

Weak-Admissibility Semantics in Abstract Argumentation Frameworks: Using Statistical Learning and Machine Learning to Determine Credulous Acceptability

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Computer Science

submitted by

Carla Irán Sánchez Aguilar

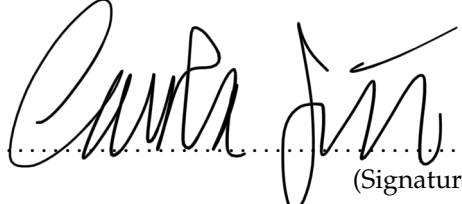
First examiner: Prof. Dr. Matthias Thimm
Artificial Intelligence Group

Advisor: Prof. Dr. Matthias Thimm
Artificial Intelligence Group

Statement

Ich erkläre, dass ich die Masterarbeit 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 Arbeit 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 Arbeit 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.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
I agree to have this thesis published on the webpage of the artificial intelligence group.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The source code is available under a GNU General Public License (GPLv3).	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Göttingen, 20.03.23 

(Place, Date) (Signature)

Abstract

In abstract argumentation, self-attacking arguments and the unequal treatment of arguments in odd- and even-length attack cycles are some of the issues of classic semantics that are addressed by weak admissibility semantics. Most computational problems under weak admissibility semantics are PSPACE-complete. Our research presents a direct implementation to decide credulous acceptability under weak admissibility semantics. On the other hand, statistical analysis and machine learning have been widely used to solve complex problems with the aid of data. The question arises whether these tools could also provide insights about the credulous acceptability of arguments under weak admissibility semantics. This research answers this question by gathering data on argumentation frameworks and arguments, determining the credulous acceptability status of each argument, and statistically analyzing the data. Our findings, based on logistic regression, show that simple techniques are able to determine credulous acceptability under weak admissibility semantics better than chance. Furthermore, based on the results of the analysis, we assess the feasibility of using more complex models (e.g. Support Vector Machines, Graph Convolutional Networks) to solve the above mentioned problem, showing moderately better results.

Acknowledgements

I would like to express my deepest appreciation to Prof. Dr. Matthias Thimm and his team for their invaluable support and advice during the completion of this thesis. Thank you all for your patience and ideas.

Special thanks go to Dr. Wolfgang Dvorak and Dr. Markus Ulbricht for helping me gaining a deeper understanding of how to solve problems in weak admissibility semantics.

I would also like to express my gratitude and never ending love to my bedazzling daughters, Valeria and Indira, and to my loving partner, Malik, who keep me up to the mark and far beyond.

My sincere appreciation goes to the people at Praxis Hassan, your dedication and engagement are exemplary and very contagious. Ihr seid einfach die besten.

Gracias mamá, gracias papá, por todo...

Contents

1	Introduction	1
2	Background Literature	5
2.1	Fundamentals of Abstract Argumentation	5
2.1.1	Classic Admissibility	6
2.1.2	Weak Admissibility	7
2.2	Computational Problems in Abstract Argumentation	8
2.2.1	Verifying weak admissibility of a set of arguments	9
2.2.2	Deciding credulous acceptability under weak admissibility semantics	10
2.2.3	Solving Search Problems with SCC-Recursiveness	10
2.3	Machine Learning in Abstract Argumentation	12
2.3.1	Graph Convolutional Neural Network (GCN) Models	13
2.3.2	Message Passing Neural Network (MPNN) Models	14
3	Credulous Acceptability under W-Admissibility	17
3.1	Types of Implementation	17
3.2	Problem Reformulation	18
3.3	Preprocessing Steps	19
3.3.1	Exclusion of Self-Attacking Arguments	19
3.3.2	Computing the Grounded Extension	20
3.4	Enumerating Weakly Preferred Extensions	21
3.4.1	Computing Strongly Connected Component (SCC)	21
3.4.2	Finding Weakly Preferred Extension with SCC-Recursiveness	22
3.4.3	Generating Weakly Preferred Extensions	23
3.4.4	Verifying weak admissibility of a set of arguments	24
3.5	Solver Design	26
3.6	Experimental Evaluation	26
3.7	Limitations	30
4	Exploratory Data Analysis	31
4.1	Data Collection	31
4.1.1	Argumentation Framework (AF) Sample Description	32

