Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Beyond Message Passing: Modern GNN Architectures for Online Planner Selection

Jana Vatter[1], Ruben Mayer[2], Hans-Arno Jacobsen[3], Horst Samulowitz[4], Michael Katz[4]
[1]Technical University of Munich, Germany
[2]University of Bayreuth, Germany
[3]University of Toronto, Canada
[4]IBM T. J. Watson Research Center, USA
jana.vatter@tum.de, ruben.mayer@uni-bayreuth.de, jacobsen@eecg.toronto.edu, samulowitz@us.ibm.com, Michael.Katz1@ibm.com

Abstract

As planning is computationally hard, the performance of automated planners varies greatly across planning tasks. Thus, the ability to predict planner performance on a given task is of great importance. While various learning methods have been applied in cost-optimal planning, Graph Neural Networks (GNNs) were found to perform well. However, existing work only explores a limited range of homogeneous GNN architectures and focuses primarily on the model perspective. We address these limitations by approaching the problem from both data and model perspectives. From the data perspective, we analyze the planners' performance data in terms of Shapley values to assess the potential contribution of individual planners to a portfolio. Our insights enable us to effectively reduce the portfolio from 17 to 6 planners, improving practicality and performance. From the model perspective, our work extends previous investigations of homogeneous graphs by modeling planning tasks as heterogeneous graphs and applying the heterogeneous Relational Graph Convolutional Network (RGCN) and Relational Graph Attention Network (RGAT) models. To analyze the problem in more depth, we thoroughly investigate the impact of GNN model, graph representation, node features, and prediction task. Going further, we propose a hybrid approach in which graph representations obtained by GNNs are used as input to a classical machine learning model (XGBoost), resulting in both a more resource-efficient and accurate approach. Our best model (RGCN+XGBoost) achieves 91.7% accuracy, a substantial improvement over previous methods with 87%, while requiring fewer computational resources. Overall, we demonstrate the effectiveness of heterogeneous GNN-based online planner selection methods, opening up new exciting avenues for future research.

Introduction

Given the inherent complexity of classical planning, as described by (Bylander 1994), no single planning algorithm is likely to perform well across all planning tasks. Over the years, a variety of planners have been developed, each tackling different aspects that make planning problems challenging. This observation has motivated the development of portfolio-based approaches (Seipp et al. 2012; Vallati 2012; Cenamor, De La Rosa, and Fernández 2013; Seipp et al. 2015), that attempt to exploit the strengths of a collection of planners either by choosing a single planner or by composing a schedule of planners. Portfolio-based planning can be divided into *offline* and *online* methods. While *offline* methods (Helmert, Röger, and Karpas 2011; Seipp et al. 2012; Büchner et al. 2023) construct a single invocation schedule ahead of time to be used across many domains, *online* methods (Cenamor, De La Rosa, and Fernández 2013; Katz et al. 2018) adapt by learning to select the most suitable single planner or a schedule for each specific planning task.

Among the online methods, a variety of deep learning based approaches were developed (Katz et al. 2018; Sievers et al. 2019). Subsequently, a first attempt was made to use homogeneous Graph Neural Networks (GNNs) to directly learn on planning graphs (Ma et al. 2020). However, this previous work has primarily explored a limited range of homogeneous graphs and corresponding GNN architectures, focusing mainly on the model perspective. In contrast, we argue that planning graphs are inherently heterogeneous: a node can represent a constant, an action, or an effect. Heterogeneous graphs explicitly model multiple types of nodes, and each edge type captures the relation between two node types. In this way, heterogeneous graphs provide a more expressive representation of the portfolio-based planning setting that aligns naturally with the structure of planning problems themselves. In addition, we adopt both data-centric and model-centric perspectives. From the data perspective, we analyze planner performance using Shapley values, which provide insights into the potential contribution of individual planners within a portfolio. We find that some planners contribute minimally to overall performance. We show that Shapley values can be used to effectively optimize the portfolio by reducing the size from 17 to 6 planners, making portfolio-based planning more practical while maintaining competitive performance. From the model perspective, we systematically study the impact of GNN architecture (homogeneous and heterogeneous), graph representation, node features, and prediction task. Going beyond pure GNN-based methods, we propose a simple hybrid between a GNN and a standard classical machine learning classifier, the Extreme Gradient Boosting (XGBoost) (Chen and Guestrin 2016) model, allowing us to switch from predicting planner performance to directly predicting the planner to choose in a

resource-efficient manner.

Our main contributions are as follows:

(1) We go beyond prior work on GNN-based planner selection by modeling planning tasks as heterogeneous graphs and applying two heterogeneous GNN architectures (RGCN, RGAT) to this task. We conduct a comprehensive comparison against four homogeneous GNN architectures (GCN, GGNN, GAT, GIN), demonstrating that heterogeneous GNNs better capture the structural characteristics relevant to automatic planner selection.

(2) We explore two graph representations, lifted and grounded, in combination with enriched node features and explore different ways to pick a planner, including predicting the planner probability to solve a task or its run time. Our experiments show that the time-based objective consistently outperforms the probability-based one, and that incorporating node degrees as features can provide valuable structural information.

(3) We propose a hybrid GNN-XGBoost pipeline where the graph representations obtained by GNNs are used as input for other ML-based methods like XGBoost, resulting in a resource-efficient solution. We improve the accuracy of previous approaches up to 91.7% (from 87%) with the heterogeneous RGCN+XGBoost approach.

