Enhancing Abstract Argumentation Solvers with Machine Learning-Guided Heuristics: A Feasibility Study

Sandra Hoffmann[0009−0008−0294−6154], Isabelle Kuhlmann[0000−0001−9636−122X], and Matthias Thimm[0000−0002−8157−1053]

FernUniversität in Hagen, Hagen, Germany
{sandra.hoffmann,isabelle.kuhlmann,matthias.thimm}@fernuni-hagen.de

Abstract. Abstract argumentation frameworks model arguments and their relationships as directed graphs, often with the goal of identifying sets of arguments capable of defending themselves against external attacks. The determination of such admissible sets, depending on specific semantics, is known to be an NP-hard problem. Recent research has demonstrated the efficacy of machine learning methods in approximating solutions compared to exact methods. In this study, we leverage machine learning to enhance the performance of an exact solver for credulous reasoning under admissibility in abstract argumentation. More precisely, we first apply a random forest to predict acceptability, and subsequently use those predictions to form a heuristic that guides a search-based solver. Additionally, we propose a strategy for handling varying prediction qualities. Our approach significantly reduces both the number of backtracking steps and the overall runtime, compared to standard existing heuristics for search-based solvers, while still providing a correct solution.

Keywords: Abstract argumentation · Heuristics · Random forest.

1 Introduction

Argumentation is central for human communication and interaction, hence there are various strategies of implementing this concept in approaches to artificial intelligence. In the field of *abstract argumentation* [8], the underlying idea is to focus on the interplay between arguments and counterarguments rather than on the content of the arguments themselves. The core formalism in this field is the *abstract argumentation framework*, which can be understood as a directed graph in which the nodes represent the given arguments, and the edges represent an attack relation between them.

Figure 1 shows an example of such a framework. Semantics are commonly expressed through so-called *extensions*, which are sets of arguments that jointly fulfill certain conditions. A fundamental semantics in the field of abstract argumentation is the concept of admissibility. In order to be an admissible extension, arguments in the set must not attack each other (i. e., the set must be *conflict-free*) and they have to defend each other from all outside attacks.

Typical problems in abstract argumentation include the problem of deciding whether an argument is included in at least one extension (or all extensions) wrt. a specific semantics, or the problem of determining an extension or enumerating all of them wrt. a specific semantics.

The literature already provides different families of (exact) approaches to solve the above-mentioned reasoning problems in abstract argumentation. One such family consists of reduction-based approaches—see, e. g., [21,17,24,1,9]—which encode a given problem in a different formalism—e. g., as a Boolean satisfiability problem—and then use an existing solver for that formalism. Another family of approaches consists of backtracking-based methods that make use of heuristics to guide the search procedure—see, e. g., [22,12,23].

Since most of the reasoning problems in abstract argumentation are computationally hard [10], this can result in exceedingly long runtimes when using an exact algorithm. To counteract this issue, machine learning-based approaches have been proposed in the literature [15,19,6,7]. However, although these approaches proved to be significantly faster than their exact counterparts, they are not guaranteed to yield correct results (for a deeper analysis, see also [16]). Thus, the main advantage of an exact method (such as a reduction- or backtracking-based approach) is that it always provides correct results, while the main advantage of a (purely heuristic) machine learning-based approach is its runtime performance. An approach for combining these advantages is the use of machine learning techniques to predict the "best" exact solver from a portfolio [25,14]. In the work at hand, we aim to harness the advantages of machine learning methods in a different manner. More precisely, we use predictions made by a machine learning model in order to inform a heuristic that guides a backtracking-based approach which ultimately yields a correct result. As an example for the overall approach we consider the task of deciding whether a given argument is accepted under admissibility [8], which is a core aspect in many reasoning problems. In an experimental evaluation we compare the use of our machine learning-based heuristic (using a random forest) to the standard heuristic of the backtracking-based solver Heureka [12], and we demonstrate that both the number of backtracking steps as well as the overall runtime can be reduced when our newly proposed heuristic is applied.

To summarize, our contributions are as follows:

- We present an approach that exploits the strengths of both machine learning and reasoning techniques by using machine learning-based predictions to create a heuristic which can accelerate an exact, backtracking-based solver.
- Our approach offers a flexible solution, as both the machine learning component and the backtracking-based solver can be specified as desired.
- In an experimental analysis, we show that our approach leads to a significant decrease in both the number of backtracking steps and overall runtime, when compared to a standard heuristic.

The remainder of the paper is organized as follows. In Section 2 we provide some preliminaries on the topic of abstract argumentation. After giving an overview on current solution approaches for problems in abstract argumentation

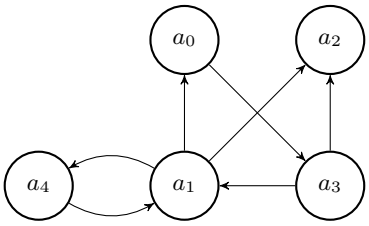


Fig. 1. An abstract argumentation framework.

in Section 3, we propose a machine learning-guided heuristic in Section 4. An extensive experimental analysis is presented in Section 5, Section 6 details the limitations of our research and finally we conclude in Section 7.

