node representations at the t-th layer h_u^t and the set of neighbors of target node v N_v. Thereafter, the node representations are combined and the target node is updated using

$$h_v^{(t+1)} = UPDATE^{(t+1)}(h_v^{(t)}, a_v^{(t+1)}) \quad (2)$$

The main difference between different GNN architectures is the choice of $AGGREGATE^{(t+1)}(.)$ and $UPDATE^{(t+1)}(.)$ (Hamilton 2020). Various methods have been developed, we will cover four of them in the following.

Graph Convolutional Network (GCN) One commonly used GNN architecture is the Graph Convolutional Network (GCN) (Kipf and Welling 2016). Inspired by convolutions used for images, GCN use convolution filters that operate directly on the graph structure. In contrast to images, the neighborhood size of a node within a graph varies. Therefore, a parameter matrix transforms the node representations obtained from the previous layer. The transformed representations are weighted according to the graph adjacency matrix (Kipf and Welling 2016; Ma et al. 2020). When using GCN, an update step is defined as

$$H^{(t+1)} = \sigma(\hat{A}H^{(t)}W^{(t)}) \quad (3)$$

where $H^{(t+1)}$ denotes the matrix with stacked node representations h_v^t, v is a node and t stands for the current layer. σ is an activation function (e.g., ReLU), the adjacency matrix A is normalized to $\hat{A}$ and W is the parameter matrix. GCN uses a shared weight for all edges, making the model relatively simple. In case of more complex graph structures, this approach might be less expressive.

Gated Graph Neural Network (GGNN) Gated Graph Neural Networks (GGNNs) (Li et al. 2016) distinguish themselves from other architectures by incorporating gated

recurrent units (GRUs) (Cho et al. 2014) in their propagation module. The current state of the nodes is updated by the GRU which views the nodes and their representations as a dynamic system. The node representation is updated as follows:

$$h_v^{(t+1)} = GRU(h_v^{(t)}, m_v^{(t+1)}) \tag{4}$$

Here, $m_v^{(t+1)}$ is a message which is aggregated in order to update the state of the node. This formulation allows for deploying selective updates of the node representations depending on the information aggregated from the neighbors. Besides local information, long-range dependencies within the graph can be captured.

Graph Attention Network (GAT) Instead of graph convolutions, Graph Attention Networks (GATs) (Veličković et al. 2018) use masked self-attentional layers. Different weights are assigned to different neighbors when aggregating the neighboring features. This enables the nodes to focus on the more relevant neighbors and helps the model to capture complex relationships in the graph. Another advantage of this approach is that the importance of the neighboring nodes is determined without knowing the graph structure beforehand. The update node representations can be obtained with

$$h_v^{(t+1)} = \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{u\in N_v}\alpha_{vu}^k W_k h_u^t) \tag{5}$$

with the normalized attention coefficient α_{vu}^k for the k-th attention head and the weight matrix W. By employing attention, the model is particularly effective for detecting local dependencies.

Graph Isomporhism Network (GIN) Another variant is the Graph Isomorphism Network (GIN) (Xu et al. 2018) which makes use of multi-layer perceptrons (MLPs) to learn the parameters of the update function. It is inspired by the Weisfeiler-Lehman (WL) graph isomporhism test (Leman and Weisfeiler 1968) which determines how *similar* two graphs are. A node update is defined by

$$h_v^{(t+1)} = MLP^{(t+1)}((1+\epsilon^{(t+1)}) * h_v^{(t)} + \sum_{u\in N(v)} h_u^{(t)}) \tag{6}$$

where ε is a learnable parameter. GIN uses a simple *sum* operator to aggregate the features which makes it computationally efficient. Through the learnable parameter, it is able to adapt well to various graph structures and can effectively capture graph-level features making it a common choice for solving graph-level tasks.

