FernUniversität in Hagen
Faculty of Mathematics and Computer Science

Artificial Intelligence Group

Stochastic Local Search Algorithms for Abstract Argumentation with Heuristics based on Machine Learning

Bachelor's Thesis

in partial fulfillment of the requirements for
the degree of Bachelor of Science (B.Sc.)
in Informatik

submitted by
Konrad Drees

First examiner:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Advisor:	Prof. Dr. Matthias Thimm Artificial Intelligence Group

Statement

Ich erkläre, dass ich die Bachelorarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Bachelorarbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Bachelorarbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Yes	No
I agree to have this thesis published in the library.	☒	☐
I agree to have this thesis published on the webpage of the artificial intelligence group.	☒	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐
The source code is available under a GNU General Public License (GPLv3).	☒	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐

Passau, 22.8.2023 K. Drees

(Place, Date) (Signature)

Zusammenfassung

Abstrakte Argumentationsframeworks sind eine formale Darstellung von Argumentationssystemen und können zur Analyse und Bewertung der Akzeptanz von Argumenten verwendet werden. Allerdings kann der Prozess der Bestimmung von Mengen akzeptabler Argumente sehr rechenintensiv sein. Stochastische lokale Suchalgorithmen haben sich als wirksam erwiesen, um Optimierungsprobleme zu lösen und eine einzelne stabile Erweiterungen in einem abstrakten Argumentationsframework zu finden.

Das Hauptziel dieser Arbeit besteht darin, eine Erweiterung eines stochastischen Suchalgorithmus zu implementieren, wobei der Schwerpunkt auf der Entwicklung von Heuristiken basiert auf Machune Learning liegt, um die Leistung des Algorithmus zu verbessern.

Abstract

Abstract argumentation frameworks are a formal representation of argument systems and can be used for analyzing and evaluating the acceptability of arguments. However, the process of determining sets of acceptable arguments can be computationally expensive. Stochastic local search algorithms effectively solve optimization problems and find stable extensions in an abstract argumentation framework.

The main objective of this thesis is to implement an extension to a stochastic search algorithm with a focus on developing heuristics based on machine learning techniques to improve the algorithm's performance.

Contents

List of Figures

List of Tables

1 Introduction

Abstract argumentation frameworks (AAFs) formalize argument systems and evaluate their acceptability. They play a crucial role in understanding argumentative interactions. However, identifying acceptable arguments or stable extensions is computationally complex due to complicated interconnections between the arguments.

Therefore computing an argument's acceptability requires an entire argumentation framework, making the task computationally intensive. For instance, determining if an argument is part of a stable extension is an **NP-hard** problem [DD17].

Stochastic local search (SLS) algorithms are a promising solution for optimization challenges. They explore the solution space in a probabilistic manner and adapt to local optima, making them effective for identifying stable extensions in AAFs, as demonstrated by **WalkAAF** solver [Thi18].

With the help of Machine learning, we can create heuristics that help the Stochastic Local Search predict and guide the algorithm's steps, potentially resulting in improved exploration efficiency.

This thesis aims to enhance an SLS algorithm by integrating machine learning-based heuristics, inspired by recent successes of machine learning in complex decision-making and optimization tasks, to ultimately improve the algorithm's performance.

2 Theoretical Background

2.1 Abstract Argumentation

The formalism of *abstract argumentation frameworks* (AF) was introduced by Dung's theory of abstract argumentation [Dun95], and Dung distilled argumentations into its most fundamental elements: arguments and the relationships between them.

Thus the abstract approach is not concerned with the arguments' internal structure and focuses only on their interactions. An argumentation framework is a directed graph with nodes representing arguments and directed edges representing attacks between arguments. If there is an edge from argument A to argument B, it means that argument A attacks argument B.

Their abstract nature and theoretical cleanness are among the strengths following from the simplicity of the formalism [CG09]. Argumentation Frameworks can be applied to many different domains like knowledge representation and reasoning.

2.1.1 Basic Concepts

An abstract argumentation framework [Dun95] is a pair $AF = (AR, R)$ where AR is a set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that argument a attacks argument b, which can also be denoted as $a \to b$. For $a, b \in AR$ define $a^- = \{b \mid b \to a\}$ and $a^+ = \{b \mid a \to b\}$.

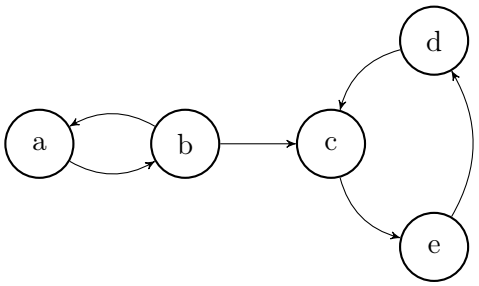

Figure 1: An abstract argumentation framework.

An abstract argumentation framework can be visualized as a directed graph in which arguments are represented as nodes and attacks are represented as arrows. Figure 1 shows an abstract argumentation framework AF with the arguments $AR = \{a, b, c, d, e\}$ and the attacks $R = \{(a, b), (b, a), (b, c), (c, e), (e, d), (d, c)\}$

Semantics for abstract argumentation are methods for determining the acceptability of arguments based on the relationships between them. Dung [Dun95] introduced four semantics, namely preferred, stable, complete, and grounded semantics, for abstract argumentation frameworks.

An argument $a \in AR$ is considered *acceptable* with respect to a set S of arguments, if for any argument $b \in AR$, if $b \to a$, then b is also attacked by the set S of arguments.

A set S of arguments is considered *conflict-free* if there are no arguments a and b in S such that $a \rightarrow b$.

A set of arguments S is *admissible* if it is *conflict-free* and each argument in S is *acceptable* with respect to S.

An argument a is considered *defeated* if it is attacked by an *acceptable* argument b.

Two main approaches to the definition of argumentation semantics are available in the literature: the labelling-based approach and the extension-based approach [BCG18].

In this thesis, the labelling-based approach is used.

2.1.2 Argument Labellings

A labelling L is a function $L : A \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ that assigns each argument $a \in A$ either the value `in`, meaning the argument is accepted, `out`, meaning the argument is rejected, or `undec` meaning the status of the argument is undecided [CG09].

Let $in(L)$ for $\{a \mid L(a) = \texttt{in}\}$, $out(L)$ for $\{a \mid L(a) = \texttt{out}\}$ and $undec(L)$ for $\{a \mid L(a) = \texttt{undec}\}$.

The labelling approach semantics can be used to define different types of semantics, including grounded, preferred, stable, and complete semantics.

A labelling is called *conflict-free* if there is for no $a, b \in in(L), a \rightarrow b$.

A labelling is called *admissible* if for all arguments $a \in AR$ it holds that [CG09]:

1. if a is labelled `in` then all attackers of a are labelled `out`
2. if a is labelled `out` then a has an attacker that is labelled `in`

An example for an *admissible* labelling for Figure 1 would be $\{(a, \texttt{out}), (b, \texttt{in}), (c, \texttt{out}), (d, \texttt{out}), (e, \texttt{in})\}$ or $\{(a, \texttt{in}), (b, \texttt{out}), (c, \texttt{in}), (d, \texttt{in}), (e, \texttt{out})\}$.

A labelling is called *complete* if it is *admissible* and it additionally for all arguments $a \in AR$ holds that [CG09]:

1. if a is labelled `undec` then it has at least one attacker that is labelled `undec` and it does not have an attacker that is labelled `in`.

Different types of classical semantics can be phrased by imposing further constraints. Thus the complete labelling L is [CG09]:

- *grounded* if and only if $in(L)$ is minimal
- *prefered* if and only if $in(L)$ is maximal
- *stable* if and only if $undec(L) = \emptyset$

The minimality and maximality of the statements are meant to be with respect to set inclusion [CG09].

2.1.3 Computational Problems in Abstract Argumentation Frameworks

Abstract Argumentation Frameworks are useful in representing complex argumentative structures and interactions. However, computational challenges arise when identifying acceptable arguments or determining extensions within the framework. Most algorithmic approaches for solving these problems are sound and complete methods [KWT22].