2 Preliminaries

An *abstract argumentation framework* (AF) [8] is a tuple $F = (\mathsf{Args}, R)$, with Args being a set of arguments and $R \subseteq \mathsf{Args} \times \mathsf{Args}$ defining an attack relation. An argument $a \in \mathsf{Args}$ *attacks* another argument $b \in \mathsf{Args}$ if $(a, b) \in R$. On the other hand, an argument $a \in \mathsf{Args}$ is *defended by* a set of arguments $E \subseteq \mathsf{Args}$ if for all $b \in \mathsf{Args}$ with $(b, a) \in R$, there exists a $c \in E$ with $(c, b) \in R$.

An *extension* is a set of arguments that are jointly acceptable, given a set of conditions. Exactly which conditions need to be satisfied is determined by a *semantics*. There exists a multitude of different semantics in the literature, however, we focus on the *preferred* semantics introduced in the seminal paper by Dung [8].

Definition 1. *Let $F = (\mathsf{Args}, R)$ be an argumentation framework. A set $E \subseteq \mathsf{Args}$ is*

- *conflict-free if there are no $a, b \in E$ such that $(a, b) \in R$,*
- *admissible if E is conflict-free and each $a \in E$ is defended by E within F,*
- *complete if every argument $a \in \mathsf{Args}$ defended by E is also included in E,*
- *preferred if E is a $\subseteq$-maximal complete extension, and*
- *grounded if E is a $\subseteq$-minimal complete extension.*

Note that the grounded extension is uniquely determined [8]. Typical decision problems in the area of abstract argumentation include the problem of deciding whether a given argument is included in at least one extension (*credulous acceptability*) or all extensions (*skeptical acceptability*) wrt. a given semantics. In the following, we denote the problem of deciding credulous acceptability wrt. preferred semantics as DC. Note that this problem is equivalent to the problems of deciding credulous acceptability under admissible, and under complete semantics.

Algorithm 1: Backtracking-based algorithm SEARCH for checking credulous acceptance wrt. admissibility

Data: $F = (\mathsf{Args}, R)$, $S_{in}, S_{out} \subseteq \mathsf{Args}$
Result: TRUE if there is admissible S' with $S_{in} \subseteq S'$.
1 **if** S_{in} *is not conflict-free* **then**
2 **return** FALSE
3 **if** S_{in} *is admissible* **then**
4 **return** TRUE
5 Pick $a \in \mathsf{Args} \setminus (S_{in} \cup S_{out})$
6 **return** $\mathsf{SEARCH}(F, S_{in} \cup \{a\}, S_{out})$ OR $\mathsf{SEARCH}(F, S_{in}, S_{out} \cup \{a\})$

Example 1. Consider the AF illustrated in Figure 1. The conflict-free sets of this AF are

$$\{\emptyset, \{a_0\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_4\},$$
$$\{a_0, a_2\}, \{a_0, a_4\}, \{a_2, a_4\}, \{a_3, a_4\},$$
$$\{a_0, a_2, a_4\}\}.$$

Out of these sets, only $\emptyset$, $\{a_4\}$, $\{a_0, a_4\}$, and $\{a_0, a_2, a_4\}$ defend themselves, and are thus admissible. Further, the only complete sets are $\emptyset$ and $\{a_0, a_2, a_4\}$, which makes the grounded extension (i.e., the $\subseteq$-minimal complete extension) $\emptyset$, and the set of preferred extensions (i.e., the $\subseteq$-maximal complete extensions) consists only of $\{a_0, a_2, a_4\}$. We can also see that the set of arguments contained in at least one admissible set (i.e., the set of credulously acceptable arguments wrt. admissibility) is $\{a_0, a_2, a_4\}$, which is equal to the set of credulously accepted arguments under complete or preferred semantics.

3 Solution Approaches in Abstract Argumentation

The methods employed to address decision problems in abstract argumentation can be broadly categorized into reduction-based or direct approaches [5]. Reduction-based solvers operate by translating the reasoning problem into other formalisms, such as answer-set programming [9,11], constraint-satisfaction problems [4,2] or Boolean satisfiability [21,26], leveraging existing solvers in those domains. The advantage of the reduction-based approach lies in the high efficiency of these existing solvers. On the other hand, direct approaches involve the implementation of a dedicated algorithm tailored to the structure of AFs, often utilizing backtracking. Direct solvers retain the structural information of the AF, allowing them to exploit specific shortcuts relevant to certain semantics [5].

