Explainable Planner Selection

Patrick Ferber[1,2] and **Jendrik Seipp**[1]
[1]University of Basel, Switzerland
[2]Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
firstname.lastname@unibas.ch

Abstract

Since no classical planner consistently outperforms all others, it is important to select a planner that works well for a given classical planning task. The two strongest approaches for planner selection use image and graph convolutional neural networks. They have the drawback that the learned models are not interpretable. To obtain explainable models, we identify a small set of simple task features and show that elementary and interpretable machine learning techniques can use these features to solve as many tasks as the approaches based on neural networks.

Introduction

Automated planning is the task of finding a sequence of actions that transforms an initial state into a goal state (Ghallab, Nau, and Traverso 2004). Over the last decades researchers invented a large collection of planning algorithms, also called *planners*. All of them exhibit different strengths and weaknesses and therefore no single planner is preferable to all others for all planning tasks (e.g., Howe et al. 1999; Seipp et al. 2012). Consequently, it is often beneficial to combine multiple planners in a *portfolio*.

There are several types of portfolios, differing in whether they run a single or multiple planners and whether the planners, their time limits and their order are chosen offline, before seeing the task or online, when the task is known. For an overview of planning portfolios, see Vallati (2012) and Cenamor, de la Rosa, and Fernández (2016). The majority of portfolio approaches from the literature learn a schedule for multiple planners offline (e.g., Helmert, Röger, and Karpas 2011; Núñez, Borrajo, and Linares López 2015; Seipp et al. 2015). The schedule defines the order and time limits of a set of planners run in sequence. This sequential approach is based on the insight that a planner usually solves a task either quickly or not at all within the given resource limits.

The disadvantage of this approach is that it splits the available time among the planners in its portfolio. It can happen that for some tasks no planners in a portfolio solves the task quickly. In this case, it is better to choose a single planner with a high chance of solving the task and let it run for all of the available time. This is the motivation for the second main approach for planner portfolios and the one that we consider in this work: portfolio selection. Portfolio selectors have a collection of planners and predict for a given task how long each planner in the portfolio requires to solve the task or how confident the model is that a planner will solve the task. Then, a single planner is selected and executed. The main obstacle in this approach is finding suitable task features for the predicting model. Fawcett et al. (2014) collect a large set of handcrafted features and train different models for predicting the runtimes of several planners.

To avoid handcrafting features and potentially ignoring important features Sievers et al. (2019a) translate a given task into a graph (Sievers et al. 2019b) which preserves all information about the original task. They interpret the adjacency matrix of the graph as an image, scale the image down to 128x128 pixels, and train a convolutional neural network (CNN) to predict which planner will solve the given task. The idea is that the neural network automatically detects good features and indeed their model results in a strong planner. This is quite surprising since the 128x128 pixel image ignores a lot of information: many entries of the adjacency matrix are combined into the same pixel and the image does not distinguish between different types of nodes in the original graph. Nonetheless, the coverage scores of the resulting portfolio selector suggests that the remaining information in the image is sufficient for planner selection. In a follow-up paper, the lossy transformation from graphs to images is eliminated by using graph convolutional networks (GCN, Kipf and Welling 2017) and feeding the graphs directly into the neural network (Ma et al. 2020). This causes a modest performance improvement and implies that the images already contain enough information for good predictions.

The drawback of the neural network approaches is that the learned model is not interpretable, that is, we cannot ask the model *why* it selects a certain planner for a given task and *which features* are actually important for the selection (Cybenko 1989; Rudin 2019). Only if we obtain models that can answer these questions, we can deploy them with confidence and use them to understand the relative strengths of the component planners.

In this work, we analyze whether we actually need complex black-box models such as convolutional neural networks for strong planner selectors. We show that even with very basic task features and the most elementary machine learning techniques, we can create portfolios selectors for optimal planning that solve as many tasks as the approaches based on neural networks. In addition, our models have the advantage that they are explainable and fast to train.

Furthermore, we analyze which features we need and which features we can ignore for accurate predictions. We examine which planners our models choose and whether they know when to choose them. Additionally, we show how to visualize and understand the choices of a simple planner selection model.

Background

We train machine learning models that select a suitable planner for a given PDDL planning task (McDermott et al. 1998). Informally, a PDDL task defines a set of objects, a set of first-order predicates, and a set of action schemas. The objects are used when grounding the predicates and action schemas. The task uses the grounded predicates to describe an initial state and a goal condition. The grounded actions determine how a state can be transformed into a new state. A planner tries to find a sequence of actions that transforms the initial state into a state which satisfies the goal condition.

The machine learning techniques we use are linear regression, decision trees, random forests and multi-layer perceptrons. Each model receives as input a vector $\vec{v} \in \mathbb{R}^N$ containing the values of the input features.