3.2 Extreme Gradient Boosting

XGBoost, short for Extreme Gradient Boosting, is a machine learning technique combining multiple decision trees to create a strong model (Chen and Guestrin 2016). The decision trees often have limited depth and the predictions of each tree are added up to obtain the final prediction. With the help of an objective function, gradient optimization is performed and a learner is fitted with respect to the current predictions. The objective function is minimized during training and comprises the loss function and a regularization term. A series of decision trees is built sequentially, with each tree correcting the errors of the combined model up to that point. The objective function at the t-th iteration is given through

$$\mathcal{L}^{(t)} = \sum_{(i=1)}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{7}$$

where the loss is calculated according to a loss function l based on the prediction $\hat{y_i}$ and the target y_i at the i-th instance. The tree structure f_t which gains the most improvement of the model is added. The regulariztaion term $\Omega(f_t)$ ensures the model is kept as simple as possible. In general, XGBoost combines the strengths of decision trees with an optimization process to efficiently and effectively handle complex datasets.

4 Experiments

In this section, we describe the methodology of our experiments including the datasets, configurations, tasks and setup. This is followed by the presentation and analysis of results and their implications. Lastly, we summarize the main findings and insights.

4.1 Methodology

Dataset and features We use the publicly available dataset consisting of tasks from various International Planning Competitions (IPC), with their graph representations and planners performance data (Ferber et al. 2019). It consists of 2439 data points and provides splits for 10-fold cross validation (2294/145 training+validation/testing). The split can either be random or domain-preserving, making sure that planning problems of the same domain are not split. In the portfolio, there are 17 planners. For each planner, the target value is the time needed for solving the problem. In case a planner exceeds the timeout limit of 1800 seconds, the target value is set to 10,000. There are two representations, the grounded and the lifted representation. While the grounded one is based on SAS+ (Bäckström and Nebel 1995) and directed Problem Description Graphs (PDG) (Pochter, Zohar, and Rosenschein 2011), the lifted representation is based on the Planning Domain Definition Language (PDDL) (McDermott 2000) and directed acyclic Abstract Structure Graphs (ASG) (Sievers et al. 2017). Figure 1 illustrates the number of nodes and edges for each graph in the grounded and lifted representation. We additionally present an aggregated representation of the grounded representation where certain node types and sequences are combined. The grounded graphs consist of up to 100,000 nodes and 800,000 edges. The lifted graphs contain up to 250,000 nodes and 300,000 edges, meaning generally more nodes, but less edges than the grounded representation. This is also reflected by the average node degree which is 12.26 for the grounded graphs and 2.92 for the lifted ones. Most of the aggregated graphs consist of up to 10,000 nodes and 100,000 edges with an average node degree of 9.69. The number of nodes per graph in the grounded and aggregated representation is shown in Figure 2. We can see that the number of nodes and edges per

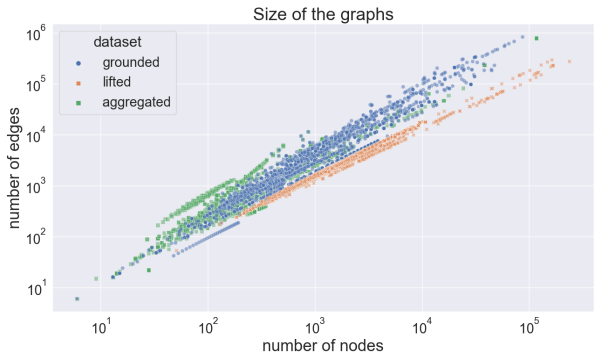


Figure 1: Graph sizes of the grounded and lifted representations

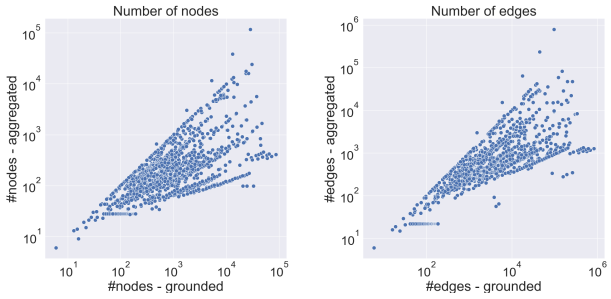


Figure 2: Number of nodes and edges per graph of the grounded and aggregated representation

graph is much smaller when using the aggregated representation.