Computing an argument's acceptability requires considering an entire argumentation framework, making the task computationally intensive. For instance, determining if an argument is part of a preferred extension is an **NP-hard** problem [DD17].

An **NP-hard** (*Non-deterministic Polynomial-time hard*) problem is a classification of problems in computational complexity theory [DD17]. This class consists of problems for which no efficient (for example in polynomial time) solution has been found.

Another task is the identification of all or some extensions within the Argumentation Framework meeting their specific criteria. Each extension is a conflict-free subset of arguments [BCG18]. Calculating all the complete, stable, or preferred extensions in large argumentation frameworks can become computationally challenging due to combining all the subsets.

An overview of computational problems and their time complexity of different problems related to argumentation frameworks can be found in the literature [KPW17] and [DD17].

2.2 Stochastic local search algorithm

Stochastic Local Search (SLS) algorithms are used for solving computationally hard decision and optimization problems and are applied in various areas of computer science, including machine learning, optimization, and artificial intelligence [HS15].

Combinatorial problems can be categorized into two major groups: combinatorial decision problems and combinatorial optimization problems [KB21]. The two well-known combinatorial problems are the Propositional Satisfiability Problem (SAT) and the Travelling Salesman Problem (TSP), one of the most extensively studied combinatorial optimization problems [HS15].

Stochastic Local Search algorithms utilize randomness while exploring the solution space. These algorithms start with an initial solution and then, in each subsequent step, select a neighboring solution for comparison and replace the current solution with a better one. The neighboring solution is generally a slightly modified version of the current solution.

2.2.1 Stochastic local search algorithm in argumentation frameworks

Stochastic Local Search algorithms can be used to evaluate arguments in abstract argument frameworks and determine the acceptability of arguments based on their relationships with other arguments.

For this thesis, the base algorithm WalkAAF [Thi18] is used, which is a direct implementation of the GSAT[SLM92] and the WalkSAT [SKC93] idea.

WalkAAF starts with a labelling of an argument framework; in each iteration, a label of some argument is modified. The aim is to reach stable labelling and to avoid mislabeled arguments. An argument $a \in A$ is mislabeled if [Thi18]:

- $L(a) = \texttt{undec}$
- $L(a) = \texttt{out}$ and there exists no $b \to a$ with $L(b) = \texttt{in}$
- $L(a) = \texttt{in}$ and:
  - there is $b \to a$ with $L(b) \neq \texttt{out}$ or
  - there is $a \to b$ with $L(b) \neq \texttt{out}$

WalkAAF is implemented in Algorithm 1 and expects two input parameters N and M with $N, M \in \mathbb{N}$. The parameter N gives the maximal number of restarts before the algorithm terminates with a failure (FAIL). The number of iterations is given by the parameter M. Each run starts with a random labelling L in which each argument a is `in` or `out` (line 2). If the labelling L is stable, then the algorithm returns the stable labelling L. If the labelling L is mislabeled the algorithm selects one of those mislabeled arguments at random (line 7) and changes its labelling (lines 8−11). Thus an argument labelled e.g. `in` will be labelled as `out`. This labelling process will be repeated M times and if there is no stable labelling found, the algorithm restarts and tries for N times before terminating with FAIL. Thus the algorithm can fail even if there exists stable labelling.

Algorithm 1 WalkAAF$_{N,M}$ algorithm ($N, M \in \mathbb{N}$)

Input: $AF = (AR, R)$ AAF
Output: L a stable labelling (or FAIL if the search failed)

```
1:  for i = 1 to N do
2:    L ← randomise in and out
3:    for j = 1 to M do
4:      if L is stable then
5:        return L
6:      else
7:        Pick random mislabeled argument a
8:        if L(a) = in then
9:          L(a) ← out
10:       else
11:         L(a) ← in
12: return FAIL
```

Considering the argumentation framework from figure 1 as an input for the algorithm and the randomized labelling from line 2 as:

$$L_1(a) = \texttt{in} \qquad L_1(b) = \texttt{out} \qquad L_1(c) = \texttt{in} \qquad L_1(d) = \texttt{in} \qquad L_1(e) = \texttt{out}$$

The labelling L_1 is mislabelled because $L(c) = \texttt{in}$ and $L(d) = \texttt{in}$ with $d \rightarrow c$. Therefore the algorithm randomly picks a mislabelled argument, e.g. c, and relabelles the argument. The new labelling L_2 is defined as:

$$L_2(a) = \texttt{in} \qquad L_2(b) = \texttt{out} \qquad L_2(c) = \texttt{out} \qquad L_2(d) = \texttt{in} \qquad L_2(e) = \texttt{out}$$

Because $L_2(e) = \texttt{out}$ and the argument labelling $L(c) = \texttt{in}$ with $c \rightarrow e$, the labelling L_2 is mislabelled. Argument e will be relabelled and the new labelling L_3 is defined as:

$$L_3(a) = \texttt{in} \qquad L_3(b) = \texttt{out} \qquad L_3(c) = \texttt{out} \qquad L_3(d) = \texttt{in} \qquad L_3(e) = \texttt{in}$$

The labelling L_3 is mislabelled because of the arguments d and e and the algorithm continues and ends after M iterations or when a stable labelling L_{st} is found. The stable labelling of the argumentation framework from figure 1 is:

$$L_3(a) = \texttt{out} \qquad L_3(b) = \texttt{in} \qquad L_3(c) = \texttt{out} \qquad L_3(d) = \texttt{out} \qquad L_3(e) = \texttt{in}$$

2.3 Machine Learning

Machine learning is a subfield of artificial intelligence that uses algorithms that improve with experience or learn the rules without explicitly being programmed [CST21].

Artificial Intelligence refers to various techniques that allow computers to imitate human behavior and perform complex tasks independently or with minimal human involvement. It involves the ability to replicate or even surpass human decision-making processes [RN21].

Deep learning is a subfield of machine learning that uses concepts based on artificial neural networks [CST21].

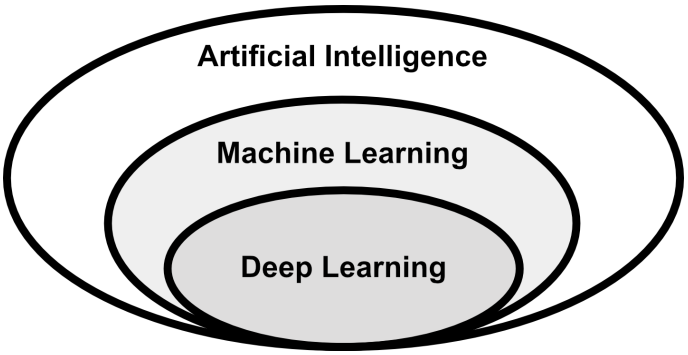

Figure 2: Relationship Between Artificial Intelligence, Machine Learning, and Deep Learning [RN21].

Figure 2 illustrates the relationships between the three domains: artificial intelligence, machine learning, and deep learning. Machine learning includes deep learning, and both are part of the wider field of artificial intelligence.

2.3.1 Fundamentals of Machine Learning Model Building

With the help of machine learning, machines learn with data to detect patterns.

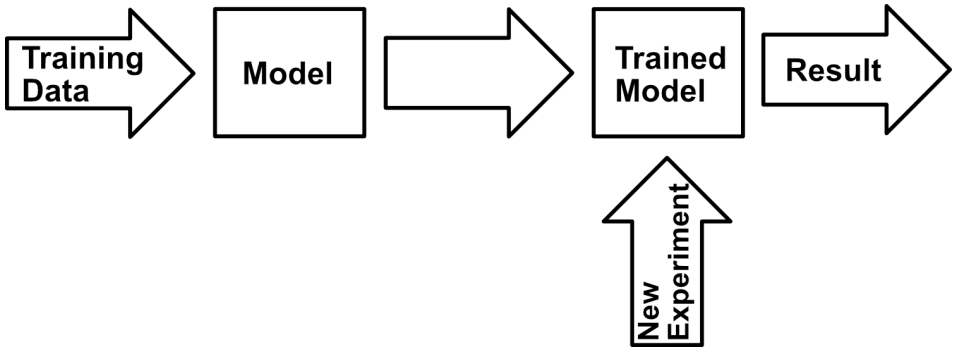

Figure 3: Machine Learning System [RN21].