(4) We provide a data perspective on portfolio-based planning by performing a Shapley value analysis to provide deeper insights into the experimental results. In addition, we demonstrate that it is possible to reduce the planner portfolio from 17 to 6 planners based on Shapley values, making portfolio-based planning more practical.

Related Work

Cenamor, De La Rosa, and Fernández (2016) proposed one of the first online planner selection methods. Their solution is based on feature representation and uses classical machine learning methods to create a sequential portfolio for non-optimal planning. The corresponding planner *IBaCoP* (Cenamor, De La Rosa, and Fernández 2016) won the International Planning Competition (IPC) 2014 track, showing generalization to previously unseen domains. Sievers et al. (2019) instead makes use of graphical representations of a planning task in combination with deep learning techniques. A planning task is converted to an image, and a CNN is used to make predictions about planner abilities to find optimal plans. The corresponding planner *Delfi* (Katz et al. 2018) won the cost-optimal track of IPC 2018. Their work analyzes the shortcomings of existing CNN-based methods and presents possible solutions. The authors identify the need for methods working well with non-IID (independent and identically distributed) data and explore alternative network architectures. In contrast to CNN-based approaches, we leverage homogeneous and heterogeneous GNNs in combination with a classical machine learning method to solve the automatic planner selection task.

The work most similar to ours is by Ma et al. (2020) using GCN (Graph Convolutional Network) and GGNN (Gated Graph Neural Network) for automatic planner selection. Their experiments with the IPC dataset show that their graph-based approach outperforms previous image-based ones by effectively capturing structural information in planning graphs and addressing the lack of node-level details. The authors show that the lifted representation is favored over the grounded one as it produces more consistent results. We distinguish ourselves by not only including two GNN architectures, but a set of four representative homogeneous architectures and two heterogeneous architectures. In addition, we explore the use of node features for GNN training and combine GNNs with a classical ML-based method to achieve a more resource-efficient approach.

Ferber and Seipp (2022) explore graph features for simple machine learning models such as linear regression or random forests, analyzing their importance for the training process. We specifically explore the impact of node features on GNN model prediction accuracy.

Chen, Trevizan, and Thiébaux (2023) use the Weisfeiler-Lehman test for learning heuristics for planning, showing better performance in terms of coverage and evaluation time than of GNN-based methods. Our objective differs as we predict the end-to-end performance of a planner.

Chen, Thiébaux, and Trevizan (2024) propose augmented graph representations to address limitations of existing grounded and lifted representations. Our work is tangential to the work on new representations.

Overview of Models and Methods

We cover key models and methods, including GNN architectures, Extreme Gradient Boosting, and Shapley values.

Graph Neural Networks

Graph Neural Networks are used in numerous domains and capture a given graph structure. The graphs can be homogeneous (one node and edge type) or heterogeneous (multiple node and edge types). Node-level, edge-level, or graph-level tasks can be solved. The forward pass during training incorporates two main steps, namely `aggregate` and `update`. First, the node representations of all neighboring nodes are aggregated according to $a_v^{(t+1)} = AGGREGATE^{(t+1)}(h_u^t : u \in \mathcal{N}_v)$ with the node representations at the t-th layer h_u^t and the set of neighbors $\mathcal{N}_v$ of target node v. Thereafter, each node's representation is updated by combining its previous embedding with the aggregated representations of its neighboring nodes using $h_v^{(t+1)} = UPDATE^{(t+1)}(h_v^{(t)}, a_v^{(t+1)})$. This is followed by a backward pass to update the GNN parameters. The main difference between different GNN architectures is the choice of $AGGREGATE^{(t+1)}(.)$ and $UPDATE^{(t+1)}(.)$ (Hamilton 2020). We cover four of the existing methods for homogeneous graphs and two for heterogeneous graphs.

Homogeneous GNNs In the following, we give an overview of the homogeneous GNN architectures.

GCN: One commonly used GNN architecture is the Graph Convolutional Network (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 $H^{(t+1)} = \sigma(\hat{A}H^{(t)}W^{(t)})$ 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. For complex graph structures, this approach might be less expressive.

GGNN: Gated Graph Neural Networks (Li et al. 2016) distinguish themselves from other architectures by incorporating gated recurrent units (GRUs) (Cho et al. 2014) in their update function. The current state of the nodes is updated by the GRU viewing the nodes and their representations as a dynamic system. The node representation is updated as follows: The message $m_v^{(t+1)}$ is aggregated to update the state of the node and updated via $h_v^{(t+1)} = GRU(h_v^{(t)}, m_v^{(t+1)})$. This helps to not only capture local information, but also long-range dependencies within the graph.

GAT: Instead of graph convolutions, Graph Attention Networks (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 updated node representations can be obtained with $h_v^{(t+1)} = \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{u\in\mathcal{N}_v}\alpha_{vu}^{k(t)}W_k^{(t)}h_u^{(t)})$ with the normalized attention coefficient α_{vu}^k for the k-th attention head and the weight matrix W. Attention is shown particularly effective for detecting local dependencies.