As node features, the node type is used and encoded into one-hot vectors. For the lifted representation, there are 15 different node types and for the grounded representation, there are 6 individual node types. For instance, the type can depict that the node represents a `constant`, an `action`, or an `effect`. We analyse the average node degree per node type, depicted in Figure 3. For both the grounded and the lifted graphs, while there are high-degree node types with an average degree of up to 70, the majority of node types have an average degree of 3 to 8 for the grounded graphs and 1 to 17 for the lifted ones.

Due to the difference of average node degree per node type, we decide to enhance the initial node features with the node degree. In addition, the type of the neighboring nodes of a target node is important within planning problems to get a more detailed context of the node within the graph. Therefore, we also experiment with the type of the neighbors as node feature.

Tasks and configurations We investigate multiple methods to select a planner. The first method is to predict the time to solve the planning problem and choose the planner with the best predicted time performance, henceforth denoted by *time*. The second is to predict how likely it is for a planner to solve the planning problem within the overall time bound

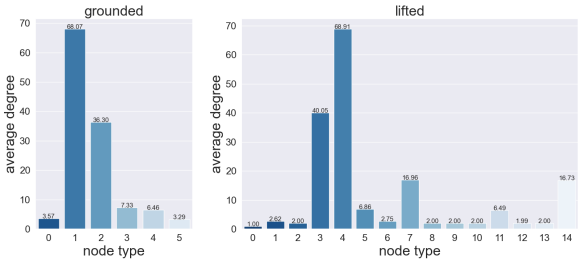


Figure 3: Average node degree per node type for the grounded representation.

of 1800s and choose the planner with the highest probability to solve within the bound, henceforth denoted by *binary*. These two methods were explored in previous work with CNNs (Sievers et al. 2019). Four GNN architectures are used to obtain the predictions, namely GCN (Kipf and Welling 2016), GGNN (Li et al. 2016), GAT (Veličković et al. 2018) and GIN (Xu et al. 2018). We mainly explore two additional node features: the node degree and the type of the neighboring nodes. The third method has not been explored so far in the context of online planner selection. We use the graph representation obtained by the last layer of a GNN as input to train a classification task with a XGBoost model (Chen and Guestrin 2016).

Training details We choose the Deep Graph Library (DGL) (Wang et al. 2019) based on PyTorch (Paszke et al. 2019) as a framework. The configurations are oriented on (Ma et al. 2020) and we did a grid-search to find the best parameters. For training, we use the Adam optimizer (Kingma and Ba 2015) with learning rate 0.001. The number of layers is set to 2, the size of the hidden dimension is 100 and we train the model for 100 epochs. For regression tasks, the MSE loss is used and for the other tasks, we choose the binary cross entropy loss. We run the experiments either on a Nvidia P100 GPU or a Nvidia RTX A600 with two Intel(R) Xeon(R) CPU E5-2640 v4 or two AMD EPYC 7282 CPU nodes, respectively. Depending on the model, training takes 10 to 60 minutes. We set the number of estimators to 500, the maximum depth to 5, and the learning rate to 0.01 when using XGBoost. Training is done with early stopping after 20 epochs. Again, these parameters have been chosen based on a hyper-parameter search.

4.2 Results

First, we present and analyze the results for all four GNN architectures, the two tasks based on probability and time, and the lifted and grounded representation in combination with the random and domain-preserving split. Subsequently, we explore how the results change when adding node features to the data. We conclude by illustrating the accuracy changes when combining a GNN model with XGBoost. It is important to note that we repeat each experiment 10 times and present the corresponding average accuracy with the standard deviation.