Algorithm 1 describes a simple backtracking strategy to assess argument justification. Given an AF F and two argument sets S_{in} and S_{out}, the algorithm recursively explores potential admissible sets by considering the inclusion or exclusion of individual arguments. It terminates and returns FALSE if S_{in} is not

conflict-free, ensuring the absence of internal attacks. If S_{in} is already admissible, the algorithm returns TRUE. Otherwise, it selects an argument a from the remaining set of arguments and recursively follows two branches: one including a in S_{in} and maintaining S_{out}, and another excluding a from S_{in} and incorporating a into S_{out}. If the search in the first branch succeeds the second branch does not have to be explored. If the search in the first branch fails, we say that the algorithm *backtracks* and it is required to continue with the second branch. The algorithm returns TRUE if either branch results in an admissible set. The algorithm can determine if an argument a is contained in at least one admissible/preferred/complete extension by calling it with $\mathsf{SEARCH}(F, \{a\}, \emptyset)$.

The order in which arguments are processed—i. e. how argument a is determined in line 5 of Algorithm 1—plays a crucial role in the algorithm's performance, and different heuristics can be employed for this purpose. The algorithm we use in our study specifies the order in a deterministic way. The selected heuristic calculates a confidence value for each argument. Subsequently, these values are arranged in descending order to establish the total ordering.

Example 2. Consider again the AF in Figure 1, which depicts an AF with the preferred extension $\{a_0, a_2, a_4\}$, and the task to decide DC wrt. a_2. Assuming the order determined by a certain heuristic is $(a_1, a_3, a_4, a_0, a_2)$[1], the binary tree visualizing the recursive calls needed to solve this task using Algorithm 1 has a depth of 4. In contrast, building the order based on a perfect prediction of each argument's acceptability yields a depth of 2, thereby enhancing the algorithm's efficiency.

Note that Algorithm 1 only showcases the general principle of search-based algorithms. Existing search-based solvers [22,12,23] are more involved and rely on similar techniques as DPLL- and CDCL-solvers from satisfiability solving [3].

4 Machine Learning-Guided Heuristics

The goal of this paper is to improve the performance of a direct solver by reducing the number of backtracking steps necessary to decide DC wrt. a given argument. In order to do so, we employ a machine learning classifier and use the obtained predictions to guide a direct solver. For our research, we decided to predict the overall acceptance status of an argument and use this prediction, along with a confidence measure, to build our heuristic. This heuristic determines the order in which the search algorithm processes the arguments. One might question why we did not employ the classifier to directly predict the optimal order for each argument, thereby eliminating the need for a priority heuristic altogether. However, it is crucial to acknowledge that for each argument, there exist multiple ideal orderings that would effectively guide the solver.

Returning to Example 2, we determined a_0, a_2, a_4 to be the preferred extension of this AF. However, when deciding DC with respect to a_2, it does not

[1] This is the order determined by the standard heuristic used in Heureka [12].

matter whether we first pass a_0 or a_4 to the algorithm. Another possible approach would be to aim to directly predict admissible sets. The author in [18] describes a similar approach by training a graph neural network to predict which arguments are jointly admissible and then use this information to guide a SAT-based solver. While this approach yielded promising results and provides opportunities for further study, it also requires extensive neural network training, whereas our goal was to investigate whether a lightweight solution could already provide substantial improvements. More information on the training process is provided in Section 5.1.

We pass the obtained prediction outcomes as input to a direct solver and use them to develop a heuristic that prioritizes arguments based on their predicted acceptability. Arguments predicted with higher confidence to be accepted wrt. DC are processed first, while those likely to be rejected are processed last.

We compare the results to those obtained using a heuristic that has demonstrated effective performance for DC in prior research [12].

Following an analysis of some initial experiments, we refine our approach by crafting a heuristic tailored specifically to the query argument. Subsequently, we assess the performance on further datasets.

To determine whether an argument a is acceptable wrt. DC, we need to find a preferred extension that contains a. In contrast, to prove that a is not acceptable wrt. DC, we need to establish that there cannot be a preferred extension containing a. As any conflict with the grounded extension signifies the rejection of a, our strategy in aiming to prove that a is not acceptable wrt. DC revolves around assuming a, as well as all arguments belonging to the grounded extension, are acceptable wrt. DC and devising a heuristic prioritizing arguments likely not being accepted. This approach aimed to ensure conflicts happen early on in the justification process, in order to enhance overall performance. While initial experiments showed promising outcomes in AFs with substantial grounded extensions, the efficacy diminished in AFs with an empty grounded extension. Even when restricting the heuristic's application only to AFs with non-empty grounded extensions, the results failed to significantly surpass those of the standard heuristic. This may be attributed to the fluctuation in prediction accuracy when considering related arguments. As detailed in Section 5, although our RF-model is able to classify most arguments correctly, when constructing a heuristic centered on a specific argument, substantial penalties can occur for inaccuracies in predicting related arguments. Additional research is needed to devise a robust heuristic for rejected arguments. Consequently, we exclusively consider accepted arguments in the subsequent sections of this paper.

5 Experimental Analysis

The following section offers an overview of the datasets we utilized, outlines our experimental setup, and presents the results obtained from our experiments.