GIN: Another variant is the Graph Isomorphism Network (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 isomorphism test (Lehman and Weisfeiler 1968) which determines how *similar* two graphs are. A node update is defined by $h_v^{(t+1)} = MLP^{(t+1)}((1+\epsilon^{(t+1)}) * h_v^{(t)} + \sum_{u\in\mathcal{N}_v} h_u^{(t)})$ where ε is a learnable parameter. GIN uses a simple *sum* operator to aggregate the features which makes it highly expressive and 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.

Heterogeneous GNNs Heterogeneous GNNs not only capture the graph structure, but also different node and relation types. We present the foundations of RGCN and RGAT.

RGCN: Relational Graph Convolutional Networks (RGCNs) (Schlichtkrull et al. 2018) extend GNNs to heterogeneous graphs containing multiple node and edge types. While homogeneous GNNs apply uniform transformations across all edges, RGCNs learn distinct weight matrices for each relation type. The update step is adapted as follows: $h_v^{(t+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_v^r}\frac{1}{c_{v,r}}W_r^{(t)}h_u^{(t)} + W_0^{(t)}h_v^{(t)}\right)$ where $r \in \mathcal{R}$ is a relation, neighbors of v connected via relation r are denoted as $\mathcal{N}_v^r$, and $c_{v,r}$ is a constant. $W_r^{(t)}$ is the weight matrix of the relation and $W_0^{(t)}$ for a self-loop.

RGAT: Similarly to RGCNs, Relational Graph Attention Networks (RGATs) (Wang et al. 2019) operate on heterogeneous graphs. In contrast to GAT, attention weights are not only assigned to different neighbors, but relation-specific attention weights are used. For each relation type, separate attention weights are learned. The node update can be expressed as $h_v^{(t+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_v^r}\alpha_{vu}^{kr(t)}W_k^{r(t)}h_u^{(t)}\right)$ where $r \in R$ denotes a relation type, $\mathcal{N}_v^r$ represents neighbors connected via relation r, and $\alpha_{vu}^{kr(t)}W_{kr}^{(t)}$ is the attention coefficient computed for the k-th attention head and relation type r. This mechanism enables the model to learn distinct feature transformations for different semantic relationships while dynamically determining neighbor importance.

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost), is a machine learning technique combining multiple decision trees to create a strong model (Chen and Guestrin 2016). It uses gradient-based optimization to minimize an objective function comprising a loss term and a regularization term, allowing for accurate yet simple models. At iteration t, the objective function is $\mathcal{L}^{(t)} = \sum_{(i=1)}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$ 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 task. The tree structure f_t which gains the most improvement of the model is added. The regularization 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.

Shapley values

Shapley values provide insights into individual components' contributions to a portfolio. Comparing model predictions with Shapley values of individual planners can help to gain more detailed explanations on how and why the different models perform the way they do. The Shapley value of $i \in N$ where N is the set of n components is *an average of average marginal contributions over the possible coalitions of each size* (Fréchette et al. 2016). It is given through

$$\phi_i = \frac{1}{n}\sum_{c=0}^{n-1}\frac{1}{\binom{n-1}{c}}\sum_{C\subseteq\mathcal{N}\setminus\{i\}:|C|=c}(v(C\cup\{i\}) - v(C))$$

with $v(C\cup\{i\}) - v(C)$ being the marginal contribution to a coalition $C \subseteq \mathcal{N}$. The standalone contribution of i to a set of planners A on a planning task $x \in X$ is denoted by

$$contr_s(i, A) = \begin{cases} 0 & \text{task x not solved by i,} \\ 1 + \frac{c-t}{|X|*c*|A|+1} & \text{otherwise.} \end{cases}$$

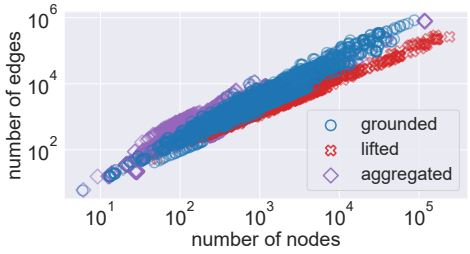


Figure 1: Graph sizes of various representations

The standalone performance, marginal contribution, and Shapley value measures differ in how the planners are rewarded or penalized. The standalone performance only regards the planner itself and does not consider the relation to other planners in the portfolio. Planners with a small, but strong expertise usually are penalized. The marginal contribution takes the overall portfolio into account and puts the planner in perspective. However, correlated planners are now penalized which could lead to certain planners falsely regarded as unimportant. The Shapley value combines both the standalone performance and the marginal contribution.