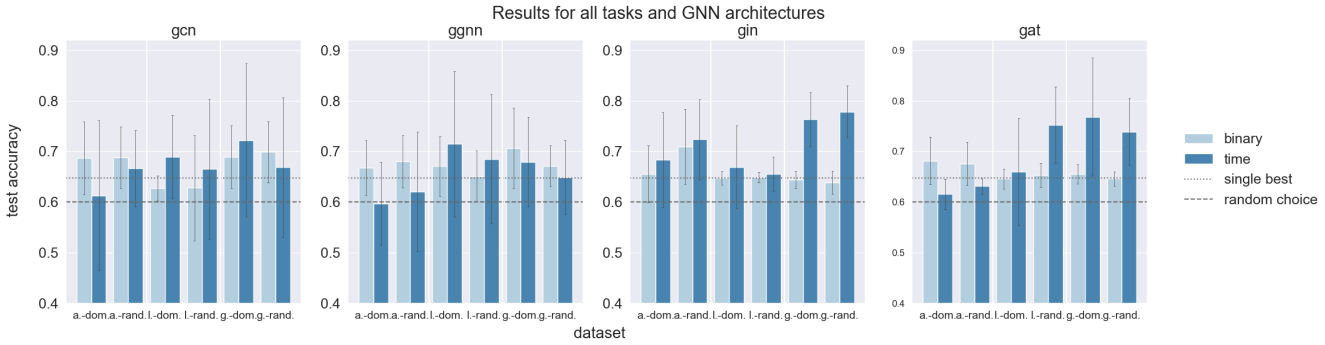
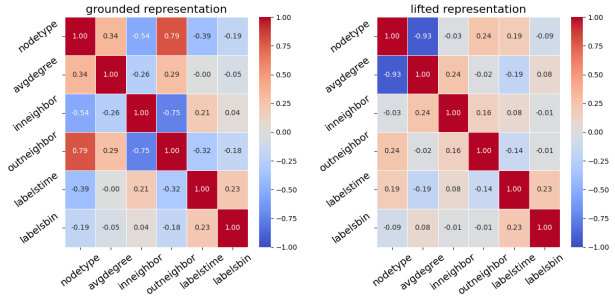


Figure 4: Results for all tasks and all GNN architectures



(a) grounded (b) lifted

Figure 5: Feature correlation matrix for the time- and binary-based task for the grounded and lifted representation

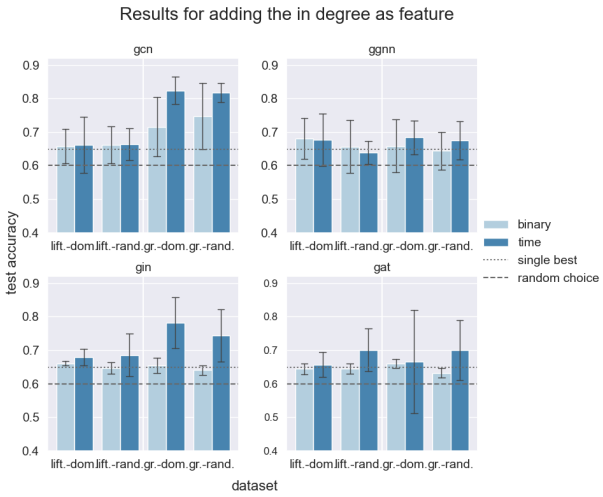


Figure 6: Results for adding the in degree as a node feature

Base experiments This experiment aims to investigate how different GNN architectures perform combined with the two graph representations and to find out how well the planning problem is captured by them. Figure 4 gives an overview of the results. For all four GNN architectures, we plot the accuracy for the lifted and grounded representation as well as the random and domain-preserving split. Overall, the task where we predict the time performs better than the one predicting whether a planning problem is solvable. Figure 5a shows the correlation of the node type to the time- and binary-based labels. For the grounded representation, we can see a higher negative correlation between node type and the time-based labels than the binary-based ones with values of -0.39 and -0.19, respectively. Analogous to the grounded representation, the correlation matrix for the lifted representation (Fig. 5b) shows that the time-based labels also have a higher correlation than the binary-based labels which explains the better performance when predicting the time. For almost all GNN architectures, the grounded representation obtains better results than the lifted one, especially together with the domain-preserving split. This is also reflected in the correlation matrices where the correlation between the node type and the labels is higher for the grounded graphs than the lifted ones (see Fig. 5). As shown in Figure 1, the lifted graphs are smaller and more abstract than the grounded ones. Due to the higher abstraction of the lifted graphs, valuable information is not captured as good as by the grounded representation leading to a performance decrease. This is further supported by our experiments with the aggregated graphs. Here, the accuracy is similar to the one obtained by the lifted representation. When looking at the results, we see that the standard deviation is quite high, especially for the experiments where the time is predicted. Although the best performance can be obtained when predicting the time in most cases, it is also not as stable as predicting a probability. This lies in the nature of the problem. Predicting the actual runtime of a planner is a very difficult task, but it is also a task working very well. To reduce variance, one could incorporate the top-k predicted planners and then refine the predictions for those by using simple ML methods.

