Graph Neural Networks for Algorithm Selection in Abstract Argumentation

Jonas KLEIN, Isabelle KUHLMANN and Matthias THIMM
Artificial Intelligence Group, University of Hagen, Germany

Abstract. We address the task of selecting the fastest algorithm, in terms of runtime, for determining skeptical acceptance under preferred semantics in abstract argumentation frameworks out of a set of multiple algorithms by means of machine learning. To be precise, we examine four "classical" machine learning techniques, as well as three graph neural networks, and compare all of these approaches with regard to both prediction accuracy and the total amount of time the selected algorithms require to solve a given test set in an experimental analysis. Our set of algorithms includes three solvers from the International Competition on Computational Models of Argumentation. Our results demonstrate that graph neural networks are a promising method for algorithm selection in abstract argumentation, as two out of three neural network models outperform all four classical machine learning approaches.

Keywords. Abstract Argumentation, Machine Learning, Graph Neural Networks, Algorithm Selection

1. Introduction

Approaches to formal argumentation [3] include non-monotonic reasoning techniques that focus on the interaction between arguments and counterarguments. One of the most influential theories in this area is the pioneering work on abstract argumentation by Dung [17], which introduced *abstract argumentation frameworks* to model the interplay between arguments, and various semantics to decide the acceptability of these arguments. Argumentation scenarios are represented as directed graphs where vertices represent arguments, and "attacks" between arguments are modeled as directed edges. In order to reason with these graphs, one is usually interested in identifying sets of arguments (*extensions*) that are mutually acceptable, given a specific semantics. Typical problems in abstract argumentation include deciding whether an argument is included in one (or all) extensions under a given semantics, and enumerating one (or all) extensions under a given semantics. Several of these reasoning problems are NP-hard [12].

In recent years, there has been an increased effort to develop algorithms and systems to solve these high-complexity problems [40,21]. Various works have shown that *combining* different algorithms, e.g., in portfolios, can be beneficial [11,42]. Vallati et al. [43] investigate predictive models using well-known machine learning approaches to perform algorithm selection. More precisely, they evaluate numerous sets of (mostly graph-based) features in terms of their expressiveness for this classification problem. Given the increasing research interest in deep learning approaches in abstract argumen-

tation [32,13,33,14], the question arises as to how these methods can be utilized for algorithm selection. In this work, we investigate the applicability of Graph Neural Networks (GNNs) to perform instance-based algorithm selection. To be precise, we follow up on the work by Vallati et al. by examining four "classical" machine learning techniques (k-nearest neighbors, naive Bayes, random forest, and support vector machine), as well as three GNN approaches (Graph Convolutional Network [31], Graph Isomorphism Network [45], and GraphSage [25]) for the task of predicting the fastest among a selection of three sound and complete solvers (*ArgSemSAT* [10], *Fudge* [39], and μ*-toksia* [34]). In this work we focus on the selection of classifiers, while Vallati et al. focus on the selection of features to use with such classifiers.

The remainder of this paper is structured as follows. In Section 2, we discuss the relevant preliminaries with regard to abstract argumentation, as well as both classical machine learning and graph neural network techniques. Section 3 comprises an overview of our approach and methodology. In Section 4, we present an experimental analysis, and we conclude in Section 5.

2. Preliminaries

In the following, we provide an overview of the fundamentals of abstract argumentation on the one hand, and of classical machine learning as well as graph neural network methods on the other hand.

2.1. Abstract Argumentation

An *abstract argumentation framework* is a tuple $\mathsf{AF} = (\mathsf{A}, \mathsf{R})$ where A is a set of arguments and R is a relation $\mathsf{R} \subseteq \mathsf{A} \times \mathsf{A}$. For two arguments $a, b \in \mathsf{A}$ the relation $a\mathsf{R}b$ means that argument a attacks argument b. For $a \in \mathsf{A}$ define $a^- = \{b \mid b\mathsf{R}a\}$ and $a^+ = \{b \mid a\mathsf{R}b\}$. We say that a set $S \subseteq \mathsf{A}$ *defends* an argument $b \in \mathsf{A}$ if for all a with $a\mathsf{R}b$ then there is $c \in S$ with $c\mathsf{R}a$.