Experiments

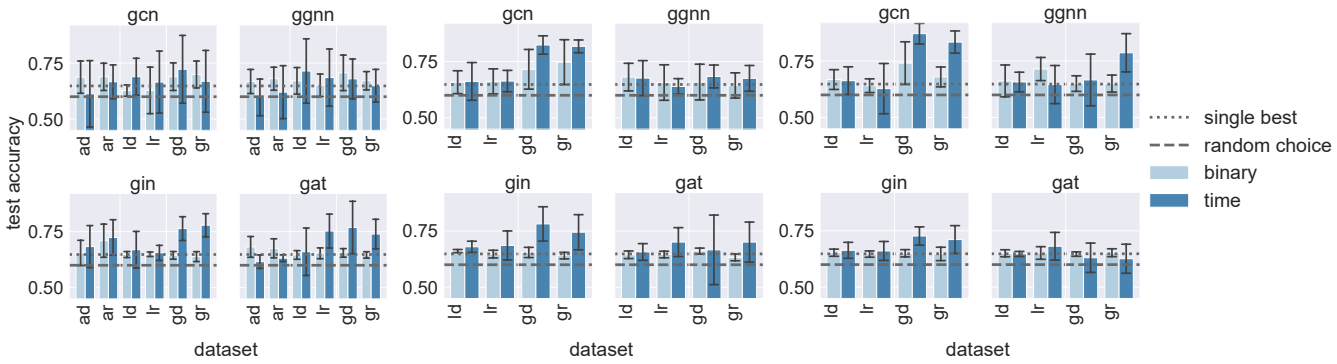
Methodology

Dataset and features We use the publicly available dataset consisting of homogeneous graph representations and planner-performance data covering tasks from various IPCs (Ferber et al. 2019). The dataset consists of 2,439 data points and provides predefined splits for 10-fold cross validation (2,294/145 training+validation/testing). Splits can be either random or domain-preserving, ensuring that planning tasks from the same domain are not split. In the portfolio, there are 17 planners with the target value being the time required to solve each task. In case a planner exceeds the timeout limit of 1,800 seconds, its target value is set to 10,000. Two graph representations are provided: grounded and lifted. The grounded representation, Problem Description Graphs (PDG) (Pochter, Zohar, and Rosenschein 2011), is based on SAS+ (Bäckström and Nebel 1995), whereas the lifted representation, directed acyclic Abstract Structure Graphs (ASG) (Sievers et al. 2017) is based on the Planning Domain Definition Language (PDDL) (McDermott 2000). Grounded graphs consist of up to 100K nodes and 800K edges, while lifted graphs contain up to 250K nodes and 300K edges, meaning generally more nodes, but fewer edges (Figure 1). This is also reflected by the average node degree which is 12.26 for the grounded graphs and 2.92 for the lifted ones. We additionally introduce aggregated graphs, in which certain node types and sequences are combined, reducing the graph size to typically up to 10K nodes and 100K edges with an average degree of 9.69.

In homogeneous graphs, node type information is encoded as one-hot vectors and used as node features. The lifted graphs contain 15 node types, and the grounded graphs contain 6. Node types include, for example, `constant`, `action`, or `effect`. We analyze the average node degree per node type. In both grounded and lifted graphs, some node types exhibit high average degrees (up to 70), but most lie between 3–8 in the grounded graphs and 1–17 in the lifted graphs. In the grounded graphs, around 77% of nodes are of type 3, followed by around 8% each for types 1 and 5. In lifted graphs, type 1 dominates (63%) followed by type 6 and 8 with 12% and 10%, respectively. Due to these variations, we enhance the initial node features with the node degree. Because the types of neighboring nodes provide critical contextual information in planning tasks, we also experiment with incorporating the types of neighboring nodes as node features. In contrast to homogeneous graphs, heterogeneous graphs inherently encode multiple node types. The type is represented directly by the node category itself, rather than needing to be encoded within the node features.

Prediction tasks and configurations We investigate three methods for planner selection. The first predicts the time required for each planner to solve a given task and selects the planner with the lowest predicted runtime, henceforth denoted by *time*. The second predicts whether a planner will solve the task within the overall time limit of 1,800 seconds and selects the planner with the highest probability, henceforth denoted by *binary*. Both approaches were previously explored using CNNs (Sievers et al. 2019). We evaluate four homogeneous GNN architectures (GCN, GGNN, GAT, GIN) and two heterogeneous ones (RGCN, RGAT). We also explore additional node features, such as node degree and neighboring node types, to capture structural and semantic context. A third method, not previously explored for online planner selection, is inspired by Loreggia et al. (2016). We use the graph representations from the final GNN layer as input to an XGBoost classifier (Chen and Guestrin 2016). We additionally include graph-level statistics such as average node degree, clustering coefficient, and density. The clustering coefficient indicates how clustered a graph is, the density depicts the number of edges in relation to the maximum number of possible edges.

Training details The Deep Graph Library (DGL) (Wang 2019) is built on PyTorch (Paszke et al. 2019) and serves as the framework for our experiments. Our configurations follow Ma et al. (2020) with an additional grid-search to find the best parameters (learning rates [0.01,0.001,0.0001], layers [2,3], hidden dimensions [64,100,128,256]). To ensure fair comparison across architectures, we select a shared configuration competitive across models rather than tuning each independently. While this may affect absolute performance, we do not expect it to alter our qualitative conclusions. For training, we employ the Adam optimizer (Kingma and Ba 2015) with learning rate 0.001. The final configuration based on hyperparameter search uses 2 layers, a hidden dimension of 100, and 100 training epochs. MSE loss is used for time-based objectives and binary cross entropy loss for binary objectives. Training uses early stopping after 10 epochs. Experiments are run on Nvidia P100 GPU and Nvidia RTX A600 with two Intel(R) Xeon(R) CPU E5-2640 v4 or two AMD EPYC 7282 CPU nodes, respectively. Training time ranges from 10 to 60 min depending on the model. For GNN+XGBoost approach, the graph representa-



(a) Base experiments (b) In-degree as a node feature (c) In- and out-degree as a node feature