In general, GCN and GGNN show a similar performance with an accuracy slightly over 0.7 for the time-based task. GIN and GAT obtain an even higher accuracy around 0.77, again for the time-based task. GIN is based on the WL graph isomorphism test (Section 3.1) to check how similar two graphs are. GIN has been proven to be especially effective for graph-level tasks which is supported by our experiments. GAT emphasizes more important nodes by applying

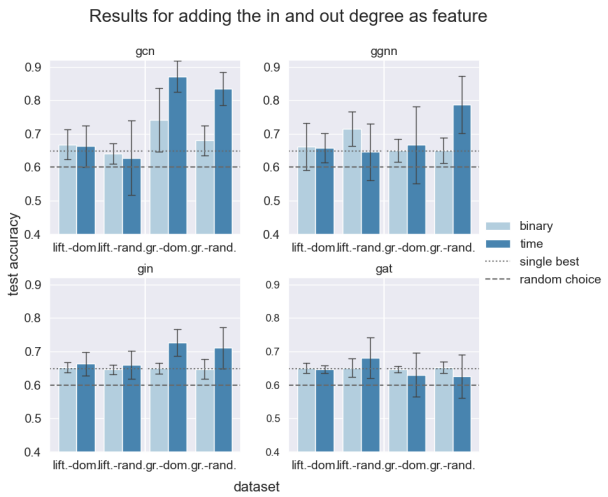


Figure 7: Results for adding the in and out degree as a node feature

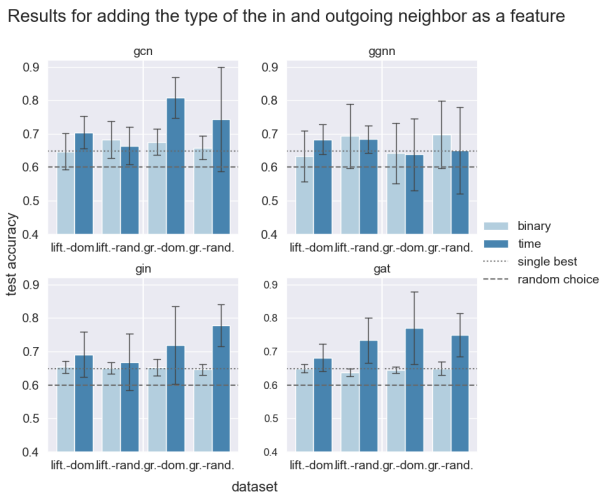


Figure 8: Results for adding the in- and out-going neighbor type

its attention mechanism, while GGNN tries to incorporate far away neighbors. The results illustrate that GGNN is not as effective as GAT at capturing a planning problem. A reason can be that far away neighbors are not as relevant as direct neighbors for the graph-level predictions. To obtain the graph-level predictions, we apply a pooling layer. By including far away neighbors on the node-level, no additional information is gained as the information is pooled in the end anyways.

Enhancing the node features In the following, we enhance the dataset with hand-picked features and investigate how they influence the results. Before diving into the experiments, we investigate the feature correlation. Figure 5 gives an overview of the correlation of node type, average degree, incoming and outgoing neighbor type with the target labels (either time- or solvable-based). We can see that the average degree has a correlation of -0.19 for the lifted representation, however nearly no correlation is detected in the grounded representation. The type of the neighboring nodes has a up to -0.32 for the grounded representation, whereas the correlation in the lifted datset is relatively low. A reason for these differences are the different representations and their characteristics. Another reason could be that we correlate each graph with the output labels and then average the values to be able to use numeric values for the calculations. Thus, this might not be completely representative, but gives a direction of the correlation. To verify this theory, we need to look at the experimental results.