5.1 Datasets and Setup

To conduct our experiments, we utilized the *kwt-train* and *kwt-test* datasets generated using the *KwtDungTheoryGenerator*[2] as described in [16]. Each graph in these datasets consists of 151 arguments, and the training and test set each contain 1000 graphs.

To assess the performance of our heuristic on larger datasets, we employed the *KwtDungTheoryGenerator* to generate a more extensive dataset called *kwt-large*. This new dataset comprises 10,000 graphs designated for training (*kwt-large-train*) and an additional 1000 graphs reserved for testing (*kwt-large-test*). The graphs within this dataset span a range of 100 to 300 arguments, with a total of 148,483 accepted arguments within the *kwt-large-test* set.

In our research, the primary emphasis lies on enhancing the performance of arguments that are credulously accepted. Accordingly, we sought to employ a graph type that featured a substantial number of accepted arguments for our third dataset. To achieve this, we harnessed the *AFBenchGen* graph generator[3] to create a supplementary set of 10,000 Barabasi graphs for training (*Barabasi-train*) and an additional 1,000 Barabasi graphs for testing (*Barabasi-test*). These graphs encompassed argument quantities ranging from 100 to 500, resulting in a total of 252,502 accepted arguments within the *Barabasi-test* set[4].

Previous research has suggested that standard machine learning classifiers are useful in predicting the acceptability status of arguments in an AF [16,13]. In [13] random forest (RF) classifiers trained using a comprehensive feature set provided the best results. This feature set comprised 10 node- and graph-based properties, namely the degree, closeness, Katz [20], and betweenness centrality as well as the number of the strongly connected components (SCC) of the AF, the size of the SCC each argument is part of, the average degree of the AF and whether it is irreflexive, strongly connected or aperiodic.

Building on these results, we trained individual RF classifiers for each dataset. The training and testing procedures were executed using Python, making use of the *scikit-learn*[5] and *networkx*[6] libraries. For a detailed overview of all three datasets, please refer to Table 1. To quantify the efficacy of our classification results, we use the standard metrics of *accuracy*, *recall* (also referred to as *true positive rate* (TPR)), *specificity* (also referred to as *true negative rate* (TNR)), and *precision*, as well as the *Matthews Correlation Coefficient* (MCC). We define a *true positive* (TP) as an argument in an AF that is accepted wrt. DC and was correctly classified as such. Accordingly, a *true negative* (TN) is a non-accepted argument that is correctly classified as such, and *false positives/negatives* (FP/FN) are the corresponding falsely classified counterparts.

[2] http://tweetyproject.org/r/?r=kwt_gen
[3] https://sourceforge.net/projects/afbenchgen/
[4] The datasets, the enhanced Heureka code and the individual results are available here: http://mthimm.de/misc/hkt_ratio24.zip
[5] https://scikit-learn.org/stable/
[6] https://networkx.org/

Table 1. Overview of the *kwt*, *kwt-large* and *Barabasi* datasets.

Dataset	No of graphs	No of nodes	YES nodes	NO nodes
kwt-train	1,000	151,000	113,539	37,461
kwt-test	1,000	151,000	112,909	38,091
kwt-large-train	10,000	2,210,000	1,574,194	635,806
kwt-large-test	1,000	220,342	148,483	71,859
Barabasi-train	10,000	3,000,000	2,524,352	475,648
Barabasi-test	1,000	300,000	252,502	47,498

Table 2. Results for classifying the *kwt*, *kwt-large* and *Barabasi* test sets using an RF classifier trained on a total of 10 graph features.

Dataset	MCC	Accuracy	Recall (TPR)	Specificity (TNR)	Precision
kwt-test	0.987	0.995	0.994	1	1
kwt-large-test	0.990	0.996	0.994	1	1
Barabasi-test	0.792	0.947	0.980	0.771	0.958

Accuracy is defined as $\frac{\mathsf{TP}+\mathsf{TN}}{\mathsf{TP}+\mathsf{TN}+\mathsf{FP}+\mathsf{FN}}$, precision as $\frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FP}}$, TPR as $\frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FN}}$, TNR as $\frac{\mathsf{TN}}{\mathsf{TN}+\mathsf{FP}}$, and MCC as $\frac{\mathsf{TP}\cdot\mathsf{TN}-\mathsf{FP}\cdot\mathsf{FN}}{\sqrt{(\mathsf{TP}+\mathsf{FP})\cdot(\mathsf{TP}+\mathsf{FN})\cdot(\mathsf{TN}+\mathsf{FP})\cdot(\mathsf{TN}+\mathsf{FN})}}$.

We decided on using the Heureka solver [12] due to its implementation of a direct solution approach and its flexibility in incorporating custom heuristics to determine the order of arguments. The objective of the heuristic is to assign a real-number value to each argument through a mapping function. A higher value indicates a higher priority for a particular argument, influencing its processing order in the justification process. Specifically for DC, Heureka employs a standard heuristic that emphasizes arguments within strongly connected components, combining this with a path-based component.