Figure 3 shows the high-level flow for a Machine Learning System [RN21]. At first, a model is created, which is trained with training data. After that, we can use the trained model and use it on new Data for prediction.

Training involves providing the model with data to enable it to learn rules and enhance its performance. The data structure employed will vary based on the machine learning type and the model selected. [RN21].

Extracting features is an important step in automating the identification of patterns and relationships from large data. These features are used for model building and are derived from the raw data input. Essentially, a feature describes a property that provides a suitable representation [JZH21].

When constructing a model, the input data is used by a learning algorithm to identify patterns and relationships that are relevant to the respective learning task. Machine Learning requires features for this task. On this basis, each learning algorithm family applies different model-building mechanisms. For example, when building a classification model, decision tree algorithms exploit the feature space by incrementally splitting data records into increasingly homogenous partitions following a hierarchical, tree-like structure [JZH21].

2.3.2 Machine Learning Approaches

When it comes to machine learning, there are several known categories of learning problems. These include supervised, unsupervised, semi-supervised, and reinforcement learning.

- **Supervised Learning**
  These algorithms need external assistance for training a model to predict for example, the acceptability of arguments based on a set of labeled examples [Lau22]. Some of the most famous supervised machine learning algorithms include Decision Tree, Naive Bayes and Support Vector Machine [Lau22].

- **Reinforcement Learning**
  This type of learning is based on training a model to learn from its past experiences. The model receives rewards for making good actions and penalties for making bad actions. Therefore the algorithm depends on trial and error and delayed outcome [Lau22].

- **Unsupervised Learning**
  These algorithms learn a few features from the data. When new data is introduced, it uses the previously learned features to recognize the class of the data. These algorithms are mainly used for clustering and feature reduction [Lau22].

- **Semi-Supervised Learning**
  It is a method that deals with similar issues as supervised learning while incorporating techniques from unsupervised learning. It involves providing the machine with labeled data as well as additional unlabeled data [RN21].

2.3.3 Machine Learning Algorithms

In this thesis, we study the use of these machine learning techniques in a heuristic with a stochastic local search algorithm in an abstract argumentation framework, and we will utilize several types of machine learning algorithms for classifications, including the following:

- **Neural Networks (NN)**
  An artificial neural network generally consists of multiple artificial neurons that are connected with each other. It works on three layers, with the input layer taking input, the hidden layer processing the input, and finally, the output layer sending the calculated output [Lau22].

- **Graph Convolutional Networks (GCN)**
  GCN is a scalable approach for semi-supervised learning on graph-structured data based on an efficient variant of convolutional neural networks operating directly on graphs.

- **GraphSAGE (Graph Sample and Aggregated)**
  GraphSAGE is a graph neural network (GNN) model developed to generate embeddings for nodes in graph data, allowing for both inductive learning and transductive learning. It can be used to generate embeddings for previously unseen data, making it an inductive model. This is achieved by sampling and aggregating features from a node's local neighborhood [HYL18].

- **k-Nearest Neighbors (KNN)**
  The KNN algorithm can be used for classification and regression tasks. It works by finding the k nearest neighbors of a new data point in the training set and using their labels or values to predict the label or value of the new data point. Thus the k-NN classification works in two stages the first is the determination of the nearest neighbors, and the second is the determination of the class using those neighbors [CD21].

- **Naive Bayes (NB)**
  Naive Bayes is a learning algorithm that utilizes Bayes rule together with a strong assumption that the attributes are conditionally independent, given the class [Web10].

- **Decision Tree (DT)**
  Decision tree is a technique for classification and regression tasks. They work by partitioning the data based on attributes and making decisions based on the resulting partitions [Für10].

- **Random Forest (RF)**
  Random forests are a learning technique that combines decision trees to improve the accuracy and robustness of the predictions. They randomly sample the features and data points and train multiple decision trees on the resulting subsets [Bre01].

- **GradientBoosting (GB)**
  Gradient Boosting is used for both regression and classification problems, which works by combining multiple weak predictive models, typically decision trees, to create a strong predictive model. It is an ensemble learning method that builds new models to correct the errors made by existing models [Fri01]. One implementation is e.g. XGBoost.

2.3.4 Performance metrics of Machine Learning

When evaluating and selecting machine learning models, various metrics are chosen. These metrics can also aid in tuning hyperparameters and improving the model. Metrics play an integral role in assessing the performance of machine learning models.

Table 1: Confusion Matrix

	Predicted: Yes	**Predicted: No**
Actual: Yes	True Positive (TP)	False Negative (FN)
Actual: No	False Positive (FP)	True Negative (TN)

When comparing classification algorithms, a confusion matrix is used to summarize their performance. Table 1 shows a confusion Matrix, a 2×2 table containing four values: true positives, false positives, true negatives, and false negatives.

The concept of true positives, false positives, true negatives, and false negatives is explained below:

- **True Positives (TP)**
  These are cases in which the model predicted 'Yes', and the actual class is also 'Yes'.
- **True Negatives (TN)**
  These are cases in which the model predicted 'No', and the actual class is also 'No'.
- **False Positives (FP)**
  These are cases in which the model predicted 'Yes', but the actual class is 'No'.
- **False Negatives (FN)**
  These are cases in which the model predicted 'No', but the actual class is 'Yes'.

We validate results by using performance measures such as Accuracy, ROC AUC, Recall, F1-Score, and precision. These measures are calculated from the confusion matrix and provide insight into a model's performance. Below are short descriptions of each performance measure.

Accuracy is defined as the percentage of correctly classified instances by the classifier [EK21]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Recall is the proportion of those instances that are actually positive and are predicted as positive [GSKJ20]:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Precision is defined as positively predicted instances that are actually positive. It is also denoted as a positive predicted value [GSKJ20]:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

The F1-Score is a harmonic mean of precision and recall [EK21] .

$$F1\text{-}Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

Receiver Operating Characteristic Area Under the curve(ROC AUC) is calculated by the area under the ROC curve, which is plotted with TPR (True Positive Rate) and FPR (False Positive Rate) [GSKJ20]:

$$True\ Positive\ Rate = \frac{TP}{TP + FN} \tag{5}$$

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \tag{6}$$

The choice of the most suitable model depends on the specific use case and comes with the cost of different types of errors. For instance, in certain scenarios, a model with a higher recall might be preferred over one with better accuracy. Therefore, the model selection should carefully balance these performance metrics in alignment with the specific requirements of the use case.

3 State of Research

3.1 Current Solvers for Abstract Argumentation Frameworks

In the last two decades, the study of argumentation has gained importance in Artificial Intelligence research [BCD07]. One of the most notable methods used is abstract argumentation frameworks, developed by Dung in his paper 'Acceptability as an Abstract Argumentation Framework' [Dun95]. This formalism is straightforward and effective in representing and evaluating arguments for their acceptability. A significant challenge in working with abstract argumentation frameworks is the computational complexity of specific problems, such as determining the acceptability status of arguments and finding their extensions [KPW17].

As stated in source [CDG$^+$15], two main types of algorithms are used for solving reasoning problems in abstract argumentation: reduction-based and direct approaches. **Dung-O-Matic**[1], **ArgSemSAT** [CGV19], and **ConArg2**[BS13] are some solvers that have been used to compute various semantics. These solvers often use standard techniques like translation to answer set programming or by applying SAT reductions. For example, the winner of the preferred semantics track at the 2017 International Competition on Computational Models of Arguments (ICCMA 2017) is **ArgSemSAT**. This biennial contest focuses on problems associated with abstract argumentation frameworks. **ArgSemSAT** uses multiple calls to an SAT solver to compute complete labellings, as stated by Cerutti in their paper [CGV19].

Another technique uses Stochastic Local Search algorithms, such as **WalkSat** [SLM92] and **GSAT** [SKC93], which have shown promising results. A significant contribution in this context is the **WalkAAF** [Thi18] solver, which is based on the WalkSAT algorithm. The **WalkAAF** solver incorporates stochastic local search into argumentation frameworks and demonstrates a strong performance on various semantics.