Semantics are given to abstract argumentation frameworks by means of extensions [17]. An extension E is a set of arguments $E \subseteq \mathsf{A}$ that is intended to represent a coherent point of view on the argumentation modelled by AF. Arguably, the most important property of a semantics is its admissibility. An extension E is called *admissible* if and only if

1. E is *conflict-free*, i. e., there are no arguments $a, b \in E$ with $a\mathsf{R}b$ and
2. E *defends* every $a \in E$,

and it is called *complete* (CO) if, additionally, it satisfies

3. if E defends a then $a \in E$.

Different types of classical semantics can be phrased by imposing further constraints. In particular, a complete extension E

- is *grounded* (GR) if and only if E is minimal,
- is *preferred* (PR) if and only if E is maximal, and
- is *stable* (ST) if and only if $\mathsf{A} = E \cup \{b \mid \exists a \in E : a\mathsf{R}b\}$.

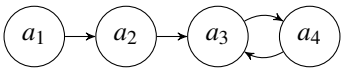

Figure 1. Abstract argumentation framework AF_1 from Example 1.

All statements on minimality/maximality are meant to be with respect to set inclusion. Note that the grounded extension is uniquely determined and that stable extensions may not exist [17].

Example 1. Consider the abstract argumentation framework AF_1 depicted as a directed graph in Figure 1. In AF_1 there are three complete extensions E_1, E_2, E_3 defined via

$$E_1 = \{a_1\}$$
$$E_2 = \{a_1, a_3\}$$
$$E_3 = \{a_1, a_4\}$$

E_1 is also grounded and E_2 and E_3 are both stable and preferred.

In this work we only consider the task of skeptical acceptance under preferred semantics, which we denote as $\mathsf{DS}_{\mathsf{PR}}$.

2.2. Classical Machine Learning Methods

The task considered in this paper is the selection of the fastest solver for a given problem instance. Thus, we are dealing with a classification problem: our goal is to *classify* which one of the algorithms at hand is most suitable to solve $\mathsf{DS}_{\mathsf{PR}}$ wrt. a given AF and a corresponding query argument $a \in \mathsf{AF}$.

There exists a plethora of machine learning (ML) approaches which solve different types of classification problems. The overall goal of an ML method is to "learn" from given data (*training data*) in order to apply this "knowledge" on unknown data (*test data*). For this work, we selected a total of four supervised machine learning techniques, namely *k-nearest neighbor*, *naive Bayes*, *random forest*, and *support vector machine*, which will be explained in more detail in the following. The term *supervised* refers to the fact that all labels of the training data are known at all times during training, meaning that in our application scenario, for each instance in the training dataset (i.e., for each AF and corresponding query argument), the fastest solver is known.

k-Nearest Neighbor The *k*-Nearest Neighbor (KNN) approach [20,15] is based on the idea of classifying a datapoint according to its nearest k neighbors. More specifically, given a datapoint Y we aim to classify, we calculate the distance of all datapoints in the training data to Y. Then we select the k datapoints with the shortest distance to Y (i.e., the *k nearest neighbors*). Finally, we assign Y the class which is most frequently found among the nearest neighbors, using a voting rule.

Naive Bayes According to Bayes's well-known Theorem (based on [5]), the probability of a datapoint $Y = (y_1, \ldots, y_n)$ belonging to class $c_d \in \{c_1, \ldots, c_m\}$ is

$$p(c_d \mid Y) = \frac{p(Y \mid c_d)p(c_d)}{p(Y)},$$

where p is a probability function. The naive Bayes (NB) classifier is built on the "naive" assumption that the value of a certain feature is independent of any other feature (i.e., given c_d, y_1 is independent of y_2, and so forth), given the value of the class variable. Because of this independence assumption, we can use $p(Y \mid c_d) = p(y_1 \mid c_d) \cdot \ldots \cdot p(y_n \mid c_d)$, and get

$$p(c_d \mid Y) = \frac{p(y_1 \mid c_d) \cdot \ldots \cdot p(y_n \mid c_d) \cdot p(c_d)}{p(y_1) \cdot \ldots \cdot p(y_n)}.$$