In our experiments, Heureka is executed on each argument within our test sets, capturing both runtime and backtracking steps for individual arguments. The standard heuristic serves as a benchmark for comparing the outcomes of our experiments. To control the overall runtime for each dataset, a timeout of 10 minutes per argument is implemented.

5.2 Initial Experimental Analysis

We begin our experiments by training an RF classifier for each dataset. An overview of the classification metrics is provided in Table 2.

Our initial approach involves simply prioritizing the arguments predicted to be acceptable wrt. DC. The order of argument processing is determined by calculating a score for each argument based on the percentage of trees that favor the assigned label. If an argument is predicted to not be contained in any extension, we prioritize arguments with predictions that are close to the decision

Table 3. Results for classifying DC arguments in the kwt Dataset using the standard Heureka heuristic as well as a simple prediction-based ordering

MCC	Standard Backtracks	Prediction Backtracks	no of AFs
>0.7	39,919	1,082,091	8
>0.8	168,312	50,328,039	58
>0.9	10,033	1,531,500	33
1	280,676	0	898
Total	498,940	52,941,630	997

boundary. We evaluate all arguments that are acceptable wrt. DC in the *kwt-test* set using both the prediction-based ordering and the standard heuristic.

The results, presented in Table 3, indicate that the simple ordering we employed successfully reduced the need for backtracking in cases with relatively accurate predictions, however, this approach severely penalizes wrong predictions, which led to an overall increase in backtracking steps. Additionally, using the simple ordering heuristic, Heureka was unable to solve three AFs within the 10-minute time limit per argument.

To gain deeper insights into the limitations of this approach, we conducted a detailed analysis of the AF that required the highest number of backtracking steps. While applying the prediction-based ordering resulted in a staggering 21,482,709 backtracking steps, the standard heuristic was able to resolve this AF without any backtracking.

Upon closer examination, we discovered that out of the 151 arguments in this AF, only 9 specific arguments were responsible for all the backtracking steps. These 9 arguments were the sole accepted arguments that were erroneously predicted as not accepted by our model. Furthermore, all of these arguments belonged to the same extension, and critically, none of them belonged to any other extension. As a result, these crucial arguments, which would be highly valuable for guiding our search algorithm, ended up being processed toward the end of the solving process. Consequently, a straightforward ordering approach proved to be insufficient. To reduce the overall number of required backtracking steps, a more refined heuristic is needed.

To establish whether an argument a is acceptable wrt. DC, we must identify a preferred extension E that contains a. Therefore, we want to prioritize arguments that are most likely part of E. We thus seperate our AF into three distinct sets: Arguments that are likely not in E (*outExt*), arguments that defend a (*defenders*) and thus have the highest chance to be in E, and arguments that might be in E (*possibleExt*). Our refined algorithm starts by adding all arguments that are in conflict with a to the *outExt* set. Likewise, arguments predicted to be outside any extension are categorized within *outExt*. Subsequently, we then iterate through the remaining arguments, identifying whether arguments act as defenders of a by attacking its attackers or whether they undermine E by targeting arguments likely to be part of E. Arguments that do not fall into the

categories of defenders or offenders are placed in the *possibleExt* set. Once all arguments are processed we determine the heuristic order, making sure, that all *defenders* are processed first. This is ensured by multiplying the prediction probability of each argument in a set by a dedicated factor for said set. Let the factors used to multiply the prediction confidence values be denoted as x, y, z for the *defenders*, *possibleExt* and *outExt* sets, respectively. The actual value of the factors is arbitrary, as long as the following conditions hold: $\mathsf{x} > \mathsf{y}$ and $\mathsf{z} > 1$. For more detailed information, please refer to Algorithm 2. In our experiments we set $\mathsf{x} = 1000$, $\mathsf{y} = 100$, and $\mathsf{z} = 2$.

Algorithm 2: MLPred Heuristic for accepted Arguments

Data: AF aaf, Prediction *pred*, Query Argument a
Result: Heuristic h

$attackRelation \leftarrow$ AttackRelation(aaf);
$attackers \leftarrow attackRelation$.attacker_set($a$);
$attackeds \leftarrow attackRelation$.attacked_set($a$);
$outExt \leftarrow attackers \cup attackeds$;
$possibleExt \leftarrow itemIndex$;
$defenders \leftarrow \emptyset$;
for $i = 0; i <$ *pred.args.size()-1*$; i++$ **do**
  $curAttackeds \leftarrow attackRelation$.attacked_set($i$);
  $argIsDefender \leftarrow curAttackeds \cap attackers$;
  $argIsOffender \leftarrow curAttackeds \cap possibleExt$;
  if *pred.predictLabel*[i] == *YES* **then**
    if *argIsOffender* **then**
      $outExt \leftarrow i$;
      continue;
    else if *argIsDefender* **then**
      $defenders \leftarrow i$;
    else
      $possibleExt \leftarrow i$;
    $attackers \leftarrow attackRelation$.attacker_set($i$);
    $outExt \leftarrow attackers \cup curAttackeds$;
  else
    $outExt \leftarrow i$;
for *arg in defenders* **do**
  $h.order[arg] \leftarrow pred.predProb[arg] * \mathsf{x}$
for *arg in possibleExt* **do**
  $h.order[arg] \leftarrow pred.predProb[arg] * \mathsf{y}$
for *arg in outExt* **do**
  $h.order[arg] \leftarrow pred.predProb[arg] * \mathsf{z}^{-1}$

Table 4. Results for classifying DC arguments in the kwt Dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2.