Figure 2: Accuracy results for the base experiments and enhancing the node features (in- and out-degree); a: aggregated, l: lifted, g: grounded, d: domain, r: random

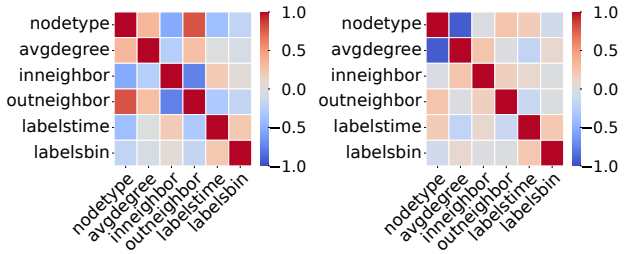
tion of the GNN is obtained after 5 epochs. After hyperparameter tuning, the XGBoost model is configured with 500 estimators, a maximum depth of 5, and a learning rate of 0.01. Early stopping is done after 20 epochs. All results are reported as the mean over 10 independent runs.

Results

First, we present the results for the homogeneous GNN architectures across the time-based and binary objectives, using both lifted and grounded representations under random and domain-preserving splits. Subsequently, node feature correlation and the effect of enhanced node features is examined, such as node degree and neighboring node types. We then evaluate the performance of heterogeneous GNN architectures and study how results change when combining GNNs with XGBoost. Finally, the portfolio and model behavior is analyzed using Shapley values. We repeat each experiment 10 times and present the average accuracy with the standard deviation. In our figures, *random choice* refers to selecting a planner at random, whereas *single best* denotes the planner that solves the largest number of planning tasks.

Base experiments For all four homogeneous GNN architectures, we plot the accuracy for the lifted and grounded representations as well as the random and domain-preserving splits (Figure 2a). Overall, the time-based objective outperforms the binary one, as shown by the top results across all datasets and architectures (Figure 2). The time objective is more informative than the binary classification task, resulting in more accurate GNN updates.

For most architectures, the grounded graphs outperform the lifted ones, especially with domain-preserving splits. Due to the higher abstraction of the lifted graphs (Figure 1), valuable information is not captured as well as in the grounded representation, leading to a performance decrease. To support this hypothesis, we perform the base experiments with the aggregated graphs, which are compressed grounded graphs. Here, the accuracy is similar to that obtained with the lifted representation. One aspect to note is that the standard deviation is quite high. To reduce variance, one could select



(a) grounded (b) lifted

Figure 3: Feature correlation matrix for the time- and binary-based objective for the grounded and lifted representation

the top-k predicted planners and refine their predictions by using standard ML methods.

In general, GCN and GGNN show similar performance with an accuracy slightly higher than 70% for the time-based objective. GIN and GAT obtain an even higher accuracy around 77%, again for the time-based objective. As supported by our experiments, GIN performs particularly well on graph-level tasks by incorporating ideas from the WL graph isomorphism test. GAT emphasizes more important nodes by applying its attention mechanism, while GGNN tries to incorporate distant neighbors. The results illustrate that GGNN is not as effective as GAT in capturing planning tasks. A possible explanation is that distant neighbors are not as relevant as direct neighbors for graph-level predictions. To obtain the graph-level predictions, we apply a pooling layer. As the output vector of the pooling operation typically is smaller than the number of nodes, information is lost. This makes it essential that GNN's intermediate layers already encode long-range relationships (Rampášek et al. 2022; Alon and Yahav 2021).

Node feature correlation Figure 3 provides an overview of the correlation between various node features (node type, average degree, incoming/outgoing neighbor types) and the target labels (time- or binary-based). These correlations help explain the performance differences observed in the base ex-

periments, where the time-based objective consistently outperformed the binary-based objective. For the grounded representation, there is a higher negative correlation between node type and the time-based labels (-0.39) than the binary-based labels (-0.19) (Figure 3a). Similarly, in lifted representation time-based labels show higher correlation than binary-based labels (Figure 3b). Overall, the correlation between node type and labels is consistently higher for grounded graphs than for lifted ones across both label types.

We observe that the average degree has a correlation of -0.19 for the lifted representation, while nearly no correlation is detected in the grounded representation. The type of neighboring nodes has a correlation of up to -0.32 for the grounded representation, whereas the correlation in the lifted dataset is relatively low due to the different characteristics and focus of each representation. These correlation patterns suggest that node degree and neighbor type information could be valuable additions to the basic node features, motivating our investigation of enhanced feature sets.

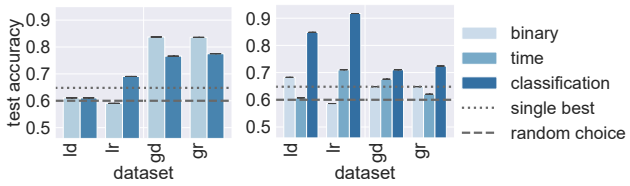
Enhancing the node features Based on the correlation analysis, we enhance the dataset with hand-crafted features to investigate how they influence model performance.