Since the denominator is constant (it is the same for each class) we can ignore it. Finally, we can classify Y by determining the class with the highest probability ($c_{\max}$):

$$c_{\max} = \underset{j \in \{1,\ldots,m\}}{\operatorname{argmax}} \, p(c_j) \prod_{i=1}^{n} p(y_i \mid c_j)$$

Random Forest A Random Forest (RF) is an ensemble of decision trees which vote for the most popular class [7]. A *decision tree* is, as the name suggests, a tree structure in which the inner nodes are essentially test nodes, and the leaf nodes correspond to class labels. A test node checks certain feature values of a given sample and computes some outcome which is associated with one of the node's subtrees. To classify a datapoint, we start at the root of the decision tree and propagate from test node to test node, until we reach a leaf node—which contains a class label, i.e., the classification result [36]. To construct an RF, for each individual decision tree, we randomly select a number of samples from the training set, and we randomly select a number of features to be considered [7].

Support Vector Machine The underlying principle of a Support Vector Machine (SVM) is that we view training samples as vectors in a vector space, which can be separated by hyperplanes, according to their class assignment. If the classes of the data at hand are not linearly separable (which is usually the case), we can apply a kernel function, which essentially transfers the training data to a higher dimension. If the dimension is high enough, the data become linearly separable [8].

2.3. Graph Neural Networks

We select a total of three different Graph Neural Network (GNN) architectures, namely *Graph Isomorphism Network*, *Graph Convolutional Network*, and *GraphSage*, which will be explained in more detail in the following. The core idea of message passing GNNs is to learn node or graph representations by iteratively aggregating local neighborhood information of a node (*messages* or *embeddings*) using non-linear transformations. Varying definitions of how the embeddings of the neighborhood nodes are aggregated (*aggregate function*) and how they are combined with the node embeddings from previous iterations (*combine function*) lead to different GNN architectures. After the final itera-

tion, the embeddings encapsulate structural information of a node, respectively graph. These generated embeddings can then be used for downstream prediction tasks. For node classification tasks, the embedding of the final iteration is used for prediction. For graph classification tasks, a so-called *readout function* is used to aggregate node embeddings to obtain a representation of the entire graph.

Graph Isomorphism Network The Graph Isomorphism Network (GIN) [45] models the Weisfeiler-Lehman graph isomorphism test [37] in a neural network. It implements the aggregate and combine functions as the sum of the node embeddings and a multi-layer perceptron (MLP) [27] with non-linearity. For graph-level readout, the node embeddings of every layer are summed up and concatenated to get the final graph representation.

Graph Convolutional Network Graph Convolutional Networks (GCNs) [31], are initially motivated by sprectral graph convolutions [26,16]. Following the definition in [45], they integrate the aggregation and combine step as an element-wise mean pooling, followed by a ReLU [1] non-linearity.

GraphSage The GraphSage [25] model was proposed with three different aggregation functions: (1) a mean aggregator, (2) a Long Short-Term Memory (LSTM) [28] aggregator and (3) a max-pooling aggregator. In this work, we consider the mean aggregator variant of GraphSage. The combination function is a concatenation followed by a linear mapping.

3. Machine Learning-based Approaches for Algorithm Selection

Various works in automated reasoning have investigated the concept of generating models that allow for identifying the most appropriate—or best—algorithm for an instance of a particular (computationally complex) problem. Empirical predictive models (EPMs) have been employed in many areas of Artificial Intelligence, such as the Satisfiability Problem (SAT) or Answer Set Programming (ASP), with great success [46,23]. A basic distinction is made between two approaches: classification approaches and regression approaches. Classification approaches assign any given instance a single category corresponding to the algorithm, which is predicted to be the fastest. Regression approaches, on the other hand, try to predict the actual runtime of each algorithm under consideration. The algorithm with the lowest predicted runtime is then selected.

In this work, we consider algorithm selection for the skeptical acceptance wrt. preferred semantics as a classification problem. Let $S = \{s_1, ..., s_n\}$ be a set of solvers and let $\mathfrak{A}$ be the set of all argumentation frameworks. Conceptually, a classifier C is a mapping $C : \mathfrak{A} \to S$, where any $\mathsf{AF} \in \mathfrak{A}$ is assigned a solver $s \in S$ that solves this instance the fastest. These mappings can be learned using ML methods. In order to do so, classical supervised ML approaches need some numerical representation of the instance in question, mainly referred to as features. In [43], Vallati et al. showed that classical ML methods can be exploited for algorithm selection in the context of abstract argumentation and identified informative features for classifying instances. However, only little work as been done in the area of computational models of argumentation for investigating the exploitability of modern deep learning techniques for algorithm selection. In this paper, we focus on GNNs, because, on the one hand, they have already been used successfully in argumentation [13,33], and on the other hand, no pre-calculation of features—which

is often time-consuming—is necessary for classification. These properties make GNNs a promising approach in the given context.