MCC	Standard Backtracks	Prediction Backtracks	no of AFs
>0.7	39,919	358,246	8
>0.8	185,587	4,320,665	60
>0.9	10,101	1,412,926	34
1	280,676	0	898
Total	516,283	6,091,837	1000

Table 5. Results for classifying the kwt Dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2 with a threshold of 0.35.

MCC	Standard Backtracks	Prediction Backtracks	no of AFs
>0.7	39,919	31,976	8
>0.8	185,587	226,790	60
>0.9	10,101	8,029	34
1	280,676	0	898
Total	516,283	266,795	1000

Running Heureka using this refined approach yielded a significant reduction in backtracking, nearly reaching a 90 % reduction, and enabling Heureka to successfully solve all argumentation AFs within the allocated time, as shown in Table 4. However, when evaluated against the standard heuristic, it is evident that the total number of backtracking steps, though significantly improved, still falls short of matching the performance of the standard heuristic.

Within our dataset, all instances of backtracking occur in AFs where the predictive accuracy is not perfect. As we have observed during our in-depth analysis of an individual AF, the quality of predictions can vary not only between AFs but also among arguments within the same AF. Therefore, we require a method to make an informed choice of whether we can rely on the predictions generated by the machine learning model to effectively guide Heureka.

In our algorithm, the *defenders* set comprises the most critical arguments, as these directly support our query argument a. We operate on the assumption that a larger *defenders* set implies a more informative prediction for guiding Heureka. We also employ a threshold parameter below, which we opt to use the standard heuristic instead of the prediction. More specifically, this threshold dictates the required size of the *defenders* set in relation to the *possibleExt* set. In our experiments, we employed a threshold of 0.35. Re-running Heureka with this threshold produced the results presented in Table 5.

By implementing the threshold to filter out uninformative predictions, we successfully reduced the number of backtracking steps by nearly 50 %. In the following section we will evaluate our initial results using larger, more diverse datasets.

Table 6. Results for classifying the *kwt-large* dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2 with a threshold of 0.35.

MCC	Standard Backtracks YES	No of AFs	Prediction Backtracks YES	No of AFs
>0.8	42,088,262	31	37,881,987	34
>0.9	90,335,877	60	16,892,061	64
1	1,287,751,506	804	0	896
Total	1,420,175,645	895	54,774,048	994

Table 7. Runtime in seconds for classifying the *kwt-large* dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2 with a threshold of 0.35.

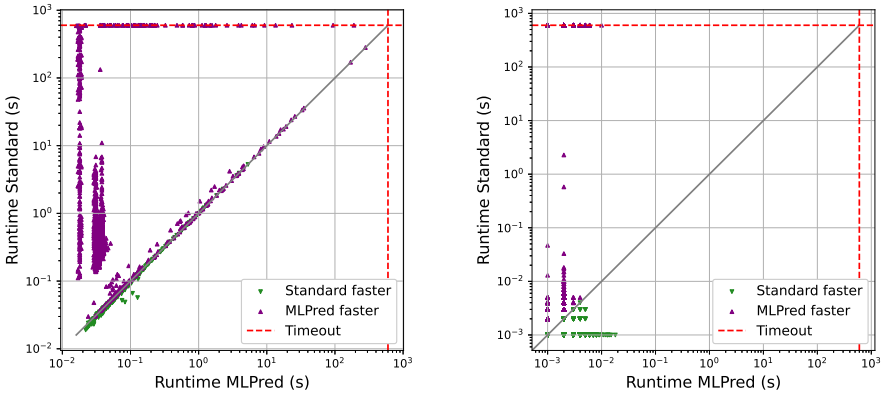
MCC	Runtime Standard	Runtime MLPred	No of AFs
> 0.8	574	552	31
> 0.9	2,148	475	60
1	30,134	3,885	804
Total	32,856	4,911	895

5.3 Evaluation and Results

The first evaluation experiment involved running Heureka on the *kwt-large* dataset using the same prediction quality threshold of 0.35. The MLPred heuristic resulted in a substantial reduction of backtracking steps required for justifying accepted arguments compared to the standard heuristic, as demonstrated in Table 6. Notably, the MLPred heuristic enabled heureka, to successfully solve 994 AFs, whereas the standard heuristic was only able to solve 895 AFs without encountering timeouts.