First, we add the in-degree, out-degree, and both to the node type feature. Including the in-degree improves GCN accuracy to up to 81% on the grounded representation (Figure 2b). Interestingly, GAT accuracy slightly decreases compared to using only the node type as a feature (Figure 2a). As GAT already captures local information well, adding redundant information (node degree) leads to a performance decrease. GGNN and GIN do not show remarkable changes. Looking at the results with in- and out-degree as node feature (Figure 2c), the most outstanding result is the increase in accuracy for the GCN, rising up to 87% with the grounded representation. Local node characteristics can influence surrounding and distant nodes, and in a planning task, a transition not only influences subsequent steps, but also those further in the future. Thus, GGNN can capture sequences, while node degree emphasizes local structural information.

In typed graphs, knowing what type of node follows or precedes another one is important. Therefore, the type of neighboring nodes is included as node feature (Table 1). Although there are some changes compared to the results of the base experiments, the improvements are not as high as with adding the node degree. This is because GNNs inherently propagate node information, including node type, during message passing. Thus, explicitly including neighbor types emphasizes, but does not significantly expand, the information or insights for model training.

These results suggest that richer features reduce architectural differences, as explicit structural signals diminish the need for complex aggregation. Performance is thus a joint effect of representation, architecture, and feature design.

Heterogeneous GNNs Table 1 provides an overview of the results with heterogeneous RGCN and RGAT and compares them to the homogeneous architectures. The results with RGCN are particularly promising. Using RGCN leads to similar results for the lifted representation as observed with homogeneous GNNs. Heterogeneous graphs are less



(a) RGCN (b) RGCN with XGBoost

Figure 4: Results for (a) RGCN and (b) combining RGCN with XGBoost (avg. node degree as graph feature)

model	mode	grounded		lifted	
		domain	random	domain	random
GCN	basic	72.2	66.9	68.9	66.5
	degree	**87.2**	83.4	66.3	62.8
	neighbor	80.7	74.3	70.4	66.4
	xgboost (best)	76.6	76.8	70.8	69.6
GGNN	basic	68.0	64.9	71.5	68.5
	degree	66.6	78.7	65.7	64.6
	neighbor	63.9	65.1	68.3	68.4
	xgboost (best)	71.3	74.1	74.4	75.7
GIN	basic	76.4	77.9	66.9	65.5
	degree	72.7	71.1	66.3	66.1
	neighbor	71.9	77.8	69.1	66.8
	xgboost (best)	69.8	71.6	68.2	67.4
GAT	basic	76.9	73.9	66.0	75.3
	degree	63.0	62.6	64.7	68.1
	neighbor	77.1	75.0	68.2	73.4
	xgboost (best)	74.8	79.2	74.6	75.7
RGCN	basic	83.7	**83.6**	61.1	69.1
	degree	76.6	72.5	64.6	72.6
	neighbor	76.6	77.6	61.1	71.6
	xgboost (best)	75.2	75.2	**84.8**	**91.7**
RGAT	basic	63.44	69.6	64.37	65.56
	degree	61.1	67.9	61.0	61.1
	neighbor	61.0	70.6	61.1	61.0
	xgboost (best)	75.2	76.1	79.5	78.4

Table 1: Accuracy of our approaches with different node features and best XGBoost-based results

abstract and exploit the graph structure better compared to homogeneous ones, leading to a more detailed representation of the tasks. In addition, heterogeneous GNNs are able to distinguish between different relation types, which helps to better capture the essence of the different planning graphs. This shows that heterogeneous graphs can and should be leveraged for solving tasks in the planning domain.

Combining GNNs with XGBoost We propose combining the strengths of GNNs and gradient boosting. A GNN is used to obtain a 100-dimensional graph representation which then serves as input for training an expressive XGBoost model. We explore three use cases: (a) the binary objective, (b) the time-based objective, and (c) directly picking one of the 17 planners through multi-class classification. Node features include the default ones, the in- and out-degree, and the in- and out-going neighbor types. After obtaining the graph representation from the GNN, another set of experiments ap-

pends hand-crafted graph features, including the average out degree, the average clustering coefficient, and the density. This extended representation is then fed into the XGBoost model. Table 1 shows the best performing XGBoost-based result per architecture.

Experiments with only using a GNN for direct classification could not produce satisfactory results with accuracies below 50%. For this reason, our GNN only experiments focus on binary- and time-based objectives. However, the combination of GNNs and XGBoost shows much better performance for direct classification and is similar to the results with GNNs only and the time-based objective (Figure 2). The heterogeneous RGCN obtains the highest accuracy 91.7% with the lifted representation, random split, classification task, basic node features, and out-degree as graph feature (Figure 4, Table 1). RGCN is specialized in capturing different node and relation types which benefits the lifted representation with a larger variety of node types (15) compared to the grounded representation (6). The average node degree as a graph feature further provides valuable graph-level information that benefits the XGBoost model. This illustrates the effectiveness of RGCN and its combination with XGBoost for direct classification, while also demonstrating the value of hybrid approaches for planner selection.

Another benefit of combining GNNs with XGBoost is resource utilization. While we have to train the GNN-only experiments on GPUs, we do not 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.

Contrary to our observations above, the performance on the lifted graphs is generally better than the grounded ones when using classification and three graph features alone without appending them to a representation obtained by a GNN (grounded: 75.9%, lifted: 79.3%). Due to the large graph sizes in the grounded dataset, describing a graph with only three features is not sufficient. The bigger the graphs become, the more variations exist, even when the values for clustering coefficient, out degree, and density remain the same. While graph-level statistics capture important information, they are most effective when combined with representations learned by GNNs. This highlights the strength of GNNs being able to extract meaningful graph representations of large graphs.