Despite the successes of **WalkAAF** and similar solvers, a key challenge lies in determining the next move during the local search. **WalkAAF** added a greedy move using flipping count in abstract argumentation frameworks. It is here that heuristics play a crucial role. However, developing efficient and robust heuristics remains complex and often are problem-specific.

Machine learning has seen significant advancements in techniques that can potentially be used to learn these heuristics from data [KWT22].

Although techniques like reinforcement learning, deep learning, and graph neural networks have shown success in various domains, their effectiveness in argumentation frameworks, particularly in learning heuristics for SLS-based solvers, remains to be explored.

[1] `http://www.arg.dundee.ac.uk/?page_id=279`

3.2 Motivation

In this thesis, we combine stochastic local search algorithms with machine learning.

Stochastic Local Search (SLS) algorithms have shown great promise in finding a single stable extension, as demonstrated by **WalkAAF** solver. SLS algorithms have the advantage of exploring the search space and achieving solutions without requiring exhaustive searching.

With the help of Machine learning, we create heuristics that help the SLS predict and guide the algorithm's steps, potentially resulting in improved exploration efficiency.

Therefore, this thesis aims to integrate machine learning techniques into the SLS algorithm in the context of AFs. Using the predictive ability of machine learning, the aim is to guide the SLS process toward an efficient exploration of solution space.

4 Methodology and Implementation

To assist the SLS Algorithm with Machine Learning-based Heuristics, we must construct two Models: one for initial Labelling and one for selecting subsequent Moves. To develop these Models, we require a substantial amount of Data, which can be obtained by generating Argumentation Frameworks and creating many different scenarios.

In the following, we will cover Graph Generating and getting the Data, Building Machine Learning Models, and the implementation of the SLS Algorithm with heuristics based on machine learning.

4.1 Graph Generating

To have many different Graphs with different properties, we use random graphs based on the Barabási-Albert-, Watts-Strogatz-, and Erdős-Rényi-model generated using *AFBenchGen2*[2]. Also, graphs generated by the KWT-Gen of the *tweetyproject*[3] were used.

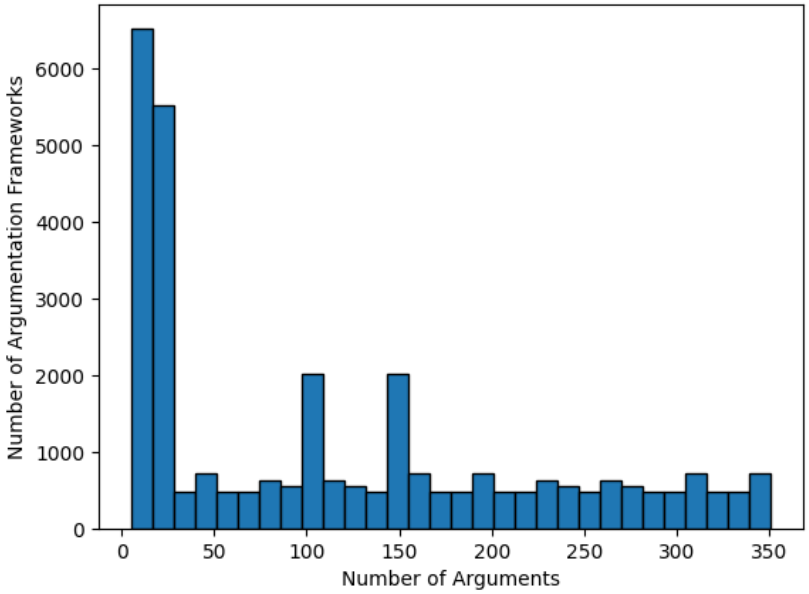

Figure 4: Distribution of Number of Arguments across all Argumentation Frameworks.

To begin, 15000 Argumentation Frameworks are generated using *AFBenchGen2*.

[2] `https://sourceforge.net/projects/afbenchgen/`
[3] `http://tweetyproject.org/r/?r=kwt_gen`

The number of arguments ranges from 5 to 350, increasing by 5 for each step.

A total of 12000 Argumentation Frameworks were generated, each with a size between 5 and 25. The number of arguments in each Framework increased by 1.

Using KWT-Gen from the *tweetyproject*, 1500 Argumentation Frameworks consisting of 100 arguments were created, and another 1500 Argumentation Frameworks consisting of 150 arguments.

Our data consists of over 30000 Argumentation Frameworks, with over 3500000 individual Arguments.

In Figure 4, we can observe the distribution of the number of arguments present in all Argumentation Frameworks. The dominant distribution of classes lies between 5 to 25 Arguments. The Argumentation Frameworks created by KWT-Gen from the *tweetyproject* are also visible in the graph.

4.2 Feature Extractions

For the Machine Learning Model to be effective, it requires relevant features that accurately represent the characteristics of the data and the argumentation framework.

Therefore to train and use machine learning models, it is necessary to extract the following features from all the generated Argumentation Frameworks:

- **Predecessors:** the number of attacks to the input argument
- **Successors:** the number of attacks from the input argument
- **Predecessors Neighbors:** the total number of the attacks by the arguments attacking the input argument
- **Successors Neighbors:** the total number of the attacks by the arguments being attacked by the input argument
- **Degree Centrality:** the fraction of arguments the argument is connected to via attacks and getting attacked
- **In-Degree Centrality:** the fraction of arguments its incoming attacks are connected to.
- **Out-Degree Centrality:** the fraction of arguments its outgoing attacks are connected to.
- **Closeness Centrality:** measuring the shortness of the path of the input argument to the other arguments.
- **Betweenness Centrality:** measuring the extent to which an argument lies on paths between other arguments.
- **Average Neighbor Degree:** The average degree of the neighbors of an argument.

- **PageRank:** ranking the input argument based on the structure of the attacking arguments
- **Edge Node Ratio:** The ratio of the total number of attacks to the total number of arguments.

As computing all features of the arguments can be very time-consuming, the most relevant features are used through feature selection depending on the machine learning model. Therefore, we must balance the accuracy of the Machine Learning Model with the time it takes to calculate the features.

4.3 Machine Learning Models

We require two Machine Learning Models that will assist the SLS Algorithm. As per algorithm 1, we will need $Model_{in}$ to label the arguments initially in line 2 and $Model_{rn}$ to determine the next switch in the labelling in line 7.

Therefore we need to cast the initial labelling and choosing the next switching into two different classification problems.

In the first classification problem for the initial labelling, some adjustments to the WalkAAF algorithm were made. It now returns the labelling with the fewest mislabeled arguments or, if found, the stable labelling. We applied this algorithm to all the Argumentation Frameworks we generated and used the best labeling.

As a result, we ended up creating two different classes.

- **Class 0:** argument with the best labelling `'out'`.
- **Class 1:** argument with the best labelling `'in'`.

The distribution for those two classes is shown in the following Table 2:

Table 2: Distibution of the Classes for the $Model_{in}$

Total Number of **Class 0**	Total Number of **Class 1**	Sum both **Classes**
2384076	1155502	3539578

Thus we have a Class Balance Ratio of 2.06 with **Class 0** double the total amount of **Class 1**. The sum of the two classes is the amount of all arguments across all abstract argumentation frameworks.

The Class Balance Ratio is determined by dividing the total number of **Class 0** by the total number of **Class 1**.

With the label and features, we can train $Model_{in}$ using the Machine Learning Techniques referenced in section 2.3.3.

For the second classification problem for selecting the next mislabelled argument for switching, various scenarios were created for each argumentation framework.

Initially, one random labelling of the argumentation framework was chosen. Then, all mislabelled arguments were selected. Each mislabelled argument's label was switched one by one, and the difference in the number of mislabelled arguments before and after the switching was calculated.

Also, the current labelling before switching of the argument was saved, creating a dictionary with the difference of the mislabelled for the `'out'` labelling and `'in'` labelling.