First, we add the node degree as node feature. We investigate adding the in-degree, the out-degree and both. Within a planning problem which is modeled as a directed graph, the subsequent transitions are explicitly known through the formal description of the planning problem. The preceding transitions, on the other hand, are only implicitly known. Therefore, we suspect an improvement of the results especially when incorporating the in-degree of each node. We model the feature as a one-hot encoded vector and append it to the vector describing the node type. As shown in Figure 6, including the in-degree improves the results to an accuracy up to 0.81 when using the GCN with the grounded representation. It is interesting to see that the accuracy with GAT slightly decreases compared to using only the node type as a feature (Fig. 4). As GAT already captures local information very well, redundant information is added when using the node degree as feature leading to a performance decrease. GGNN and GIN do not show any remarkable changes.

Looking at the results with in- and out-degree as node feature (see Fig. 7), the most outstanding result is the increase of accuracy for the GCN rising up to 0.87 with the grounded representation. GGNN also shows some improvement up to an accuracy of 0.8 (grounded representation). Local node characteristics can influence surrounding and far away nodes, as in a planning problem, a transition not only influences the following steps, but also further steps in the future. Therefore, when using GGNN, the node degree as feature enhances the training as it emphasizes sequences and not only local neighborhoods.

For each step in a planning task, it is important to know what type of node follows or precedes another one. Therefore, we include the type of the neighboring nodes as node feature. In Fig. 8, we see can see a higher accuracy for GCN when using the node type only. Although there are some changes compared to the results of the basic experiments, the improvements are not as high as with adding the node degree. This can be explained by how GNNs work in general. In each epoch, each node passes its current information to its neighbors. Among other things, the node type is transmitted. Thus, the neighboring node type is captured anyways, without needing to add it as a feature. By adding it, we emphasize the importance of the node type, but do not necessary gain new information or insights for model training. Further, when looking at the feature correlation matrix (Fig. 5), the correlation of the neighbor type especially for

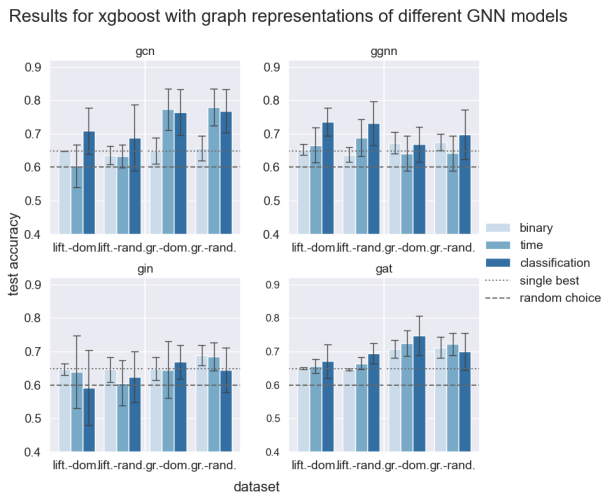


Figure 9: Results for combining GNNs with XGBoost

the lifted representation is much lower compared to the node degree.

Combining GNNs with XGBoost Instead of training a single GNN model, we combine the strengths of GNNs and gradient boosting in the following experiments. We use the GNN to obtain a graph representation as one strength of GNNs is the ability to capture graph structures. This representation is 100-dimensional and is used as input for training an expressive XGBoost model. We train XGBoost models in the following three ways: predict the probability and then pick the planner with the highest probability, predict the time and pick the planner which uses the least amount of time, directly pick one of the 17 planners through multi-class classification. We use the default features provided by the dataset, namely the node type.

In Fig. 9, the accuracy for the experiments with all four GNN architectures and all three tasks is shown. GCN and GAT obtain the overall best results, especially for the time-based and classification-based task with values of slightly under 0.8 for both GNNs. Again, the results with the grounded representation are better than the lifted one which is caused by the higher correlation between the features (e.g., node type) and the labels. When using GCN, predicting the time produces the best results, but the classification task is comparable. In case of GAT, the classification works best. Here, the grounded representation and the domain-preserving split are used. Using a GNN for the classification task could not produce satisfactory results with accuracies of under 0.5. However, the combination of GNNs and XGBoost shows a much better performance and is similar to the results with GNNs only and the time-based task (Fig. 4). Another benefit of combining GNNs with XGBoost is the resource-utilization. While we have to train the GNN-only experiments on GPUs, we do not necessarily need GPU-training for XGBoost. Training the GNN plus XGBoost method on our CPU takes a similar amount of time on the CPU as the GNN-only experiments on our GPU.