Linear regression (Galton 1886) learns a weight vector $\vec{w} \in \mathbb{R}^N$ that assigns a weight to each feature. The output of a linear regression model is the weighted sum $\vec{v} \cdot \vec{w}$ of the features and weights. Linear regression chooses the weight vector that minimizes the squared error. This often uses unimportant features and causes overfitting. Thus, we also employ linear regression with L1 regularization (Tibshirani 1996). L1 regularization adds the L1 norm of the weight vector as penalty to the optimization process, which penalizes unnecessary feature weights and filters them out. The L1 penalization can be scaled with a parameter to make the filtering weaker or stronger.

A decision tree (Breiman et al. 1984) is a classifier which asks a sequence of questions and at the end predicts a class. To train a decision tree, we start with a single root node and assign all training examples to this node. Then, the training algorithm selects a feature and a threshold to split the training data such that some impurity metric (e.g., *Gini score*) is optimized. The feature and threshold are stored for the current node and the two parts of the split training data are associated with the two children of the current node. The algorithm is recursively applied to the children nodes until all samples associated to a node belong to the same class or some stopping criterion is reached.

Random forests (Breiman 2001) are an ensemble of decision trees. Here, we optimize multiple decision trees independently and obtain the overall prediction by averaging over the individual predictions.

The last type of machine learning model we train are multi-layer perceptrons (MLP, Goodfellow, Bengio, and Courville 2016). An MLP is a simple neural network consisting of multiple layers of neurons. Each layer is densely connected to the next layer. The value for each neuron is the weighted sum of the neurons connected to it (cf. linear regression). The value of the neuron is modified by a non-linear function (e.g., $ReLU(x) = \max(0, x)$) and is forwarded to the next neurons. The output of an MLP are the values of the neurons in the final layer.

Training

For each task in our benchmark set, we compute the values of our features and measure the runtimes of a set of planners for the task. Then we use supervised learning to train models for planner selection. To be comparable to previous work, we use the data set from Ferber et al. (2019), which contains both a list of benchmark tasks and their planner runtimes.

Benchmarks The benchmarks in the data set stem from the 1998–2018 classical planning tracks of the International Planning Competition (IPC). Additionally, the set includes the domains BRIEFCASEWORLD, FERRY, and HANOI from the IPP benchmark collection (Köhler 1999), the GEDP domain (Haslum 2011), domains from the T0 conformant-to-classical planning compilation (Palacios and Geffner 2009), and the FSC domain (Bonet, Palacios, and Geffner 2009). All runtime measurements are limited to 30 minutes and 7744 MiB of memory. We remove those tasks from the data set that none of the planners below solves within these limits. This leaves us with 2439 tasks, 145 of which were introduced for the IPC 2018.

Features For each task in the data set we compute four different sets of features. The first set (FAWCETT) contains the features described by Fawcett et al. (2014). This set contains features from the PDDL description of the task (e.g., the number of action schemas), features from its translation to a SAS$^+$ task (e.g., the number of mutex groups) and to a SAT formula, features from short runs with Fast Downward, and many more. These features are interpretable for domain experts, but some take very long to compute or require additional expertise to understand them. To analyze how complex features have to be for good planner selection, the second feature set (FPDDL) uses only the PDDL features of Fawcett et al. (2014). These features are very easy to interpret and they only require access to the PDDL files (i.e., no grounding, external planner or SAT solver is needed). The third feature set (PDDL) extends the PDDL features of Fawcett et al. (2014) with further PDDL features such as the minimum, mean, and maximum number of prevail conditions in all actions or the ratio of initial state facts over the number of objects. The fourth and last set (UNION) is the union of the other three feature sets.

Planners We use the same 17 planners as Sievers et al. (2019a) and Ma et al. (2020): SymBA* (Torralba et al. 2017) and 16 Fast Downward configurations (Helmert 2006). All Fast Downward configurations use A* search (Hart, Nilsson, and Raphael 1968) and strong stubborn sets (Wehrle and Helmert 2014). Each of the following eight heuristics is used twice, once with DKS structural symmetries pruning (Domshlak, Katz, and Shleyfman 2012; Shleyfman et al. 2015) and once with structural symmetries pruning using