For example, if the current labelling is `'in'` with total of 6 mislabelled arguments and after switching it to `'out'` with the new total of 3 mislabelled arguments, we saved the difference 3 to the label `'in'`. Therefore switching the arguments labelling from `'in'` to `'out'` results to 3 less mislabelled arguments.

After that, the amount of positive impacts (leading to fewer mislabelled arguments) and the amount of negative impacts (leading to more mislabelled arguments) for each argument is calculated.

When the number of positive impacts is bigger than the negative impact ones, we have

As a result, we ended up creating two different classes:

- **Class 0:** When the number of positive impacts is smaller than the negative impact ones.
- **Class 1:** When the number of positive impacts is bigger than the negative impact ones.

The distribution for those two classes is shown in the following Table 3:

Table 3: Distibution of the Classes for the $Model_{rn}$

Total Number of **Class 0**	Total Number of **Class 1**	Sum both **Classes**
2694916	4384240	7079156

Thus we have a Class Balance Ratio of 0.61 with **Class 1** double the total amount of **Class 0**. The sum of the two classes is double that of all arguments across all abstract argumentation frameworks because we used the labelling `'in'` and `'out'` as a feature for Machine Learning.

With the label and features, we can train $Model_{rn}$ using the Machine Learning Techniques referenced in section 2.3.3.

For the training of $Model_{rn}$ and $Model_{in}$, 80% of the classes and features as training data were used and 20% were used as test data.

The Implementation of the Neural Network is based on [KKT22]. We used Graph Convolutional Networks (GCN) and GraphSage. Each model features one pre-message

passing layer (256- dim Multi-Layer Percepton), three message passing layers (determined by the respective GNN model), and two post-message passing layers (256-dim Multi-Layer Percepton).

4.4 Optimizations

Hyperparameters are parameters that are not learned directly within the estimators. In *scikit-learn*, the constructor of the estimators accepts them as parameters, allowing those for optimization.

For the **Random Forest Classifier** and **XGBoost Classifier** an **Exhaustive Grid Search** is used for hyperparameter optimization.

Exhaustive search over specified parameter values for an estimator. The implementation GridSearchCV [4] uses a 'fit' and a 'score' method. GridSearchCV will run the model using each combination of hyperparameters in the grid and select a scoring metric by which the model's performance will be evaluated. The ROC AUC Score is used for the model performance evaluation.

For the **Random Forest Classifier** these parameters will be optimized:

- **Number of estimators:** The Number of Trees in the Forest.
- **Maximum Depth:** The maximum depth of the tree.
- **Minimum Samples Leaf:** The minimum number of samples required at a leaf node.
- **Minimum Samples Split:** The minimum number of samples required to split an internal node.

For the **XGBoost Classifierr** these parameters will be optimized:

- **Number of Estimators:** Number of gradient boosted trees.
- **Learning Rate:** Boosting learning rate.
- **Minimum child weight:** Minimum sum of instance weight needed in a child.
- **Maximum Depth:** The maximum depth of the tree.

Hyperparameter optimization can significantly improve performance. But it is also worthwhile to consider the computational trade-offs because it can be computationally expensive.

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV`

4.5 Algorithms Implementation

The different Machine Learning Models are evaluated using the metrics and the best ones are selected for the WalkAAF Algorithm. In total three new SLS algorithms based on WalkAAF are implemented. The first one uses Machine Learning to select the next mislabelled argument, while the second one also includes the initial labelling. The third one only uses machine learning model for the initial labelling. Also, a new Parameter P is added for controlling how often the machine learning heuristic will be used. The first one is shown in Algorithm 2, which uses the $Model_{rn}$ on line 9 for selecting the best argument.

Algorithm 2 WalkAAF$^{ML}_{N,M}$ algorithm ($N, M \in \mathbb{N}, P \in [0, 1]$)

Input: $AF = (AR, R)$ AAF
Output: L a stable labelling (or FAIL if the search failed)

```
1: extract features of all the arguments
2: for i = 1 to N do
3:   L ← randomise in and out
4:   for j = 1 to M do
5:     if L is stable then
6:       return L
7:     else
8:       if random < P then
9:         predict the best argument for label flipping
10:      else
11:        Pick random mislabeled argument a
12:        if L(a) = in then
13:          L(a) ← out
14:        else
15:          L(a) ← in
16: return FAIL
```

The second algorithm is shown in algorithm 4, which uses the $Model_{rn}$ on line 11 for selecting the best argument and $Model_{in}$ for the best initial labelling. The idea is that it predict the initial labelling one time and if the first times fails did not lead to a stable labelling, the random labelling will be used.

Algorithm 3 WalkAAF$^{ML}_{N,M}$ algorithm ($N, M \in \mathbb{N}, P \in [0,1]$)

Input: $AF = (AR, R)$ AAF
Output: L a stable labelling (or FAIL if the search failed)

```
extract features of all the arguments
L ← predict the best initial labelling
for i = 1 to N do
  if predicted initial labelling did not lead to a stable labelling then
    L ← randomise in and out
  for j = 1 to M do
    if L is stable then
      return L
    else
      if random < P then
        predict the best argument for label flipping
      else
        Pick random mislabeled argument a
        if L(a) = in then
          L(a) ← out
        else
          L(a) ← in
return FAIL
```

The third algorithm is shown in algorithm 4, which uses the $Model_{in}$ for the best initial labelling.

Algorithm 4 WalkAAF$^{ML}_{N,M}$ algorithm ($N, M \in \mathbb{N}, P \in [0,1]$)

Input: $AF = (AR, R)$ AAF
Output: L a stable labelling (or FAIL if the search failed)

```
1: extract features of all the arguments
2: L ← predict the best initial labelling
3: for i = 1 to N do
4:    if predicted initial labelling did not lead to a stable labelling then
5:       L ← randomise in and out
6:    for j = 1 to M do
7:       if L is stable then
8:          return L
9:       else
10:         Pick random mislabeled argument a
11:         if L(a) = in then
12:            L(a) ← out
13:         else
14:            L(a) ← in
15: return FAIL
```

For the three algorithms, the features of all the arguments of the abstract argumentation framework are calculated on line 1.

4.6 Experimental Settings

Our experiment aims to evaluate the effect of machine learning models on the SLS-algorithm.

We trained and evaluated the different machine learning techniques, k-Nearest Neighbor (KNN), Naive Bayes (NB), Random Forest (RF), Decision Tree (DT), GradientBoosting (GB) using scikit-learn3 [PVG+11], the machine learning framework for Python.

The two Neural Networks, Graph Convolutional Network (GCN) and GraphSAGE are trained and evaluated using the Pytorch library, a machine learning framework for Python. Each model is trained for 1000 epochs, and the metrics are calculated using the last epoch.

The SLS algorithms are implemented in Python and use the networkx library for calculating the features of the abstract argumentation frameworks.

We use the four best features identified in the feature selection as input for the classifiers.

For evaluating the usage of the machine learning models on the SLS-Algorithm. We run the three Algorithms on a Benchmark Set of Abstract Argumentation Frameworks from the International Competition on Computational Models of Argumentation (ICCMA) and compare their runtime and success rate.

5 Results and Discussions

5.1 Model Selections for the Initial Labelling of Arguments

After the training $Model_{in}$, the models were tested on 20% of the overall data. The classification results were calculated using the built-in Python functions of each classifier. Also, all the features were used for training.

Table 4: Classification results after training the $Model_{in}$.

Classifier	**Accuracy**	**ROC AUC**
NB	0.47	0.58
DT	0.78	0.75
RF	0.84	0.79
KNN	0.82	0.78
XGBoost	0.84	0.79
GradientBoost	0.84	0.78
GCN	0.86	0.83
GraphSAGE	0.84	0.72

The results of each classifier are shown in table 4. From the classical Machine Learning techniques, the Random Forest and XGBoost, and GradientBoost did have the best overall score, with Random Forest and XGBoost having an Accuracy of 84% and ROC AUC of 79%. GradientBoost has the same overall Accuracy, but the ROC AUC is a bit worse at 78%, so the difference is negligible.

The reason is that XGBoost and Gradientboost both use Decision Trees as base learners, just like Random Forest [CG16]. Also, XGBoost and GradientBoost use both Gradient Boosting[CG16].