4. Experimental Analysis

In this section, we present the results of an experimental analysis, in which we (1) investigate the applicability of different GNN architectures to select the most appropriate solver given an AF and (2) compare them to "classical" machine learning approaches. The analysis aims to give an overview of *whether* and *to what extent* GNNs are suitable for algorithm selection in abstract argumentation, and how they differ from classical methods in terms of performance. Below, we describe the experimental setup and subsequently discuss our findings.

4.1. Experimental Setup

In this work, we consider three SAT-based approaches for solving the problem of skeptical acceptance under preferred semantics: *ArgSemSAT*, *Fudge*, and *μ-toksia*.

ArgSemSAT The *ArgSemSAT* solver [10] is the winner of the preferred semantics track at the 2017 International Competition on Computational Models of Argumentation (ICCMA'17). It iteratively calls a SAT solver to compute complete labelings and encoding constraints to drive the search towards the solution of decision and enumeration problems. It is written in C++ and can be used with the Minisat [18] or the Glucose [4] SAT solver. For our experiments we use *ArgSemSAT* with Glucose.

Fudge The *Fudge* solver [39] tightly integrates satisfiability solving technology to solve a series of abstract argumentation problems. While most of the encodings used by *Fudge* derive from standard translation approaches, *Fudge* makes use of completely novel encodings to solve the skeptical reasoning problem wrt. preferred semantics. It is written in C++ and uses the satisfiability solver CaDiCaL 1.3.13[1].

μ-toksia The *μ-toksia* solver [34] ranked first in all reasoning tasks of the ICCMA'19. It is a "purely" SAT-based system that is heavily based on the incremental use of SAT solving. This means, for iterative calls, the state of the SAT solver is maintained. By that, only a single SAT solver is instantiated during a single program run. It is implemented in C++ and includes interfaces to the Glucose [4] and CryptoMiniSAT [38] SAT solvers. We used *μ-toksia* with CryptoMiniSAT as an underlying SAT solver for our experiments.

It should be noted that we considered various other solvers such as *Pyglaf* [2], *Heureka* [24], or *ConArg2* [6] in our initial experiments. However, finding a fruitful mix of solvers to select from is a challenge in itself. We analyzed different combinations of solvers with regard to two criteria: (1) the number of instances for which each solver achieved the best performance and (2) the differences in runtimes (compared to the other solvers) wrt. the best-solved instances of each solver. The second criterion, in particular, is essential, since significant differences in the runtimes of solvers are fundamental in order for a given instance to benefit from the selection of a certain solver. This is also reflected in

[1] `http://fmv.jku.at/cadical/`

Table 1. Characteristics overview of considered argumentation frameworks.

	# Arguments	# Attacks	Density	Degree
Mean	409.95	14,441.19	0.088	70.45
Minimum	100.00	224.00	0.003	0.00
Maximum	1,499.00	204,110.00	0.702	726.00

the fact that all machine learning-based approaches for algorithm selection introduce some overhead, for example, for calculating instance features. If there is no significant difference in execution times, the potential time savings of selecting the fastest solver get nullified due to the mentioned overhead. Our analysis showed that the combination of the selected solvers yielded the best-balanced ratio of the best-solved instances per solver and exhibited significant differences in their runtimes. It should also be noted that the goal of the evaluation is to compare the machine learning algorithms for algorithm selection, and not the selected algorithms themselves.

To collect sufficient training and test data, we randomly generated a total of 6200 argumentation frameworks using three different generators of the ICCMA'17: *AFBenchGen2*, *SccGenerator*, and *StableGenerator*.