		Linear Regression					MLP		Rnd. Forest
		0.0	0.1	1.0	2.0	5.0	3	5	50
FAWCETT	binary	78.6 (8.3)	77.2 (10.5)	82.1 (8.7)	82.4 (9.4)	80.9 (9.4)	**87.1 (6.1)**	78.2 (15.3)	**84.8 (7.5)**
	logtime	79.3 (9.2)	79.0 (10.0)	81.5 (7.7)	81.7 (6.5)	83.6 (5.2)	82.2 (8.4)	82.2 (8.4)	84.1 (7.1)
	time	78.6 (8.2)	81.8 (7.1)	80.5 (7.5)	80.4 (7.2)	80.3 (7.9)	82.2 (7.6)	**85.3 (6.7)**	81.8 (15.7)
FPDDL	binary	**87.7 (7.2)**	74.3 (15.1)	72.7 (15.7)	74.3 (16.6)	71.4 (15.4)	81.0 (8.0)	81.5 (7.3)	77.5 (16.0)
	logtime	82.5 (11.8)	84.0 (6.8)	78.5 (8.3)	77.7 (9.0)	80.3 (8.4)	78.2 (6.2)	79.7 (7.6)	82.0 (6.1)
	time	86.5 (7.8)	**86.5 (7.8)**	**86.5 (7.9)**	**86.6 (7.8)**	**86.6 (7.8)**	80.2 (6.6)	81.9 (6.0)	78.8 (15.5)
PDDL	binary	81.4 (9.3)	75.7 (12.1)	72.6 (16.3)	74.1 (16.6)	71.4 (15.4)	78.1 (9.9)	79.8 (6.8)	80.2 (13.3)
	logtime	82.1 (8.4)	79.7 (11.2)	80.4 (9.1)	79.8 (8.7)	77.8 (13 .0)	79.5 (8.2)	78.0 (7.5)	82.8 (7.0)
	time	81.6 (8.9)	82.0 (9.1)	81.2 (10.1)	79.0 (10.9)	78.7 (11 .6)	77.8 (11.0)	78.4 (10.0)	79.7 (16.7)
UNION	binary	74.8 (9.7)	81.0 (8.3)	79.4 (11.1)	82.4 (9.3)	80.9 (9.4)	84.7 (7.7)	78.3 (13.6)	82.1 (8.5)
	logtime	75.6 (10.2)	80.0 (9.2)	80.7 (7.9)	81.8 (6.7)	83.4 (5.8)	82.2 (8.4)	82.2 (8.4)	84.7 (7.6)
	time	74.8 (8.8)	77.3 (11.9)	75.7 (11.0)	76.1 (11.5)	77.1 (10.3)	84.3 (6.8)	83.6 (7.7)	84.0 (13.9)
	average	80.3	79.9	79.3	79.7	79.4	81.5	80.8	84.9

Table 1: Mean coverage and (in brackets) standard deviation (in percentage of tasks solved) over ten domain-preserving test folds for linear regression models with different L1 regularization weights, MLPs with 3 and 5 layers, and a random forest with 50 trees trained on (FAWCETT) the features of Fawcett et al. (2014), (FPDDL) the PDDL features of Fawcett et al. (2014), (PDDL) the extended set of PDDL features, and (UNION) the union of all features. The best setting in each column is highlighted.

orbital space search (OSS, Domshlak, Katz, and Shleyfman 2015): blind heuristic, LM-Cut (Helmert and Domshlak 2009), iPDB (Haslum et al. 2007), a zero-one cost partitioning pattern database (ZO-PDB) using a genetic algorithm to compute the patterns (Edelkamp 2006), and four Merge-and-shrink (M&S) heuristics (Dräger, Finkbeiner, and Podelski 2006; Helmert et al. 2014) using bisimulation (BS) (Nissim, Hoffmann, and Helmert 2011), full pruning (Sievers 2017), Θ-combinability (Sievers, Wehrle, and Helmert 2014), partial abstractions (Sievers 2018), and merging based on either DFP (Sievers, Wehrle, and Helmert 2014), strongly connected components (SCC) of the causal graph (Sievers, Wehrle, and Helmert 2016), MIASM (Fan, Müller, and Holte 2014), or score-based MIASM (sbMIASM, Sievers, Wehrle, and Helmert 2016). All planners except for two M&S configurations use h^2 mutexes to prune irrelevant actions (Alcázar and Torralba 2015).

Target Functions We want to train models that select a suitable planner for a given planning task. To this end, we compare three different target functions for the machine learning models: *time*, *logtime* and *binary*. The first variant (*time*) predicts for each planner the time expected for the planner to solve the task. Then we can select the planner with the shortest expected runtime. Because the runtime distribution is heavily skewed to short runtimes, we also train models on the logarithmically-scaled runtimes, called *logtime*. In the end, we are not interested in selecting the fastest planner for a task, but the planner with the highest chance to solve the task. Therefore, we also train our models on the *binary* information whether a planner solves a task within the resource limits.

Machine Learning Models We use three types of machine learning techniques. First, we train plain linear regression models (Galton 1886) and models with L1 regularization (Tibshirani 1996) using regularization weights of 0.1, 1.0, 2.0 and 5.0. Second, we train random forests (Breiman 2001), i.e., ensembles of decision trees (Breiman et al. 1984). Linear regression and random forests internally train an independent model for each planner. Finally, we train fully-connected multi-layer perceptrons (MLP) with 3 and 5 layers. Although they are one of the simplest kind of neural networks, they are not easily explainable. We include them mainly as an intermediate approach between Delfi, which uses a complex neural network with learned latent features, and linear regression, which can be seen as a single-layer network with handcrafted features. The last layer of our MLP models contains an output neuron for every planner. We use the Adam optimizer (Kingma and Ba 2015) with a learning rate of 0.001 to optimize the weights. For the networks that predict the *time* or *logtime* we use the *ReLU* activation function and the *mean squared error*. For the networks that predict the *binary* label we use the *Sigmoid* activation function and the *cross entropy loss*. In contrast to our linear regression and random forest models, our MLP technique learns a single model for planner selection.