The model based on Naive Bayes has the worst overall metrics with an Accuracy of 47% and a ROC AUC of 58%.

Looking at the neural networks Graph Convolutional Networks (GCN) and GraphSAGE, GCN does have the better Accuracy and ROC AUC Scores with an Accuracy of 97% and ROC AUC of 83%. The Graph Convolutional Network (GCN) has the best overall scores of all the classifiers. Generally, neural networks seem suited to labelling the arguments in an Abstract Argumentation Framework. Overall Graph Convolutional Network (GCN), Random Forest, and XGBoost seem to have the best results for the initial labelling problem and will be used for further optimizations and discussions.

Table 5 shows the Classification report of the $Model_{in}$ using the Random Forest classifier. The Precision for Class 0 is 0.84 meaning means that 84% of the instances predicted as Class 0 were actually Class 0 and the Precision for Class 1 is 0.94 meaning means that 94% of the instances predicted as Class 1 were actually Class 1. The model has a higher precision for Class 1 but a lower recall for Class 1. This suggests that

Table 5: Classification report for $Model_{in}$ using Random Forest.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.84	0.94	0.84
Recall	0.94	0.63	0.84
F1-Score	0.89	0.72	0.84

while the model is confident about the instances it labels as Class 1, it misses quite a few actual Class 1 instances, meaning the model performs well in identifying Class 0 instances but needs to work on Class 1. For Class 0, the model has a good balance between precision and recall, as evidenced by similar values and high F1-Score. It is also noted that the model has a reasonably high accuracy. Still, this metric needs to be more accurate given the class imbalance with a Class Balance Ratio of 2, meaning there is around double the amount of Class 0 than Class 1.

Table 6: Classification report for $Model_{in}$ using XGBoost.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.84	0.84	0.84
Recall	0.94	0.64	0.84
F1-Score	0.89	0.73	0.84

Table 6 shows the Classification report of the $Model_{in}$ using the XGBoost classifier. Both the XGBoost model and the Random Forest model have similar performances with slight variances. The XGBoost model shows marginally better results in terms of Class 1 recall. The minor performance boost might be due to XGBoost's regularization, handling of missing data, and the gradient boosting framework.

Table 7: Classification report for $Model_{in}$ using Graph Convolutional Network.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.91	0.93	0.92
Recall	0.92	0.89	0.91
F1-Score	0.92	0.91	0.91

The Graph Convolutional Network performs well in accuracy and classifying Class 0 instances. However, the performance in Class 1, specifically in terms of recall, indicates some room for improvement. Overall GCN has a better performance than XGBoost and Random Forest. It is also noted that GCN exploit graph structures to make predictions, and the structure of the argumentation frameworks influences their performance.

Since using and calculating all the features is computationally expensive, only four features will be selected for the next model training. The feature score was calculated using the built-in function of the classifier Random Forest, Decision Tree, and Gradient Boosting. Obtaining feature importance in GCNs can be challenging since GCN operate on graphs, where both the nodes and edges can have different features [KW17].

Table 8: Feature Importances after training the $Model_{in}$ across different classifiers.

Feature	**Random Forest**	**Gradient Boosting**	**Decision Tree**
Betweenness Centrality	0.15	0.07	0.09
Closeness Centrality	0.14	0.04	0.15
Predecessors	0.12	0.56	0.27
Pagerank	0.11	0.03	0.11
In Degree Centrality	0.10	0.02	0.04
Predecessors Neighbors	0.08	0.10	0.07
Degree Centrality	0.07	0.00	0.03
Average Neighbor Degree	0.06	0.02	0.04
Out Degree Centrality	0.05	0.01	0.03
Successors Neighbors	0.05	0.00	0.04
Edge Node Ratio	0.04	0.11	0.11
Successors	0.03	0.02	0.01

Table 8 shows the Importances of the used features for the Random Forest, Gradient Boosting and Decision Tree classifiers, provided by the built-in functions. The Gradient Boosting model heavily uses the amount of Predecessors feature, showing a preference compared to the Random Forest and Decision Tree models. Centrality measures, like Betweenness Centrality, are considered more important in Random Forest and Decision Tree. The different feature importances show that each classifier might have different relationships in the data, leading to different feature importances.

For the initial labelling of an Abstract Argumentation Framework, the classifier GCN, RF, and XGBoost seem best suited.

5.2 Model Selection for Switching the Arguments Labelling

After the training $Model_{rn}$, the different models were tested on 20% of the overall data. The classification results were also calculated using the built-in Python functions of each classifier. Also, all the features were used for training just like the $Model_{in}$.

The results of each classifier are shown in table 9. Among the conventional algorithms, the Random Forest outperforms others with an Accuracy of 90% and an ROC AUC of 87%. The Decision Tree is close with an accuracy of 85%. XGBoost performs the same with an accuracy of 89% and has the highest ROC AUC of 89% among the

Table 9: Classification results after training the $Model_{rn}$.

Classifier	**Accuracy**	**ROC AUC**
NB	0.61	0.68
DT	0.85	0.84
RF	0.90	0.87
KNN	0.88	0.87
XGBoost	0.89	0.89
GradientBoost	0.88	0.88
GCN	0.86	0.77
GraphSAGE	0.82	0.73

traditional models. The Naive Bayes classifier shows the least performance with an accuracy of 61%.

When we examine graph-based classifiers, the Graph Convolutional Network achieves an Accuracy of 86%. However, its ROC AUC of 77% indicates that it has some problems with class discrimination. GraphSAGE lags slightly behind GCN, achieving an accuracy of 0.82 and a ROC AUC of 73%.

The graph-based classifiers perform a bit worse than in the $Model_{in}$, indicating that it is more suitable for the initial labelling of the Arguments in an Abstract Argumentation Framework.

Overall Graph Convolutional Network, Random Forest, and XGBoost also seem to have the best results and will be used for further optimizations and discussions.

Table 10: Classification report for $Model_{rn}$ using Random Forest.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.87	0.91	0.89
Recall	0.85	0.92	0.90
F1-Score	0.86	0.92	0.89

Table 10 shows the Classification report for $Model_{rn}$ using Random Forest. For Class 0, the precision is 0.87, indicating that out of all the instances predicted as Class 0, 87% are correctly classified. Class 1, on the other hand, has a slightly higher precision of 0.91. The Random Forest model performs very good for both classes, with a slightly better performance in detecting Class 1 instances. The balance between precision and recall, as indicated by the F1-scores, suggests that the model is robust in its predictions for the given data.

Table 11 shows the Classification report for $Model_{rn}$ using XGBoost. With a recall of 0.85 for Class 0, XGBoost manages to recognize 85% of the total Class 0 samples correctly. For Class 1, the model shows a higher recall of 0.92, hinting that

Table 11: Classification report for $Model_{rn}$ using XGBoost.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.87	0.91	0.89
Recall	0.85	0.92	0.89
F1-Score	0.86	0.92	0.89

it performs better identifying Class 1 instances. XGBoost shows strong performance in identifying Class 1 instances. It maintains a consistent balance between precision and recall, as seen by the F1-score.

Table 12: Classification report for $Model_{rn}$ using Graph Convolutional Network.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.76	0.81	0.79
Recall	0.66	0.87	0.79
F1-Score	0.71	0.84	0.79

Table 11 shows the Classification report for $Model_{rn}$ using Graph Convolutional Network. Class 0 has a precision of 0.76, meaning that of all the instances predicted as Class 0. Class 1 exhibits a higher precision of 0.81, suggesting that the model has a slightly better performance for detecting instances of this class. The Graph Convolutional Network's performance is quite acceptable, especially its ability to detect Class 1 instances. The differences in precision and recall for the two classes hint at possible areas for model improvements.

Table 13 shows the Importance of the used features for the Random Forest, Gradient Boosting, and Decision Tree classifiers provided by the built-in functions. The actual labelling of the argument has a huge influence on all three models shown in the Feature 'State'. This underscores the expectation that the actual labelling would be a useful feature in the prediction.