Analysis of Planner Portfolio using Shapley values Figure 5 illustrates the standalone performance, Shapley value, and marginal contribution of the IPC Graphs dataset (Ferber et al. 2019). The best performing planner is `SymBA` for both Shapley and marginal categories. The values for marginal contribution are extremely small, except for `SymBA`. This indicates that many planners are regarded as similar and thus penalized. However, the Shapley values highlight that the planners are still important for the portfolio and their contribution should not be neglected. Looking at, for example, `h2-DKS-LMcut`, the standalone performance and marginal contribution are lower than the Shapley value. With the Shapley value, we can see that its importance is much higher than previously anticipated. As a model-independent

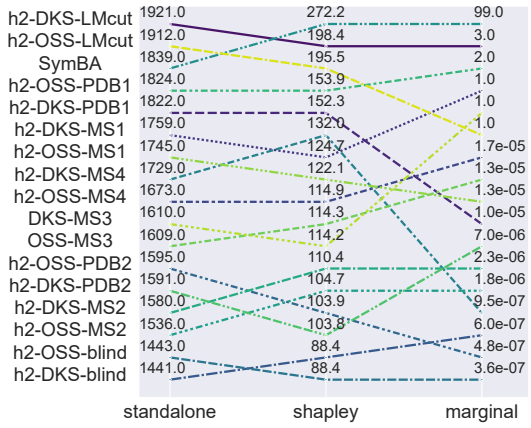


Figure 5: Per-planner contributions to the portfolio

attribution method, Shapley values are not limited to this planner portfolio and can generalize to other portfolios or similar planning task distributions.

To obtain deeper insights how the different models perform, we analyze the performance of the planners by establishing connections between the model predictions and Shapley values. Most of the lower performing models usually predict the same one or two planners for the different planning tasks. In the training set, each planner on its own solves 64%-83% of tasks. However, in the test set, performance varies considerably, with some planners only solving 34% of tasks while others solve up to 83%. This discrepancy means that choosing one planner for all tasks is much more penalized in the test set than the training set. Consequently, some models fail to learn to choose different planners for different tasks, but only choose one or two for all tasks. In such cases, it can happen that a planner solves 70% of tasks in the training set but only 34% of tasks in the test set, leading to a performance decrease. When analyzing the best-performing models, we observe that they choose a variety of planners, their predictions are not limited to one or two. Additionally, the most frequently chosen planners typically reflect the highest-performing planners in terms of Shapley values (Figure 5), meaning they are able to assess both the standalone performance and marginal contribution to the overall portfolio. Table 2 shows the performance of three individual models trained with the in- and out-degree as node feature with the GCN on the grounded graphs using the domain-preserving split. Some models only predict planners with a high marginal contribution, leading to poor performance on the test set as the marginal contribution tends to penalize correlated but important planners.

Shapley values can also be used to optimize the portfolio. Figure 5 shows that the marginal contribution of the first 6 planners is higher compared to the other ones. Based on this observation, we experiment with restricting the portfolio to only the top 6 planners instead of all 17. We evaluate both the basic GCN-only setup and the GCN+XGBoost approach, comparing the full portfolio with the top 6 planners based on Shapley values and marginal contribution.

h2-DKS-LMcut	h2-DKS-PDB1	SymBA	h2-OSS-PDB1	h2-OSS-LMcut	accuracy
3	47	65	29	1	**90.6**
28	41	65	11	0	**86.6**
23	42	64	16	0	**89.1**

Table 2: Predictions of three selected models (inoutdegree, GCN, grounded, domain) on 5 most freq. selected planners.

			all	shapley	marginal
GCN	**grounded**	**domain**	72.2	77.1	**81.1**
		random	66.9	69.9	**70.4**
	lifted	**domain**	68.9	62.1	**75.3**
		random	66.5	62.1	**75.4**
GCN + XGBoost	**grounded**	**domain**	76.4	**79.3**	75.2
		random	76.8	**87.6**	80.7
	lifted	**domain**	70.8	**73.8**	73.1
		random	68.9	74.5	**75.2**

Table 3: Accuracy for the portfolio with all 17 planners, and top 6 planners based on Shapley and marginal values

The results in Table 3 show that for the GCN-only setup, the marginal-based top 6 planners lead to improved performance. The largest improvement occurs with the grounded representation and domain split, where the accuracy improves from 72.2 to 81.1. For the XGBoost-based experiments, the Shapley-based top 6 planners usually outperform the other configurations.

Comparison to Related Approaches We quantitatively compare the performance of the different approaches with related methods, namely the CNN-based approach of (Katz et al. 2018) and (Sievers et al. 2019) as well as the GNN-based approach of (Ma et al. 2020). Due to the unavailability of the original code, we re-implemented the system based on the descriptions in (Ma et al. 2020), which correspond to the GCN and GGNN. The results in Table 1 are the outcome of our implementation and do not match the reported in the literature (85.6% (domain) and 87.2% (random) with GCN). The CNN-based method reported accuracy is 82.1% (domain) and 86.1% (random) on the lifted set. We surpass these results with a top accuracy of 91.7% when using our heterogeneous RGCN+XGBoost approach with out-degree as graph feature.