Model Evaluation For training and evaluating models we split the tasks into groups of training and test tasks. Since neither linear regression nor random forests support validation data, we do not use validation data for the MLPs either. Because the range of some feature values varies greatly, we augment all feature sets by normalizing each feature to values between 0 and 1 and add these normalized features to the original feature sets. We use only the feature values of the training tasks to estimate the parameters for the normaliza-

Feature	Degradation
requires negative preconditions	4.4 (10.0)
max params per predicate	2.7 (7.0)
mean negations per effect	2.6 (10.6)
mean predicates per effect	2.4 (10.2)
requires conditional effects	2.1 (9.1)
requires equality	1.8 (8.9)
max predicates per effect	1.8 (8.3)
#types	1.6 (9.9)
min predicates per effect	1.6 (7.7)
#actions with neg. effects / #actions	1.5 (9.8)
requires STRIPS	1.5 (7.7)
requires typing	1.4 (8.1)
mean params per predicate	1.4 (8.0)
#goals	1.2 (7.6)
has types	1.0 (7.9)
min predicates per precondition	0.9 (8.2)
#predicates	0.9 (7.2)
requires ADL	0.8 (6.9)
max negations per effect	0.8 (6.1)
min negations per effect	0.7 (8.0)
#actions	0.7 (7.4)
#initial conditions	0.6 (7.0)
max predicates per precondition	0.5 (8.6)
mean predicates per precondition	0.4 (10.3)
requires action costs	0.2 (6.8)
#initial functions	0.1 (6.9)

Table 2: Mean coverage degradation and (in brackets) standard deviation (in %), over ten domain-preserving test folds, when ignoring a single group of highly correlated features of the FPDDL feature set for training a linear regression model without L1 regularization on the *binary* labels. Groups without performance degradation are omitted.

tion. We train the model to learn for each planner a function mapping from the features to a target function. To evaluate the final performance of the model on the test tasks, we use the model to predict the runtime for each planner on each test task, respectively their likeliness to solve the task. For each task we select the planner with the shortest runtime, respectively the highest chance. Afterwards, we count how many test tasks we would have solved with our decision. We note that our evaluation metric, which counts the solved tasks, differs from the training metric which optimizes a squared error, cross entropy, or the Gini score.

Experiments

Our experiments are structured as follows. First, we evaluate how helpful simple machine learning techniques with explainable features are for selecting a planner that solves a given task. Then, we analyze which features are important for the models. Next, we inspect which planners are favored by our models. Afterwards, we train and visualize a decision tree for planner selection. Finally, we compare our models to Delfi1 (Katz et al. 2018), the winner of the IPC 2018, which uses convolutional neural networks for planner selection.

Usage	Cov_P	Cov_C	Planner
43.7	80.1	94.4	🟪 SymBA*
12.3	82.4	89.9	🟫 h2 + OSS + LM-Cut
9.7	78.7	54.5	🟧 h2 + DKS + iPDB
9.4	78.8	88.5	🟦 h2 + OSS + iPDB
8.1	82.7	78.1	🟥 h2 + DKS + LM-Cut
5.4	67.9	74.8	🟩 DKS + M&S-MIASM-DFP
3.3	74.8	97.5	🟦 h2 + DKS + M&S-BS-sbMIASM
2.8	65.9	86.6	🟩 h2 + OSS + M&S-SCC-DFP
2.1	75.8	100	🟪 h2 + DKS + M&S-BS-SCC-DFP
1.0	67.7	84.0	⬜ OSS + M&S-MIASM-DFP
0.8	72.2	75.0	🟦 h2 + OSS + M&S-BS-sbMIASM
0.7	68.4	6.2	🟥 h2 + DKS + ZO-PDB
0.4	67.6	60.0	🟩 h2 + DKS + M&S-SCC-DFP
0.2	68.6	100	🟪 h2 + OSS + ZO-PDB
0.1	62.3	100	⬛ h2 + DKS + Blind
0.0	62.5	–	🟨 h2 + OSS + Blind
0.0	75.2	–	🟫 h2 + OSS + M&S-BS-SCC-DFP

Table 3: Planners selected by the linear regression model without L1 regularization trained on the FPDDL features and optimizing the *binary* labels. The columns show how often each planner is chosen (in %), the coverage (in %) for the planner on all tasks (Cov_P), and the coverage (in %) on tasks for which the model chooses the planner (Cov_C).

All experiments — except for the comparison to Delfi1 — use 10-fold cross-validation, that is, we split the data into ten similarly-sized folds. We use one fold for testing and train the model on the other nine folds and repeat this procedure ten times. Every time a different fold is used for testing. The final performance is the mean performance over all ten runs. Cross-validation allows us to evaluate our approach on all benchmark tasks instead of just a subset (e.g., the tasks from the last IPC).