Centrality measures are more important in Random Forest and Decision Tree, Gradient Boosting seems to see the feature less important. This shows that the classifiers interpret the interconnectedness and influence within the argumentation framework differently.

The actual labelling is the most dominant feature for selecting the next argument for labelling switching. A combination of topological, centrality and network metrics also contribute significantly to the features, just like in the features of $Model_{in}$.

5.3 Feature Selection and Optimisation

Since using all the features would be computationally expensive, the following features will be used for both models for each traditional and graph-based classifier. For the

Table 13: Feature Importances after training the $Model_{rn}$ across different classifiers.

Feature	**Random Forest**	**Gradient Boosting**	**Decision Tree**
State	0.29	0.38	0.22
Predecessors	0.09	0.20	0.07
Closeness Centrality	0.09	0.00	0.07
Predecessors Neighbors	0.08	0.06	0.08
Edge Node Ratio	0.08	0.17	0.22
Pagerank	0.07	0.00	0.07
Betweenness Centrality	0.07	0.00	0.07
Successors Neighbors	0.05	0.10	0.09
Average Neighbor Degree	0.04	0.01	0.03
In Degree Centrality	0.04	0.00	0.02
Out Degree Centrality	0.03	0.01	0.02
Degree Centrality	0.03	0.00	0.02
Successors	0.03	0.05	0.02

actual models, these features are used for $Model_{in}$ and $Model_{rn}$:

- **Betweenness Centrality**
- **Closeness Centrality**
- **Predecessors**
- **Pagerank**

$Model_{in}$ also uses Feature 'State', the actual labelling `'in'` or `'out'` of the argument.

When comparing table 8 with 13, there's a distinct difference in how features are prioritized. For instance, in $Model_{rn}$, the 'State' feature (actual labelling) holds considerable weight across models. In $Model_{in}$ the feature 'Predecessors' is important, especially in Gradient Boosting.

Some features like Closeness Centrality and Pagerank maintain consistent importance across the different classifiers, suggesting their general utility in capturing the characteristics of the argumentation framework. On the other hand, features like Betweenness Centrality shows model-dependent significance, indicating that their importance may be useful in specific machine learning technique.

The classification reports from the different classifiers are shown in Table 14 to Table **??**.

Overall the performance of all classifiers went down, because fewer features, thus information about the argument or the argumentation framework, is lost. Also, it is best not to overengineer Machine Learning Models; you have to find a good balance between performance and accuracy. Too many features about the argumentation

Table 14: Classification report for $Model_{rn}$ using XGBoost after feature reduction..

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.81	0.90	0.86
Recall	0.83	0.88	0.85
F1-Score	0.82	0.89	0.86

Table 15: Classification report for $Model_{in}$ using XGBoost after feature reduction.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.83	0.83	0.83
Recall	0.94	0.59	0.78
F1-Score	0.88	0.69	0.82

Table 16: Classification report for $Model_{rn}$ using Random Forest after feature reduction..

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.82	0.90	0.87
Recall	0.84	0.89	0.86
F1-Score	0.83	0.89	0.87

Table 17: Classification report for $Model_{in}$ using Random Forest after feature reduction.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.83	0.83	0.83
Recall	0.94	0.59	0.78
F1-Score	0.88	0.69	0.82

Table 18: Classification report for $Model_{rn}$ using GCN after feature reduction..

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.72	0.78	0.76
Recall	0.61	0.85	0.74
F1-Score	0.66	0.81	0.76

frameworks lead to longer feature extraction runtime and longer runtime of the machine learning model, but too few features lead to worsening accuracy but better runtime.

Table 19: Classification report for $Model_{in}$ using GCN after feature reduction.

Metric	**Class 0**	**Class 1**	**Weighted Avg.**
Precision	0.87	0.83	0.86
Recall	0.93	0.72	0.84
F1-Score	0.90	0.77	0.86

The Hyperparameter of the classifiers XGBoost and Random Forest are optimized using the GridSearchCV function from the sklearn library. Table 20 shows the hyperparameter of the Random Forest Classifier after optimization and Table 21 the hyperparameter of the XGBoost after optimization.

Table 20: Best Hyperparameter for the Random Forest Classifier

Model	**Max Depth**	**Min Leaf**	**Min Split**	**Num. Estimators**
$Model_{in}$	30	1	10	200
$Model_{rn}$	20	1	10	200

Table 21: Best Hyperparameter for the XGBoost Classifier

Model	**Learning Rate**	**Max Depth**	**Min Weight**	**Num. Estimators**
$Model_{in}$	0.1	7	1	500
$Model_{rn}$	0.1	7	5	500

For the SLS Algorithms, the Classifiers Random Forests and the Graph Convolutional Networks are chosen and will be used for the heuristics based on machine learning, because those have the overall best performances after feature reduction and optimization.

5.4 SLS Algorithm

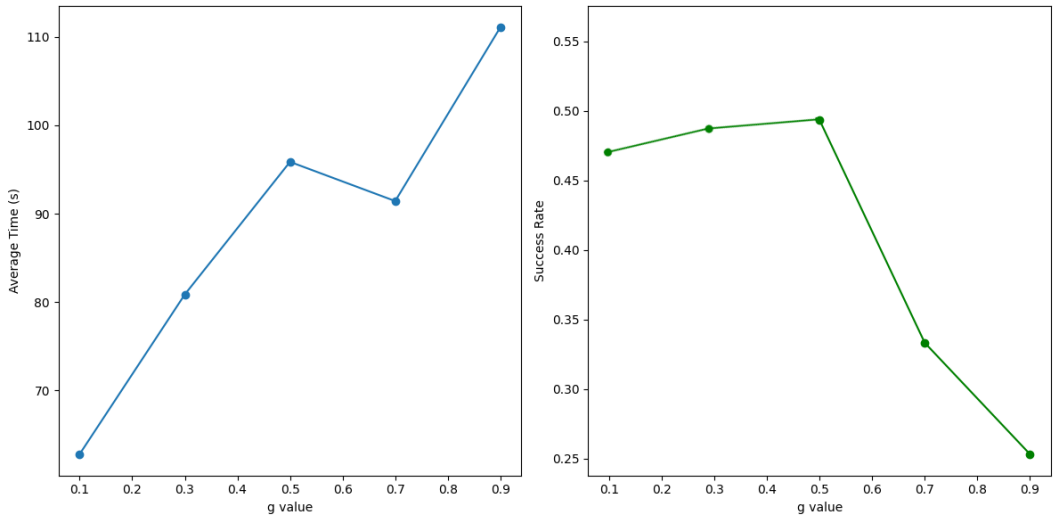

Figure 5: The Success rate and Runtime of WalkAAF plotted against the Parameter g.

The WalkAAf Algorithm with a heuristic based on Machine Learning uses an additional parameter g to control how often the heuristic is applied. The optimal value for g is determined by testing it against a Test Set of Argumentation Frameworks with various g values. The timeout for the Algorithm was less than 3 minutes, allowing us to find the optimal g because using the heuristic is computationally expensive. Figure 5 shows a plot of the Averegate Time the Algorithm uses for solving an Argumentation Framework against the different values of g and the success rate(how many Argumentation frameworks are solved) against different values of g. The WALKAAF Algorithm uses the Random Forest Classifier.

In the first plot, it is evident that the more the WalkAAF Algorithm uses the Heuristic, the longer it takes to solve an Argumentation Framework. As a result, when the value of g is 0.5 or higher, the algorithm's success rate drops suddenly because it times out more frequently.

The success rate of the algorithm increases gradually as the use of heuristics increases, until the value of g reaches 0.5 or higher.

Therefore the Heuristic based on Machine Learning has a positive effect on finding a single stable extension.

For further experiments, the value of g is set to 0.3 for the classical machine learning techniques which seem to be a good trade-off between runtime and success rate.