The *AFBenchGen2* [9] generator was used to create instances of (1) Erdös-Renyi [19] and (2) Watts-Strogatz [44] graphs. The *SccGenerator* aims to generate AFs with many strongly connected components, whereas the *StableGenerator* aims to generate AFs with many stable extensions (and therefore many preferred extensions). A detailed description of these generators can be found in [41]. All generators have been parameterized as described in [22]. We randomly selected a query argument for the $\mathsf{DS}_{\mathsf{PR}}$ task for each generated instance. A cutoff value of 600 seconds (10 minutes) per instance was imposed. For each solver, we recorded: (1) the number of solved instances, (2) the number of timed-out instances, (3) the number of crashed instances, and (4) the execution time per instance. The runtime of unsolved instances—timed-out or crashed—was set equal to the cutoff. A total of 785 instances were excluded because all solvers failed to solve them within the given time frame. Due to memory limitations of the experimental environment, another 230 (very large) instances had to be excluded, as they could not be processed. No further systematic exclusions were made. We randomly divided the remaining 5185 instances into separate training and test data following an 80%/20% train/test split[2]. Table 1 shows the characteristics of the considered AFs.

For each instance, the fastest solver determined its ground-truth label. We ran the experiments on a virtual machine running Ubuntu 20.04 with a 2.5 GHz Intel(R) Xeon(R) E5-2680 CPU and 60 GB of RAM.

We trained and evaluated four different supervised machine learning techniques: *k*-Nearest Neighbor (KNN), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM) (see Section 2.2), using *scikit-learn*[3] [35], a machine learning framework for Python[4]. As input for the classifiers, we use the three best features that Vallati et al. [43] identified for the classification task. Namely, these are (1) the number of vertices,

[2]Download: `https://fernuni-hagen.sciebo.de/s/UkL9WRjegFlGyhk`

[3]`https://scikit-learn.org/stable/index.htm`

[4]The KNN classifier was parameterized with `n_neighbors` = 17 and the RF classifier with `random_state` = 11, `min_samples_split` = 30, and `min_samples_leaf` = 10. The remaining classifiers (NB and SVM) were used with their default configurations.

(2) the density of the directed graph and (3) the minimum degree value of the directed graph. We also included the in- and out-degree of the query argument with regard to the corresponding argumentation framework as additional features.

Further, we trained and evaluated three widely adopted GNN models: Graph Isomorphism Networks (GIN) [45], Graph Convolutional Networks (GCN) [31], and GraphSage [25] (see Section 2.3). Each model features one pre-message passing layer (256-dim MLP), three message passing layers (determined by the respective GNN model) and two post-message passing layers (256-dim MLP). Each model is trained for 1000 epochs using the Adam [30] optimizer with a learning rate of 0.01 and batch normalization [29]. Results are reported for the final epoch. All models were trained without additional pre-calculated node, graph, or edge features. For the training and evaluation we use *GraphGym* [47], a platform for designing and evaluating GNNs.

To compare the performance of the approaches—both GNNs and classical ML methods—we use *accuracy*, *precision*, and *recall* to measure how accurate the predictions are. For all approaches the precision and recall are reported for each individual solver, following the definitions

$$\text{Precision}_s = \frac{TP_s}{TP_s + FP_s}$$
$$\text{Recall}_s = \frac{TP_s}{TP_s + FN_s},$$

where s denotes the class of the corresponding solver. True Positive (TP) are the elements that have been labeled as positive by the model and are actually positive, while False Positive (FP) are the elements labeled as positive, but they are actually negative. False Negative (FN) elements have been labeled as negative but are actually positive. The accuracy is defined as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TN denotes the True Negative elements, i.e., the elements that the model correctly labeled as negative[5]. In addition, we measure the overhead (CPU-time) of calculating the features (which is only necessary for the classical ML approaches), and the actual classification time. For the GNN models, we also measure the prediction times when utilizing a GPU instead of a CPU.

All models were trained and evaluated on a virtual machine running Ubuntu 20.04 with a 3.7 GHz AMD Ryzen 5 5600X 6-Core Processor, 32 GB of RAM and a NVIDIA GeForce RTX 3070 GPU with 8 GB of RAM. In order to accelerate the training process of the GNNs, training was carried out on the GPU.