		grounded		lifted	
		domain	**random**	**domain**	**random**
CNN-based	**CNN**	-	-	82.1	86.1
Ma et al.	**GCN**	-	-	**85.6**	**87.2**
	GGNN	-	-	76.6	74.4
basic	**GCN**	72.2	66.9	68.9	66.5
	GGNN	68.0	64.9	71.5	68.5
	GIN	76.4	77.9	66.9	65.5
	GAT	76.9	73.9	66.0	75.3
inoutdegree	**GCN**	**87.2**	**83.4**	66.3	62.8
	GGNN	66.6	78.7	65.7	64.6
	GIN	72.7	71.1	66.3	66.1
	GAT	63.0	62.6	64.7	68.1
inoutneighbor	**GCN**	80.7	74.3	70.4	66.4
	GGNN	63.9	65.1	68.3	68.4
	GIN	71.9	77.8	69.1	66.8
	GAT	77.1	75.0	68.2	73.4
xgboost	**GCN**	76.4	76.8	70.8	68.9
	GGNN	66.9	69.7	73.6	73.2
	GIN	67.0	64.5	59.2	62.4
	GAT	74.8	70.1	67.1	69.5

Table 1: Accuracy of related approaches compared to our work (domain and random split)

Comparison to Related Approaches We quantitatively compare the performance of the different approaches to related methods, namely the CNN-based approach by (Katz et al. 2018) and (Sievers et al. 2019) as well as the GNN-based approach by (Ma et al. 2020) (see Table 1). We report our results based on the time-based task. For XGBoost we present the accuracy for the direct classification. To split the data, the domain-preserving and random split are used. We can see that we perform especially well with GCN on the grounded representation and the in- and out-degree as node features. The accuracy lies at 87.2 with the domain-preserving split and 83.4 when using the random split, respectively. For the lifted dataset, one of the highest accuracies is around 71.5 which can be explained through the differences of the representations. As shown above, valuable information is lost when compressing the grounded graphs leading to a decrease in terms of performance.

5 Discussion and Conclusions

5.1 Discussion & Insights

We have seen that GNNs are able to effectively capture planning problems. The best performing way to pick a planner is by predicting the time and then choosing the best planner based on the predictions. Without adding additional features, GIN and GAT prove to work well with an accuracy of up to 0.87. It is shown that by adding simple graph features like the node degree, the accuracy can be improved up to 0.9. The type of the neighbor is also important, but introduces redundant information for some models not reaching the performance obtained with the node degree.

GNNs combined with XGBoost are good at directly picking a planner without having to predict things like the time. Although the accuracy is only comparable to our basic

GNN-only experiments, training XGBoost can be done on CPUs instead of GPUs in a similiar amount of time. Thus, this method produces good results and proves to be more resource efficient. Further, multiple GNNs can be used to add different graph representations which helps to capture a planning problem of different perspectives. The results indicate that there is potential, but still room for improvement.

5.2 Future Work

This section gives an overview of future research directions based on our findings. First, we use quite simple graph-based features for training like the node type and the node degree. To further improve the results, one could include more features like the node centrality, information about clusters or learned node embeddings with tools like node2vec.

Another possible direction could be to use a mixture of experts model (Shazeer et al. 2017) or to train a separate GNN for each of the 17 target planners to make them more specialized and better at deciding whether a planner solves a task or not. Alternatively, the process of selecting a planner could be divided into two phases: first, the top-k planners are chosen and then, the selection is refined by focusing only on the top-k planners instead of all 17.

We use standard GNN architectures for our experiments which work on a variety of tasks. However, they are not adapted to automatic planner selection. Important is the ability to directly select a planner without the need to predict other features like the time first. Ideas of the well-working GNNs in our experiments can be combined with knowledge of how planning problems are formally described and transformed to graphs. For instance, most planning problems are directed acyclic graphs. This knowledge could be a large benefit for the resulting specialized architectures.