Planning tasks from the same benchmark domain share the same structure. Therefore, if the training and test data contain tasks from the same domain, the test performance does not show how well the model generalizes to new unseen tasks, but how well the model generalizes to tasks from known domains. Thus, we use *domain-preserving* splits, i.e., we ensure that all tasks of the same planning domain are assigned to the same data fold.

We cannot use cross-validation for the comparison to Delfi1, because it is trained on all tasks available prior to the IPC 2018 and its code is not available for retraining. For the comparison to Delfi1, we train our models 10 times on the same tasks that Delfi1 was trained on.

We run all experiments on single Intel Xeon Silver 4114 cores and limit memory usage to 3 GiB. All our data sets, code, and experiment results are published online (Ferber and Seipp 2020).

Comparison of Machine Learning Models

We begin by evaluating how useful elementary machine learning techniques with basic features are at choosing a planner to solve a given task. For each of the four feature sets and each label representation (*binary*, *logtime*, and *time*), we train five linear regression configurations with L1 regularization weights from 0.0 to 5.0, a single random forest with 50 trees, and two neural network configurations with 3 resp. 5 hidden layers.

Table 1 shows the percentage of solved tasks for all models. A portfolio selector that chooses planners randomly obtains a coverage of 67.2%. We see that all models surpass this baseline. Although the models are not optimized for the coverage metric, the ability to predict the runtimes of a planner (resp. the chance to solve a task) helps to select a good planner for a task.

Averaging the coverage fractions of our machine learning configurations over all feature sets and target functions reveals that the random forest is the most robust technique (84.9%). The next best configuration are MLPs with 3 layers with an average coverage of 81.5%. The most robust linear regression configuration uses no L1 regularization and solves 80.3% of the test tasks.

Averaging over the feature sets and the machine learning configurations reveals how useful the different target functions are. Because we train 5 linear regression configurations, 2 MLP configurations, but only one random forest configuration, we weight the average. The *binary* labels are the least informative (80.0%). The *logtime* (81.5%) and the *time* (81.2%) labels are approximately equally informative. Although, we only need to learn if a planner solves a given task, training the model to approximate planner runtime helps us to improve our planner selection.

The results also show that some of the strongest models use the FPDDL feature set, although all of its features are also contained in all other feature sets. The larger feature sets sometimes lead to lower coverage because they make it easier for the models to overfit on the training data. The linear regression models in particular, but also the MLPs suffer from overfitting: with growing feature sets, the training error decreases, but the test error increases.

Feature Importance

Having trained well-performing models, we can now analyze how important each feature is for planner selection. This shows which properties of a task are important for the runtime of a planner and allows us to skip unnecessary features to speed up the predictions.

To measure the importance of a feature, we retrain the model without using that feature. Exploratory experiments uncovered that some features are highly correlated (e.g., the number of PDDL objects and the number of equality conditions). Removing a feature that is highly correlated with another one has no impact on the performance, because the model will instead use the correlated feature. For example, the FPDDL feature set has 49 features in 47 groups of correlated features, while the FAWCETT feature set has 410 features in 189 groups. Therefore, we retrain the base model, but exclude groups of correlated features.

Table 2 shows the performance degradation of the best linear regression model for FPDDL features. The most important information for the model is whether the task requires negative preconditions: removing it degrades the performance by 4.4%. This suggests that some planners work

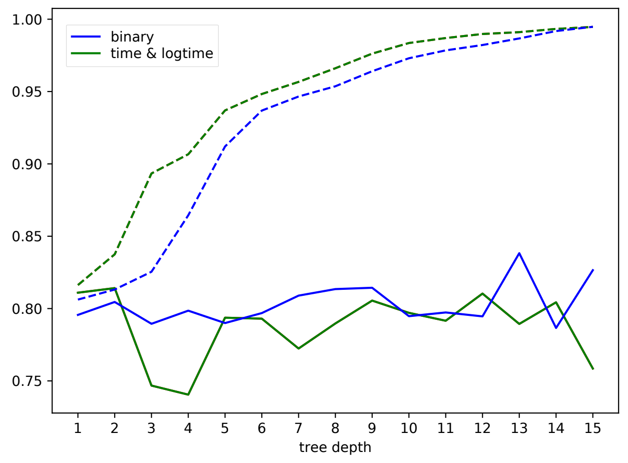

Figure 1: Mean coverage over all 10 domain-preserving folds, on training (dashed) and test (solid) data using decision trees trained on binary (blue) and time/logtime labels (green) for increasing tree depth on the PDDL features.

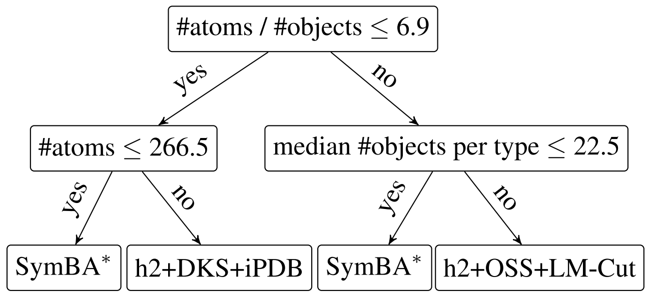

Figure 2: Example decision tree for one cross-validation fold. The decision tree uses *time* labels and has two layers.

better for tasks with negative preconditions than other planners. 21 out of 47 feature groups can be removed without a negative impact on the performance. Those 21 groups contain 22 features.