4.2. Results

To begin with, we examine how accurate the predictions of both the classical ML methods and the GNN approaches are. For this purpose, we first draw a comparison to the work by Vallati et al. [43]. The authors report a precision of 0.68 wrt. *ArgSemSAT* (which

[5]Note that, unlike precision and recall, the accuracy is not calculated by class but across all classes.

Table 2. Overview of precision, recall, and accuracy wrt. the predictions made by the different ML and GNN techniques. Note that we abbreviated *ArgSemSAT* by "*ArgSS*".

	Precision			**Recall**			**Accuracy**
	ArgSS	*Fudge*	*μ-toksia*	*ArgSS*	*Fudge*	*μ-toksia*	
KNN	0.79	0.63	0.46	0.59	0.88	0.17	0.64
NB	0.77	0.59	0.38	0.46	0.89	0.18	0.60
RF	0.84	0.63	0.49	0.61	0.90	0.15	**0.65**
SVM	0.86	0.60	0.46	0.47	0.91	0.12	0.62
GCN	0.82	0.61	0.58	0.55	0.92	0.13	0.64
GIN	0.93	0.60	0.50	0.45	0.96	0.08	0.63
GraphSage	0.95	0.66	0.90	0.46	0.99	0.33	**0.71**

is the only solver considered in both their work and ours) when using an RF classifier with the three features that were determined to be the most expressive (see Section 3 for more details). Our experiments resulted in a precision of 0.83 when training an RF with the same three features, and a precision of 0.84 when using the in- and out-degree of the query nodes as additional features (see Table 2). Although the precision value wrt. *ArgSemSAT* is significantly higher in our experiments, we also observe a lower value wrt. the overall accuracy: Vallati et al. report an accuracy of 0.70 when using the three most expressive features, while our experiments only yield an accuracy of 0.64 (or 0.65 when additionally using in- and out-degree as features). This is due to the fact that the other two solvers we used in our experiments are predicted less precisely. As the upper part of Table 2, which includes the results regarding the classical ML methods, indicates, the precision values wrt. both *Fudge* and *μ-toksia* are lower than those wrt. *ArgSemSAT*, regardless of the selected ML method. Nevertheless, the precision values regarding *Fudge* are still higher than those regarding *μ-toksia* in all cases. Moreover, we can observe very low recall values wrt. *μ-toksia* (between 0.12 and 0.18), but rather high recall values wrt. *Fudge* (between 0.88 and 0.91). The reason for this is that a large number of *μ-toksia* instances are classified as *Fudge* instances—e.g., our RF classifier predicts 215 out of 260 *μ-toksia* instances to be *Fudge* instances. Further, the results show that the overall accuracy of the classical ML methods lies between 0.60 (NB) and 0.65 (RF).

With regard to the GNN methods, our experiments reveal similar results: again, *ArgSemSAT* tends to have the hightest precision, *μ-toksia* has rather low and *Fudge* a rather high recall, and a large number of *μ-toksia* instances are classified as *Fudge* (see the lower part of Table 2). However, GraphSage exhibits a slightly different behavior: here, the precision is higher wrt. *μ-toksia* (0.90) than wrt. *Fudge* (0.66). The overall accuracy values of the GNN approaches lie between 0.63 (GIN) and 0.71 (GraphSage), and are consequently a bit higher on average than those of the classical ML approaches.

The second aspect we aim to examine, besides classification accuracy, is the overall solving time. For each approach, we sum up the solving time of each individual test instance with regard to the respective predicted solver, to calculate the total solving time. For comparison we consider the time required by each solver when solving all test instances (see Table 3). Note that *Fudge* is clearly overall the fastest solver with 24,248 seconds, compared to 45,248 seconds (*ArgSemSAT*) and 77,734 seconds (*μ-toksia*). The results concerning the classical ML techniques are presented in the upper part of Table 4. We see that the overall solving time is quite similar with regard to KNN, RF, and SVM

Table 3. Overview of the total amount of time required to solve the entire test set wrt. each of the three solvers, as well as the number of times each solver was the fastest, and the number of times each solver produced a timeout. Note that each timeout added 600 seconds to the total solving time.

	Total solving time (s)	**Fastest**	**Timeouts**
ArgSemSAT	45,248.24	224/1037	52/1037
Fudge	24,720.96	553/1037	20/1037
μ-toksia	77,733.74	260/1037	80/1037

Table 4. Overview of the total amount of time required to solve the entire test set if the solvers predicted by each ML/GNN method were used, respectively. We additionally provide the number of times the fastest solver was predicted (which corresponds to the respective accuracy value), and the number of timeouts that still occur.