When training the GNN models for XGBoost, one could use the enhanced datasets with richer node features. This could help getting a more precise graph representation which would then increase the results of XGBoost. Instead, graph-level features could be included leading to a more concrete description of the graph. Possible graph features are the size of the graph, the clustering coefficient, the majority node type, the diameter or the strongly connected components.

5.3 Conclusions

Automatic planner selection can be tackled using different ML-based approaches, including GNNs. In our work, we investigated the use of different GNNs for choosing a planner for a given planning problem. We explore four GNN architectures, two graph representations, various node features and different ways to pick a planner. Overall, the grounded dataset showed a better performance than the lifted one which is due to the higher feature correlation. Further, predicting the time and then picking a planner based on that led to an accuracy of up to 0.87 when using a GCN model. We analyse the characteristics of the four models to understand what is important for improving automatic planner selection. Further, we investigate the influence of different node features. It is shown that by adding the node degree, the results can easily be improved. Combining GNNs with XGBoost allows to train a classification task where a planner is directly chosen instead of first predicting the time and then make the decision based on that. Our results show a similar accuracy to the ones obtained in our basic GNN-only experiments, but there is no need for GPUs in contrast to GNN training. In addition, we combine multiple graph representations obtained from multiple GNNs to improve the results of XGBoost. To conclude, automatic planner selection with GNNs shows a good performance and can be done in different ways. We obtain the best results with an accuracy of up to 0.87 with the GCN, the grounded representation and the node degree as a feature.

References

Bäckström, C.; and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4): 625–655.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2): 165–204.

Cenamor, I.; De La Rosa, T.; and Fernández, F. 2013. Learning predictive models to configure planning portfolios. In *Proceedings of the 4th workshop on Planning and Learning (ICAPS-PAL 2013)*, 14–22. Citeseer.

Cenamor, I.; De La Rosa, T.; and Fernández, F. 2016. The IBaCoP planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research*, 56: 657–691.

Chakraborti, T.; Isahagian, V.; Khalaf, R.; Khazaeni, Y.; Muthusamy, V.; Rizk, Y.; and Unuvar, M. 2020. From Robotic Process Automation to Intelligent Process Automation: –Emerging Trends–. In *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18*, 215–228. Springer.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2023. Graph Neural Networks and Graph Kernels For Learning Heuristics: Is there a difference? In *NeurIPS 2023 Workshop on Generalization in Planning*.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Cho, K.; van Merrienboer, B.; Gulçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Ferber, P.; Ma, T.; Huo, S.; Chen, J.; and Katz, M. 2019. IPC: A Benchmark Data Set for Learning with Graph-Structured Data. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*.

Ferber, P.; and Seipp, J. 2022. Explainable Planner Selection for Classical Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 9741–9749.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Hamilton, W. L. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, volume 2835.

Karpas, E.; and Magazzeni, D. 2020. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3: 417–439.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, 57–64.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

Leman, A.; and Weisfeiler, B. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9): 12–16.

Li, Y.; Zemel, R.; Brockschmidt, M.; and Tarlow, D. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of ICLR'16*.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

McDermott, D. M. 2000. The 1998 AI planning systems competition. *AI magazine*, 21(2): 35–35.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pochter, N.; Zohar, A.; and Rosenschein, J. 2011. Exploiting problem symmetries in state-based planners. In *Proceedings of the AAAI conference on artificial intelligence*, volume 25, 1004–1009.

Rogalla, A.; Fay, A.; and Niggemann, O. 2018. Improved domain modeling for realistic automated planning and scheduling in discrete manufacturing. In *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, volume 1, 464–471. IEEE.

Russell, S. J.; and Norvig, P. 1995. Artificial intelligence: a modern approach.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, 368–372.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2017. Structural symmetries of the lifted representation of classical planning tasks. In *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning*, 67–74.

Vallati, M. 2012. A guide to portfolio-based planning. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, 57–68. Springer.

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

Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; Xiao, T.; He, T.; Karypis, G.; Li, J.; and Zhang, Z. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*.