Planner Selection

To understand how the models obtain high coverage scores, we examine which planners they choose and whether the models correctly learned when to choose them. Table 3 shows for the best linear regression model on the FPDDL feature set which planners are selected, how often those planners are selected, the coverage of those planners on all test tasks, and the coverage of those planners only on the tasks they have been chosen for.

We observe that the model almost always selects a planner from a group of planners with high coverage on the test task. Most often — for 43% of the tasks — it chooses SymBA*, which is not the strongest planner, but almost solves as many tasks as the strongest one. Given this data, we could suspect that the model just detected a group of good planners and randomly chooses one of them. We analyze in the following why this is not the case. If the model selected a planner for each task (weighted) randomly, then each individual planner would be used for a uniform subsample of the test tasks. Therefore, their coverage on their subsample would be approximately equivalent to their coverage on all test tasks. The linear regression model obtains for most planners a significantly better coverage on the tasks it assigns to them than the planners obtain on all test tasks. This shows that the model indeed learned when to use which planner.

Single-Model Planner Selection

In the previous experiments, we used basic machine learning algorithms to learn models whose predictions can be easily interpreted. For example, the predictions of linear regression can be explained by multiplying the learned weights with the features of a task, then sorting those products (not the learned weights) by absolute magnitude and finally by comparing the positive and negative impact of the different features. However, our models make a prediction per planner, instead of choosing a planner directly. Therefore, their explanations answer the question "Why does the model think that planner A solves the given task?", but not "Why is planner A preferable to planner B?". Models that answer the second question make it even easier to understand which planners work well for which tasks.

To obtain such a model we train a single decision tree using the planner names as training labels. A decision tree does not support multiple labels, i.e., planner names, for a single sample. Thus, we duplicate each training sample for each planner that solves it and assign one of the planner labels to each duplicate. Because this overrepresents frequently solved tasks, each duplicate is weighted by one over the number of times it was duplicated. This setup corresponds to the *binary* labels of previous experiments. To incorporate the *logtime* or *time* information into the training, we add an additional runtime factor to the weight of the duplicates. For two duplicates x and y where the planner for x is n times faster than the planner for y, the runtime factor for x is n times larger than the factor for y and for all duplicates that belong to the same task, the runtime factors sums up to one. This lets the decision tree prefer faster planners.

The result of the training procedure is a decision tree that directly predicts which planner to use for a given task. Obviously, the deeper the decision tree grows, the more questions it asks and each additional layer doubles the number of leaf nodes. Since using too many layers can lead to overfitting, we train decision trees with different depths and compare their performance with the same cross-validation procedure as above. Figure 1 shows the training (dashed line) and test (solid line) coverage of decision trees with increasing depth. The coverage on the training data quickly approaches 100%, while the test coverage does not vary much for different tree depths. The trees obtained for the *time* and *logtime* labels are identical, so they share the same color in the plot.

Figure 2 illustrates how easy it is to interpret the learned decision trees. We show an exemplary decision tree with a tree depth of 2 and *time*-weighted labels, because the tree has a good test performance and is small enough to be visualized. Each internal tree node contains the question asked.

	FAWCETT			PDDL			Delfi Features		
Seconds	0.1	0.2	10.8	0.2	0.3	11.0	0.4	0.8	50.2
MiB	16	17	200	24	25	138	26	69	3023

Table 4: Minimum/mean/maximum time and memory usage to extract the features from the IPC 2018 tasks.

Depending on the answer for the given task, we traverse to the first or second child. Once we reach a leaf, the prediction is the most frequent class in the training data associated with the leaf. The example decision tree reveals that SymBA* is preferable to the other two planners in the tree only when the number of atoms and objects is small. The authors of SymBA* confirmed that this finding is aligned with their experience (personal communication).

Comparison to Delfi1

In our final experiment, we compare our models against the current strongest portfolio selector, Delfi1. Since for Delfi1 only the model and not the code is available, we cannot retrain it with cross-validation. Instead we retrain the best configurations of our machine learning techniques on the training data of Delfi1 and evaluate all models on the tasks from the IPC 2018. For the single decision tree approach, we use the decision tree from Figure 2.

Delfi1 converts a given task to a graph and then to a raster image and finally passes this image to a CNN. In Table 4 we compare how computationally demanding these conversions are in comparison to computing the FAWCETT and PDDL features. On average, Delfi1 uses the most time and memory, but the mean resource requirements are small enough to be neglected for all three variants. However, on resource-constrained systems it might be problematic to run Delfi1 for the largest tasks.