	Total solving time (s)	**Predicted fastest**	**Timeouts**
KNN	24,531.20	662/1037	20/1037
NB	27,794.33	624/1037	22/1037
RF	24,538.38	675/1037	20/1037
SVM	**24,506.66**	643/1037	20/1037
GCN	24,507.27	666/1037	20/1037
GIN	**24,410.00**	654/1037	20/1037
GraphSage	24,414.01	739/1037	20/1037

(between 24,507 and 24,538 seconds), only NB is considerably slower (27,794 seconds). The former three classifiers also perform an algorithm selection which results in a total solving time that is shorter than that of using *Fudge* (i.e., the fastest individual solver) for all instances. Only the NB classifier yields an algorithm selection which results is a longer solving time. However, NB also exhibited the lowest accuracy (0.60), which could explain this outcome. On the other hand, we notice that the SVM classifier produces the shortest total solving time, even though it did not feature the highest accuracy. Likewise, this effect can be observed wrt. the GNN methods. The lower part of Table 4 shows that GIN, which has the lowest accuracy of all GNN methods (0.63), produces a slightly lower total solving time than GraphSage, which has a significantly higher accuracy of 0.71. Overall, the GNN methods perform superior to the classical ML methods. Only the "fastest" classical ML method (SVM) accomplishes a slightly superior algorithm selection than the "slowest" GNN method (GCN).

As of yet, we only considered the solving time required by the selected algorithms. However, we additionally need to consider the time required for each prediction, and in the classical ML case also the time required to compute the features of each test instance. The latter amounts to an average of 0.0077 seconds per instance (i.e., 8.02 seconds for the entire test set). The time needed to predict all test instances is < 0.5 seconds for KNN, NB, and SVM, only the RF takes longer (5.2 seconds in total). The GNN approaches require significantly more time for the prediction process—they take between 58.4 seconds (GIN) and 68.2 seconds (GCN). However, these values were measured when the predictions we conducted on the CPU. When using the GPU, we can drastically reduce the prediction time to values between 0.9 seconds (GIN) and 1.3 seconds (GCN). Moreover, even if there was no GPU available, both GIN and GraphSage would still outperform all classical ML models. For instance, wrt. GIN, we have 24,410.0 seconds of to-

tal solving time plus 58.4 seconds of prediction time, i.e., a total of 24,468.4 seconds, which is already less than the shortest total solving time wrt. the classical ML methods (24,506.7 seconds), regardless of the additional time required for prediction and feature generation.

5. Conclusion

In the scope of this work, we examined different ML approaches for the task of algorithm selection in the field of abstract argumentation. We followed up on a study by Vallati et al. [43], who investigated the use of different sets of features to be used in ML methods for algorithm selection. Our work, on the other hand, does not focus on the selection of features, but on the selection of the classifier. Moreover, in addition to a number of "classical" ML methods (namely KNN, NB, RF, and SVM), we also considered three different graph neural networks (namely GCN, GIN, and GraphSage). For the classical ML methods, we used those three features which Vallati et al. determined to be the most expressive. To be precise, these are the number of vertices, the density of the directed graph, and the minimum degree value of the directed graph. In addition to these graph-based features, we used the in-degree and out-degree of the query nodes.

One noteworthy result of our experiments is that μ-*toksia* instances were often classified as *Fudge* instances, and that this effect did not only occur with the classical ML methods, which are feature-based, but also with the GNN methods, which are not given any pre-calculated features explicitly[6]. Nevertheless, our results demonstrated that neural networks are generally a useful approach for the task of algorithm selection. In particular, GIN and GraphSage performed superior to all classical ML approaches in terms of the total solving time of the predicted algorithms, even including a rather lengthy prediction time when no GPU is available. However, there is still room for improvement—if we always used the fastest solver, the total solving time would be 14,430 seconds, which is 40.9% less than our best result (see Table 4).