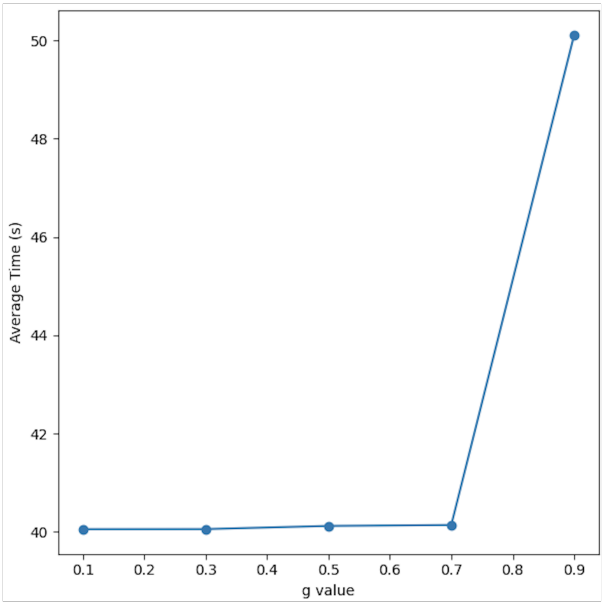

Figure 6: Runtime of WalkAAF using GCN plotted against the Parameter g.

Figure 6 displays the runtime of the WalkAAF Algorithm with GCN as a classifier. The comparison between the Random Forest classifier and GCN shows significant differences. There is nearly no difference in runtime when using GCN except for values of g over 0.7. This shows that the GCN is significantly faster than Random Forest even though you have to use the whole Argumentation Framework as a Feature Matrix.

For further experiments, the value of g is set to 0.4 for the Neural Network.

We tested the plain WalkAAF without any Heuristcs, then WalkAAF with a $Model_{rn}$ for choosing the next Argument for Flipping (Algorithm 1), then WalkAAF with the $Model_{in}$ choosing the initial Labelling of the Arguments (Algorithm 2) and then WalkAAF with the Combination of the two Models $Model_{rn}$ and $Model_{in}$ (Algorithm 3). First, the models are based on Random Forest and then on GCN. The algorithms are tested ten times on the benchmark set for their average runtime and success rate. The results are then displayed in the following table.

Table 22: Results of the different WalkAAF with and without Heuristic based on Machine Learning.

Algorithm	Avg. Runtime [s]	Success Rate
WalkAAF	33.16	0.69
WalkAAF RF (Algorithm 1)	149.97	0.63
WalkAAF RF (Algorithm 2)	46.56	0.66
WalkAAF RF (Algorithm 3)	143.27	0.69
WalkAAF GCN (Algorithm 1)	30.73	0.69
WalkAAF GCN (Algorithm 2)	32.33	0.65
WalkAAF GCN (Algorithm 3)	30.34	0.69

Overall, the WalkAAF without the Random Forests classifier outperforms the WalkAAF with it. The use of Random Forests led to longer runtimes and timeouts (6 minutes). Algorithm 2 performed better than Algorithm 1 and Algorithm 3 with the Random Forests classifier. This is because using the Random Forests classifier for argument selection was slow and resulted in overall worse performance.

Using Graph Convolutional Network with WalkAAF shows promising results. Algorithm 1 and Algorithm 3 outperform Algorithm 2 and even the original WalkAAF without heuristicss.

This thesis shows that Graph Convolutional Networks are a suitable option for heuristics based on machine learning in Data represented as Graphs, such as Abstract Argumentation Frameworks.

Overall the results depend on the model quality and classification problem.

The problem of selecting the next argument to switch its labell and the initial labelling was turned into a binary problem, but the models can be trained into a multi-classification problem or into a non binary classification with a different labeling approach.

6 Conclusion and Future Work

SLS Algorithms like WalkAAF can be optimized using machine learning-based heuristics. This thesis demonstrated that Graph Convolutional Networks (GCNs) outperform classical machine learning algorithms, such as Random Forests when used as heuristics. GCNs perform better due to the use of data that represent graphs, specifically Abstract Argumentation Frameworks.

Training the GCN model can be optimized through the use of other classification methods, such as binary and non-binary approaches, as well as unsupervised and semi-supervised learning. Additionally, incorporating different neural network algorithms can improve the model's performance.

Also, cooperating with other Heuristics not based on Machine Learning can lead to a better performance of the SLS Algorithm. Greedy Moves showed a promising Heiristic. But other heuristics, like choosing the Argument with the highest count of Attackers can be cooperated.

Furthermore, as with most machine learning models, the GCN models benefit from a more bigger, diverse dataset and extended training durations. This may lead to overall better predictions of the model because the models still have some Problems in deciding the next best move.

In conclusion, this work sets the stage for more advanced integration of neural networks into SLS algorithms. There is significant potential for further research in this direction.

References

[BCD07] T.J.M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10):619–641, 2007. Argumentation in Artificial Intelligence.

[BCG18] Pietro Baroni, Martin Wigbertus Antonius Caminada, and Massimiliano Giacomin. Abstract argumentation frameworks and their semantics. In *Handbook off Formal Argumentation*, 2018.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BS13] Stefano Bistarelli and Francesco Santini. Conarg: a tool to solve (weighted) abstract argumentation frameworks with (soft) constraints, 2013.

[CD21] Pádraig Cunningham and Sarah Jane Delany. K-nearest neighbour classifiers - a tutorial. *ACM Comput. Surv.*, 54(6), jul 2021.

[CDG+15] GÃ¼nther 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, mar 2015.

[CG09] Martin W. A. Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, nov 2009.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785â€“794, New York, NY, USA, 2016. Association for Computing Machinery.

[CGV19] 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.

[CST21] Paul D. Crutcher, Neeraj Kumar Singh, and Peter Tiegs. *Machine Learning*, pages 225–240. Apress, Berkeley, CA, 2021.

[DD17] Wolfgang Dvorak and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *FLAP*, 2017.

[Dun95] 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.

[EK21] Bradley J. Erickson and Felipe Kitamura. Magician's corner: 9. performance metrics for machine learning models. *Radiology: Artificial Intelligence*, 3(3):e200126, may 2021.

[Fri01] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

[Für10] Johannes Fürnkranz. *Decision Tree*, pages 263–267. Springer US, Boston, MA, 2010.

[GSKJ20] Aakanshi Gupta, Bharti Suri, Vijay Kumar, and Pragyashree Jain. Extracting rules for vulnerabilities detection with static metrics using machine learning. *International Journal of System Assurance Engineering and Management*, 12(1):65–76, sep 2020.

[HS15] Holger H. Hoos and Thomas StÃ¼tzle. Stochastic local search algorithms: An overview. In *Springer Handbook of Computational Intelligence*, pages 1085–1105. Springer Berlin Heidelberg, 2015.

[HYL18] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[JZH21] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, apr 2021.

[KB21] Muhamet Kastrati and Marenglen Biba. Stochastic local search: a state-of-the-art review. *International Journal of Electrical and Computer Engineering (IJECE)*, 11(1):716, feb 2021.

[KKT22] Jonas Klein, Isabelle Kuhlmann, and Matthias Thimm. Graph neural networks for algorithm selection in abstract argumentation. In *ArgML@ COMMA*, pages 81–95, 2022.

[KPW17] Markus Kröll, Reinhard Pichler, and Stefan Woltran. On the complexity of enumerating the extensions of abstract argumentation frameworks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 1145â€“1152, Melbourne, Australia, 2017. AAAI Press.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[KWT22] Isabelle Kuhlmann, Thorsten Wujek, and Matthias Thimm. On the impact of data selection when applying machine learning in abstract argumentation. In *Proceedings of the 9th International Conference on Computational Models of Argumentation (COMMA'22)*, 2022.

[Lau22] Sindayigaya Laurent. Machine learning algorithms: A review. *Information Systems Journal*, ISJ-RA-3392:6, 08 2022.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825â€“2830, nov 2011.

[RN21] Stuart Russell and Peter Norvig. *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland, 2021.

[SKC93] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability*, 1993.

[SLM92] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI Conference on Artificial Intelligence*, 1992.

[Thi18] Matthias Thimm. Stochastic local search algorithms for abstract argumentation under stable semantics. In *Proceedings of the 7th International Conference on Computational Models of Argumentation (COMMA'18)*, 2018.

[Web10] Geoffrey I. Webb. *Naïve Bayes*, pages 713–714. Springer US, Boston, MA, 2010.