Next, we compare the performance of the different models. Table 5 shows that all models significantly outperform the random baseline. Delfi1 performs best and solves 86.9% of the test tasks. However, our linear regression models performs almost equally well (86.2% coverage). The single decision tree learned to select almost always the planner performing best on the test data (not the planner performing best on the training data) and obtains a coverage of 82.7%. The random forest model also achieves a high coverage (80.4%), but the MLP solves many fewer tasks (70.8% coverage). It is striking that the linear regression, decision tree, and random forest models have a standard deviation of 0.0 over ten repetitions. This is in contrast to Delfi1, whose authors note that the coverage of the Delfi approach has a high variance and that their retrained models did not reach the performance of the Delfi1 model that participated in the IPC 2018 (Sievers et al. 2019a).

For our models, the MLP takes the longest time for training: 111 seconds on average using a single CPU core. Similarly, training the Delfi1 models was a matter of minutes (personal communication with the authors). Since training can be done offline before encountering new tasks and the resource consumption is within reasonable limits, the time for training is negligible. Evaluating the models requires

	Random	Linear Regression	MLP	Random Forest	Decision Tree	Delfi1/CNN
coverage in %	60.6 (0.3)	86.2 (0.0)	70.8 (9.0)	80.4 (0.0)	82.7 (0.0)	86.9 (N/A)
$\text{time}_{\text{training}}$ in sec.	0	0.0 (0.0)	111.1 (16.8)	38.8 (6.1)	0.5 (0.1)	<3600 (N/A)
$\text{time}_{\text{selecting}}$ in sec.	0	3.8 (0.2)	3.9 (0 .1)	4.1 (0.3)	4.6 (0.4)	3.7 (0.4)
$\text{memory}_{\text{selecting}}$ in MB	0	275.9 (434.0)	286 .9 (131)	323.2 (110.3)	245.1 (23.5)	313.7 (0.2)

Table 5: For the best variant of each basic machine learning technique and Delfi1, we show mean coverage, training time, and the runtime and memory usage for a single prediction on the IPC 2018 tasks. The numbers in brackets show standard deviation.

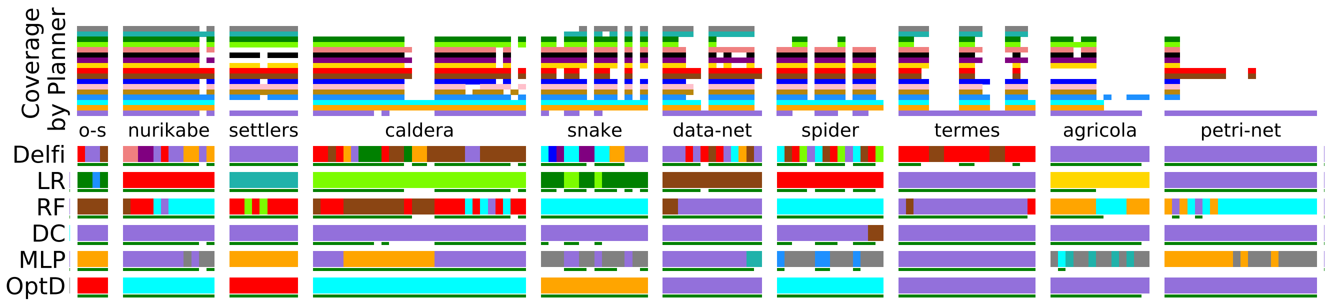

Figure 3: (Top) For each planner in our data set a row indicates which tasks it solves from the test set. The tasks are grouped by domain and the domains are sorted by their average coverage. (Bottom) For each model of Table 5 a row indicates for every test task the planner chosen. Beneath each row a small green bar indicates whether the selected planner solved the task. The final row (OptD) selects for each domain the planner with the highest coverage in that domain (breaking ties by selecting the planner with smaller cumulative runtime in the domain). Table 3 shows which planner belongs to which color.

roughly the same time and memory for all approaches. Again the resource requirements are negligible.[1]

The bottom half of Figure 3 shows for each model when it chooses which planner and whether these choices solve the given task. Delfi1 solves the most tasks, but in many domains it uses multiple planners. This suggests that the decisions of Delfi1 are not based on detecting common structures within a domain, but on some other structures. Due to the black-box nature of the model of Delfi1, it is not easily possible to infer what it learns.

In contrast, the linear regression model solves almost the same number of tasks as Delfi1, but clearly learns to assign planners to domains. Only twice it selects two different planners for a domain. Intuitively, it makes sense to learn to identify domains and which planner is good for a domain. To verify our intuition, the last row *OptD* shows an oracle that selects for each domain the planner with the highest coverage in that domain. Indeed, we see that selecting a single planner per domain solves all but one test task.

The random forest model also obtains high coverage on the IPC 2018 tasks and in most domains it solves more tasks than the linear regression. Its worse overall performance is only due to the difficult *petri-net* domain where it solves only two tasks. The decision tree model learned to almost always select SymBA* which is the planner with the highest coverage on the test tasks. This is more impressive than it sounds, because SymBA* is not the best planner for the training tasks.