Furthermore, we examined the prediction accuracy of the different approaches and discovered that a higher accuracy does not automatically lead to a shorter overall solving time. This suggests that in some cases the classifier did not only fail to predict the fastest solver, but also failed to predict the second fastest (i.e., it predicted the slowest one). Hence, one idea to consider in future work is to introduce some sort of weighting which penalizes the prediction of a slower solver more than the prediction of a faster one during training. Regarding the classical ML methods, one could use a different set of features which is more suitable for the newly considered solvers (*Fudge* and μ-*toksia*). However, this might lead to an increase in feature generation time which must always be weighed against the potential reduction in solving time. Further issues which could be addressed in future work are the consideration of other argumentation semantics and tasks, as well as other GNN architectures.

[6] We also conducted an experiment in which we additionally fed the neural networks the same features that were used with the other ML methods. However, this did not improve our results.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Mario Alviano. The pyglaf argumentation reasoner. In *Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[3] Katie Atkinson, Pietro Baroni, Massimiliano Giacomin, Anthony Hunter, Henry Prakken, Chris Reed, Guillermo Simari, Matthias Thimm, and Serena Villata. Towards artificial argumentation. *AI magazine*, 38(3):25–36, 2017.

[4] Gilles Audemard and Laurent Simon. Lazy clause exchange policy for parallel sat solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 197–205. Springer, 2014.

[5] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, pages 370–418, 1763.

[6] S. Bistarelli, Fabio Rossi, and Francesco Santini. Conarg 2 : A constraint-based tool for abstract argumentation. 2015.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[9] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Generating structured argumentation frameworks: Afbenchgen2. In *COMMA*, pages 467–468, 2016.

[10] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. How we designed winning algorithms for abstract argumentation and which insight we attained. *Artificial Intelligence*, 276:1–40, 2019.

[11] Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. On the impact of configuration on abstract argumentation automated reasoning. *International Journal of Approximate Reasoning*, 92:120–138, 2018.

[12] Günther Charwat, Wolfgang Dvořák, Sarah A Gaggl, Johannes P Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 220:28–63, 2015.

[13] Dennis Craandijk and Floris Bex. Deep learning for abstract argumentation semantics. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, page 1667–1673, 2020.

[14] Dennis Craandijk and Floris Bex. Enforcement heuristics for argumentation with deep reinforcement learning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 2022.

[15] Padraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers-a tutorial. *ACM Computing Surveys (CSUR)*, 54(6):1–25, 2021.

[16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[17] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[18] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[19] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[20] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[21] Sara A Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Summary report of the second international competition on computational models of argumentation. *AI Magazine*, 39(4):77–79, 2018.

[22] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Benchmark selection at iccma'17. 2018.

[23] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 260–265. Springer, 2007.

[24] Nils Geilen and Matthias Thimm. Heureka: a general heuristic backtracking solver for abstract argumentation. In *International Workshop on Theorie and Applications of Formal Argumentation*, pages

143–149. Springer, 2017.

[25] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[26] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[27] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[32] Isabelle Kuhlmann and Matthias Thimm. Using graph convolutional networks for approximate reasoning with abstract argumentation frameworks: A feasibility study. In *International Conference on Scalable Uncertainty Management*, pages 24–37. Springer, 2019.

[33] Lars Malmqvist, Tommy Yuan, Peter Nightingale, and Suresh Manandhar. Determining the acceptability of abstract arguments with graph convolutional networks. In *SAFA@ COMMA*, pages 47–56, 2020.

[34] Andreas Niskanen and Matti Järvisalo. μ-toksia: An Efficient Abstract Argumentation Reasoner. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 800–804, 9 2020.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] J Ross Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1):71–72, 1996.

[37] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[38] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009.

[39] Matthias Thimm, Federico Cerutti, and Mauro Vallati. Fudge: A light-weight solver for abstract argumentation based on sat reductions. In *The Fourth International Competition on Computational Models of Argumentation (ICCMA'21)*, May 2021.

[40] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, 2017.

[41] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, 2017.

[42] Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. On the combination of argumentation solvers into parallel portfolios. In *Australasian Joint Conference on Artificial Intelligence*, pages 315–327. Springer, 2017.

[43] Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Predictive models and abstract argumentation: the case of high-complexity semantics. *The Knowledge Engineering Review*, 34, 2019.

[44] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[46] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[47] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. In *NeurIPS*, 2020.