Conclusions

Automatic planner selection can be tackled using different ML-based approaches, including GNNs. In our work, we introduce heterogeneous GNNs to the selection task, demonstrating that respecting the typed structure of planning graphs significantly improves performance. We provide a data and model view on the challenge of choosing a single planner for a given planning task. In addition to the portfolio analysis using Shapley values, we investigate the use of different GNNs for choosing a planner. We explore four homogeneous and two heterogeneous GNN architectures, two graph representations, various node features and different ways to pick a planner. Overall, the grounded dataset shows a better performance than the lifted one because of a higher feature correlation. In addition, predicting run time and choosing a planner based on the best predicted run time led to an accuracy of up to 87% when using a GCN model. We analyze the characteristics of the four models to understand what is important for improving automatic planner selection. Furthermore, we investigate the influence of different node features. We show that adding the node degree can improve the prediction accuracy. Combining GNNs with ML models such as XGBoost allows to effectively train a classification task where a planner is directly chosen without the need to first predict its run time. The heterogeneous RGCN+XGBoost method achieves a top accuracy of 91.7%. Unlike prior GNN-only methods that require GPU training, our GNN with XGBoost pipeline achieves similar accuracy while enabling CPU-based training and fast run times. Finally, using this method allows us to combine multiple graph representations obtained from multiple GNNs to improve the results of XGBoost. To conclude, automatic planner selection with GNNs shows a promising performance and allows for a wide variety of approaches. We obtain the best results with the accuracy of up to 91.7% with the heterogeneous RGCN model followed by XGBoost, the lifted representation and the average node degree as graph feature.

Our work opens up many possible directions for future work, here, we elaborate on four possible directions.

Graph features: We used simple graph features for training: node type, node degree, clustering coefficient, and density, showing prediction improvement. To further improve the prediction accuracy, more complex features might be needed. Examples of such features include the node centrality, information about clusters or learned node embeddings with methods like node2vec (Grover and Leskovec 2016).

Mixture of experts: Another direction could be to use a mixture of experts model (Shazeer et al. 2017) or to train a separate GNN for each of the portfolio component planners, predicting whether a planner should be considered for the task at all. Selecting a planner could be divided into two phases: first, a subset of planners are chosen based on the first prediction, then the selection is refined by focusing only on this subset instead of the entire set of planner candidates.

Specialized architectures: We use standard GNN architectures for our experiments working on a variety of tasks. However, they are not adapted to automatic planner selection. Our experiment with XGBoost shows the benefit of being able to directly predict the planner to choose instead of predicting planner performance and choosing based on that prediction. Further, these ideas can be combined with specialized architectures constructed for planning tasks, as these have a specific structure, e.g., PDGs are bi-partite graphs, ASGs are cyclic.

Planners space exploration: Shapley values analysis presents us with interesting opportunities. Armed with a tool that can evaluate the relative contribution of a planner to the portfolio, we can attempt at automatically enrich the set of planners in a portfolio. We can explore large spaces of planning algorithms, much like the offline portfolio methods (Büchner et al. 2023) and find the meaningful subsets.

Acknowledgments

This work has been partially funded by the European Union through the Horizon Europe CLOUDSTARS project (101086248).

References

Alon, U.; and Yahav, E. 2021. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*.

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

Büchner, C.; Christen, R.; Corrêa, A. B.; Eriksson, S.; Ferber, P.; Seipp, J.; and Sievers, S. 2023. Fast Downward Stone Soup 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

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

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

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

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *Proc. of the AAAI Conf. on AI*, 20078–20086.

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 *Proc. of the 22nd ACM SIGKDD Int. Conf.*, 785–794.

Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proc. of the 2014 Conf. on 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 *Proc. of the AAAI Conf. on AI*, 9741–9749.

Fréchette, A.; Kotthoff, L.; Michalak, T.; Rahwan, T.; Hoos, H.; and Leyton-Brown, K. 2016. Using the shapley value to analyze algorithm portfolios. In *Proc. of the AAAI Conf. on AI*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proc. of the 22nd ACM SIGKDD Int. Conf.*, 855–864.

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 (ICAPS-PAL))*, volume 2835.

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 *ICLR*.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Lehman, 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 *Proc. of ICLR'16*.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. 2016. Deep learning for algorithm portfolios. In *Proceedings of the aaai conference on artificial intelligence*, volume 30.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *Proc. of the 34th AAAI Conf. on AI*.

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. *Adv. in NeurIPS*, 32.

Pochter, N.; Zohar, A.; and Rosenschein, J. 2011. Exploiting problem symmetries in state-based planners. In *Proc. of the AAAI Conf. on AI*, 1004–1009.

Rampášek, L.; Galkin, M.; Dwivedi, V. P.; Luu, A. T.; Wolf, G.; and Beaini, D. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35: 14501–14515.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, 593–607. Springer.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proc. of the ICAPS Conf.*, volume 22, 368–372.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proc. of the AAAI Conf. on AI*.

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 *Proc. of the Thirty-Third AAAI Conf. on AI*.

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 *Int. Workshop on Multi-disciplinary Trends in AI*, 57–68. Springer.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Wang, M. Y. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous graph attention network. In *The world wide web conference*, 2022–2032.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How Powerful are Graph Neural Networks? In *ICLR*.