[1]All variants use Python for training and evaluation, which needs ∼3.5s just to load the Tensorflow and scikit-learn packages.

The MLP surpasses the random baseline, but has the lowest coverage of all models. Again, we clearly see the domains where it makes wrong predictions. In three out of ten domains it solves only one task and in two domains only half of the tasks. In all other domains it solves almost all tasks.

The top half of Figure 3 shows that there are many tasks which are solved by almost all planners and many tasks that are solved by almost no planner. Obviously, selecting a good planner is much more important for hard tasks. If we check where the models, including Delfi, fail to select a solving planner, we see that this often happens for tasks that are solved by almost no planner. An extreme case can be seen for the linear regression model in agricola. The model selects a single planner for the domain and solves half of the tasks. This sounds fine until we notice that those tasks are solved by most planners. Therefore, focusing on hard tasks could improve future models.

Conclusions

We showed that simple and explainable machine learning techniques like linear regression can produce strong portfolio selectors. Arguably our simplest model, linear regression, solves roughly the same number of tasks as Delfi1, the state of the art for planner selection. In addition to obtaining high coverage scores, the linear regression model is easy to interpret and fast to train and evaluate. We also analyzed which features are important for planner selection and presented how a single decision tree can be trained to directly predict a planner and showed that such a tree can visualize how the model makes decisions.

Acknowledgments

Patrick Ferber was funded by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science). This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Certified Correctness and Guaranteed Performance for Domain-Independent Planning" (CCGP-Plan).

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-state Controllers Using Classical Planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

Breiman, L. 2001. Random Forests. *Machine Learning* 45(1): 5–32.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth.

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.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2(4): 303–314.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry Breaking in Deterministic Planning as Forward Search: Orbit Space Search Algorithm. Technical Report IS/IE-2015-03, Technion.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed Model Checking with Distance-Preserving Abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Edelkamp, S. 2006. Automated Creation of Pattern Database Search Heuristics. In Edelkamp, S.; and Lomuscio, A., eds., *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-Linear Merging Strategies for Merge-and-Shrink Based on Variable Interactions. In Edelkamp, S.; and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

Fawcett, C.; Vallati, M.; Hutter, F.; Hoffmann, J.; Hoos, H.; and Leyton-Brown, K. 2014. Improved Features for Runtime Prediction of Domain-Independent Planners. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 355–359. AAAI Press.

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

Ferber, P.; and Seipp, J. 2020. Supplementary Material for the ICAPS 2020 XAIP Workshop Paper "Explainable Planner Selection". https://doi.org/10.5281/zenodo.4282210.

Galton, F. 1886. Regression Towards Mediocrity in Hereditary Stature. *Journal of the Anthropological Institute of Great Britain and Ireland* 15: 246–263.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2): 100–107.

Haslum, P. 2011. Computing Genome Edit Distances using Domain-Independent Planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM* 61(3): 16:1–63.

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*, 28–35.

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and von Mayrhauser, A. 1999. Exploiting Competitive Planner Performance. In Biundo, S.; and Fox, M., eds., *Recent Advances in AI Planning. 5th European Conference on Plan-*

ning (ECP 1999), volume 1809 of *Lecture Notes in Artificial Intelligence*, 62–72. Heidelberg: Springer-Verlag.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online Planner Selection for Cost-Optimal Planning. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 57–64.

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

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations (ICLR 2017)*.

Köhler, J. 1999. Handling of Conditional Effects and Negative Goals in IPP. Technical Report 128, University of Freiburg, Department of Computer Science.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 5077–5084. AAAI Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990. AAAI Press.

Núñez, S.; Borrajo, D.; and Linares López, C. 2015. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artificial Intelligence* 226: 75–101.

Palacios, H.; and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research* 35: 623–675.

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1(5): 206–215.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning Portfolios of Automatically Tuned Planners. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 368–372. AAAI Press.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic Configuration of Sequential Planning Portfolios. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3364–3370. AAAI Press.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S. 2017. *Merge-and-shrink Abstractions for Classical Planning: Theory, Strategies, and Implementation*. Ph.D. thesis, University of Basel.

Sievers, S. 2018. Merge-and-Shrink Heuristics for Classical Planning: Efficient Implementation and Partial Abstractions. In Bulitko, V.; and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 90–98. AAAI Press.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019a. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7715–7723. AAAI Press.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2019b. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 446–454. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized Label Reduction for Merge-and-Shrink Heuristics. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An Analysis of Merge Strategies for Merge-and-Shrink Heuristics. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 294–298. AAAI Press.

Tibshirani, R. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58(1): 267–288.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence* 242: 52–79.

Vallati, M. 2012. A Guide to Portfolio-Based Planning. In Sombattheera, C.; Loi, N. K.; Wankar, R.; and Quan, T. T., eds., *Proceedings of the 6th International Workshop on Multi-disciplinary Trends in Artificial Intelligence (MIWAI 2012)*, volume 7694, 57–68. Springer.

Wehrle, M.; and Helmert, M. 2014. Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.