We also experienced a drastic decrease in runtime when using the MLPred heuristic. Table 7 shows an overview over the runtime in seconds needed to solve the 895 AFs that both heuristics were able to solve completely. The MLPred heuristic achieved a runtime reduction of 85 %. The performance gain achieved by using the MLPred heuristic is also evident, when comparing the runtime for individual arguments. Figure 2a shows the runtime comparison for both heuristics. To limit the overview to instances of a certain difficulty, we only plot arguments, where the amount of backtracking steps exceeds the mean amount of backtracking steps for at least one heuristic. We can clearly see, that on the vast majority of arguments the MLPred heuristic outperformed the standard heuristic.

In order to assess the performance of the prediction heuristic on a different graph type, we ran the same experiments on the *Barabasi-test* Dataset. Again, the prediction heuristic resulted in a significant reduction in the need for



(a) Runtime per argument for the *Kwt-large* dataset.

(b) Runtime per argument for the *Barabasi* dataset.

Fig. 2. Runtime per argument for arguments with above-average backtracking steps

backtracking. In fact, as can be seen in Table 8, the need for backtracking was eliminated almost completely.

We also compared the runtime for both heuristics for the *Barabasi* dataset. Interestingly, as is evident in Table 9 the standard heuristic overall was the faster choice for the AFs both heuristics could solve. This stems from the fact that Heureka needs more time to parse and build the MLPred heuristic. As the graphs in this dataset in general can be solved much faster than those in the *Kwt-large* dataset, the decreased runtime for the justification process is not enough to outweigh the overhead added by using the prediction.

However, when comparing the runtime for the individual arguments in Figure 2b, we can see that for the hardest arguments of this dataset the MLPred heuristic performed better. Combined with the fact, that the MLPred heuristic was able to solve all AFs of this dataset we can still conclude that using a machine learning prediction was beneficial when solving the *Barabasi* dataset.

6 Limitations

Our study primarily focuses on enhancing the solution runtime for arguments classified as DC. While we successfully utilized machine learning predictions to guide the Heureka solver in justifying rejected arguments in several test cases, our approach did not yield satisfactory results when applied to a larger number of arguments. Additionally, our investigation only considered credulous acceptance under preferred semantics. Furthermore, we limited our analysis to two different graph types, namely *kwt* and *barabasi* graphs.

Table 8. Results for classifying the *Barabasi* dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2 with a threshold of 0.35.

MCC	Standard Backtracks YES	No of AFs	Prediction Backtracks YES	No of AFs
>0.5	0	4	2	4
>0.6	1,080	53	362	53
>0.7	2,521,167	467	2,843	481
>0.8	236,081,763	428	1,774	433
>0.9	1,510	29	63	29
Total	238,605,520	981	5,044	1,000

Table 9. Runtime in seconds for classifying the *Barabasi* dataset using the standard Heureka heuristic as well as the MLPred heuristic explained in Algorithm 2 with a threshold of 0.35.

MCC	Runtime Standard	Runtime MLPred	No of AFs
> 0.5	0	0	4
> 0.6	7	17	53
> 0.7	113	294	467
> 0.8	156	243	428
> 0.9	3	6	29
Total	278	561	981

While *kwt* graphs are intentionally designed to pose a challenge wrt. deciding DC under preferred semantics, and *barabasi* graphs are advantageous to our study due to their tendency to contain a large number of accepted arguments, it would be beneficial to assess the efficacy of our approach on other graph types in the future. This assessment should specifically include graphs used as benchmarks in the *International Competition on Computational Models of Argumentation*[7], enabling a direct comparison to other state-of-the-art solvers.

We chose a lightweight approach, employing a standard random forest classifier trained on different graph properties. Although the classification results were reasonably good, more advanced techniques such as neural networks have demonstrated even better results and could, therefore, prove beneficial in our pursuit to improve the runtime of justification algorithms.

7 Conclusion

The goal of our research was to improve the runtime of a search-based solver, by reducing the backtracking steps needed to justify whether an argument is DC.

[7] http://argumentationcompetition.org/index.html

Our study revealed that using machine learning predictions to assist a search-based solver leads to notable advantages in minimizing backtracking steps and improving runtime in decision-making processes, specifically in the context of argument acceptance. The integration of machine learning resulted in a significant reduction of backtracking steps, achieving a minimum reduction of 96 %. Across all datasets examined, the overall runtime could be decreased by up to 85 %. Furthermore, our approach was able to enhance the solvability of argumentation frameworks within a specified time constraint.

Further research opportunities could involve combining the classifier and the solver into a standalone application, eliminating the necessity to provide the solver with external predictions. However, given that the prediction quality of the RF classifier depends on the similarity between training and testing data, exploring alternative classifiers becomes imperative. Existing studies propose that employing graph neural networks holds promise for achieving robust prediction results.

Notably, our research did not yield significant improvements for rejected arguments. Subsequent investigations should look into strategies to effectively apply predictions to rejected arguments.

Acknowledgement

The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grant 375588274).

References

1. Alviano, M.: The pyglaf argumentation reasoner. In: Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
2. Amgoud, L., Devred, C.: Argumentation frameworks as constraint satisfaction problems. In: Scalable Uncertainty Management: 5th International Conference, SUM 2011, Dayton, OH, USA, October 10-13, 2011. Proceedings 5. pp. 110–122. Springer (2011)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
4. Bistarelli, S., Santini, F.: Modeling and solving afs with a constraint-based tool: Conarg. In: Modgil, S., Oren, N., Toni, F. (eds.) Theorie and Applications of Formal Argumentation. pp. 99–116. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
5. Cerutti, F., Gaggl, S.A., Thimm, M., Wallner, J.P.: Foundations of implementations for formal argumentation. In: Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L. (eds.) Handbook of Formal Argumentation, chap. 15. College Publications (February 2018)
6. Craandijk, D., Bex, F.: Deep learning for abstract argumentation semantics. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. p. 1667–1673 (2020)
7. Craandijk, D., Bex, F.: Enforcement heuristics for argumentation with deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 5573–5581 (2022)

8. Dung, P.M.: On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artificial Intelligence **77**(2), 321–358 (1995)
9. Dvořák, W., Rapberger, A., Wallner, J.P., Woltran, S.: Aspartix-v19-an answer-set programming based system for abstract argumentation. In: International Symposium on Foundations of Information and Knowledge Systems. pp. 79–89. Springer (2020)
10. Dvořák, W., Dunne, P.E.: Computational problems in formal argumentation and their complexity. In: Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L. (eds.) Handbook of Formal Argumentation, chap. 14. College Publications (February 2018)
11. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. Argument & Computation **1**(2), 147–177 (2010). https://doi.org/10.1080/19462166.2010.486479
12. Geilen, N., Thimm, M.: Heureka: a general heuristic backtracking solver for abstract argumentation. In: Theory and Applications of Formal Argumentation: 4th International Workshop, TAFA 2017, Melbourne, VIC, Australia, August 19-20, 2017, Revised Selected Papers 4. pp. 143–149. Springer (2018)
13. Hoffmann, S.: Investigating the influence of graph properties on the prediction quality of machine learning methods in the context of abstract argumentation
14. Klein, J., Kuhlmann, I., Thimm, M.: Graph neural networks for algorithm selection in abstract argumentation. In: ArgML@ COMMA. pp. 81–95 (2022)
15. Kuhlmann, I., Thimm, M.: Using graph convolutional networks for approximate reasoning with abstract argumentation frameworks: A feasibility study. In: International Conference on Scalable Uncertainty Management. pp. 24–37. Springer (2019)
16. Kuhlmann, I., Wujek, T., Thimm, M.: On the impact of data selection when applying machine learning in abstract argumentation. In: Computational Models of Argument, pp. 224–235. IOS Press (2022)
17. Lagniez, J.M., Lonca, E., Mailly, J.G.: Coquiaas: A constraint-based quick abstract argumentation solver. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 928–935. IEEE (2015)
18. Malmqvist, L.: Approximate Solutions to Abstract Argumentation Problems Using Graph Neural Networks. Ph.D. thesis, University of York (2022)
19. Malmqvist, L., Yuan, T., Nightingale, P., Manandhar, S.: Determining the acceptability of abstract arguments with graph convolutional networks. In: SAFA@ COMMA. pp. 47–56 (2020)
20. Newman, M.: Networks. Oxford university press (2018)
21. Niskanen, A., Järvisalo, M.: µ-toksia: An Efficient Abstract Argumentation Reasoner. In: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning. pp. 800–804 (9 2020). https://doi.org/10.24963/kr.2020/82, https://doi.org/10.24963/kr.2020/82
22. Nofal, S., Atkinson, K., Dunne, P.E.: Looking-ahead in backtracking algorithms for abstract argumentation. International Journal of Approximate Reasoning **78**, 265–282 (2016)
23. Thimm, M.: Dredd - a heuristics-guided backtracking solver with information propagation for abstract argumentation. In: The Third International Competition on Computational Models of Argumentation (ICCMA'19) (May 2019)
24. Thimm, M., Cerutti, F., Vallati, M.: Fudge: A light-weight solver for abstract argumentation based on sat reductions. arXiv preprint arXiv:2109.03106 (2021)

25. Vallati, M., Cerutti, F., Giacomin, M.: Predictive models and abstract argumentation: the case of high-complexity semantics. The Knowledge Engineering Review **34** (2019)
26. Wallner, J.P., Weissenbacher, G., Woltran, S.: Advanced sat techniques for abstract argumentation. In: Computational Logic in Multi-Agent Systems: 14th International Workshop, CLIMA XIV, Corunna, Spain, September 16-18, 2013. Proceedings 14. pp. 138–154. Springer (2013)