4.1.2	Experimental Data Set	34
4.2	Argument-level Analysis	36
4.2.1	Exploratory Analysis	36
4.2.2	Data Transformation	41
4.3	Logistic Regression	43
4.3.1	Experimental Setup	44
	Results	44
5	Machine Learning (ML) Experiments and Analysis	49
5.1	Support Vector Machine (SVM)	49
5.1.1	Experimental Setup	51
	Results	52
5.2	Graph Convolutional Neural Network (GCN)	53
5.2.1	Experimental Setup	54
	Results	55
6	Discussion and Conclusion	57
6.1	Weak Admissibility Semantics	57
6.2	Statistical Analysis and Logistic Regression	58
6.3	Machine Learning (ML) Models	59
6.4	Recommendations for Future Work	60
A	Weak Admissibility Exact Solver	63
A.1	Algorithm using SCC-recursiveness	63
	Bibliography	73

List of Figures

3.1	Composition of the International Competition on Computational Models of Argumentation (ICCMA) Sample	28
3.2	Differences between solved and unsolved AFs in the ICCMA sample	29
4.1	Composition of the Generated AFs	34
4.2	Differences between the unsolved and the solved AFs	35
4.3	In- and Out-degree Histogram	37
4.4	Histogram in- and out-degree excluding self-attacking arguments . .	39
4.5	Differences between non-accepted and accepted arguments	40
4.6	Differences between credulously accepted status: ratio features . . .	42

List of Tables

3.1	ICCMA Sample	27
3.2	ICCMA Solved Sample Description	28
3.3	Graph Types in Dataset	29
3.4	Solved Dataset Run-Time	30
4.1	Number of arguments in the Generated AF Sample	33
4.2	Number of attacks in the generated AF sample	33
4.3	Partitions of the solved AF Sample	35
4.4	Argument Properties Summary	37
4.5	Self-attacking and Credulously accepted arguments	38
4.6	Data set summary: excluding self-attacking arguments	38
4.7	Spearman correlation coefficient of argument features	39
4.8	Spearman correlation coefficient of argument ratio features	43
4.9	Logistic Regression Analysis of credulous acceptability	45
4.10	Odd ratios	46
4.11	Logistic Model Assessment	47
4.12	Logistic Model Assessment on the Test data set	47
5.1	Linear SVM Results ($C = 0.0001$)	52
5.2	4th Degree Polynomial SVM Results ($C = 1.0$)	52
5.3	RBF SVM Results ($C = 0.1, \gamma = 1.0$)	53
5.4	Sigmoid SVM Results ($C = 0.01, \gamma = 0.01$)	53
5.5	FM2 Model Results	55

List of Algorithms

1	Verifying w-admissibility of a set S	9
2	Deciding credulous acceptability under adm^w semantics	10
3	Finding Weakly Preferred Extensions with SCCs	12
4	Generating the Grounded Extension	21
5	Finding Weakly Preferred Extensions	22
6	Generating Weakly Preferred Extensions	23
7	Verifying Weak Admissibility	24

List of Abbreviations

AI Artificial Intelligence

AF Argumentation Framework

AGNN Argumentation Graph Neural Network

CNN Convolutional Neural Network

GCN Graph Convolutional Neural Network

ICCMA International Competition on Computational Models of Argumentation

ML Machine Learning

MCC Matthews Correlation Coefficient

MPNN Message Passing Neural Network

RBF Radial Basis Function

SCC Strongly Connected Component

SVM Support Vector Machine

Chapter 1

Introduction

Two important issues have motivated the use of argumentation in Artificial Intelligence (AI), one related to reasoning with incomplete and uncertain information and the second is related to explanation under the same circumstances (Bench-Capon and Dunne, 2007; Simon, 1996; Atkinson et al., 2017). Efforts to advance this field have led to the development of abstract argumentation frameworks. These frameworks are based on a premise discussed by Simon (1996). This premise states that it is the organization of components in an artificial system that matters and not their individual internal properties. In an abstract argumentation framework only the attack relation among single arguments matter, while ignoring the internal structure and properties the arguments themselves.

Dung (1995) laid the foundation towards formalization of abstract argumentation frameworks by establishing an analogy to human argumentation. He pleaded that, in a debate among rivaling positions, the winning argument would be that which can offer each counterargument a response, thus remaining undefeated. He then proceeded to formally define an argumentation framework in terms of a set of arguments and binary relations among these arguments. Furthermore, he formalized the different ways in which undefeated arguments could be singled out from defeated arguments. Dung called these collections of undefeated arguments **extensions** and the rules followed to procure these extensions were called **semantics**. Dung also recognized some limitations of his semantics. He realized that in some particular constellations of argumentation frameworks (e.g. frameworks containing an argument indirectly attacking a defender argument or the existence of a self-defeating argument in the framework), his semantics could affect the existence and the composition of extensions in an inconsistent manner.

Serious attempts to address and manage the inconsistencies mentioned above have been presented in the definitions of **CF2** and **stage2** semantics (Gaggl and Woltran, 2013; Baroni et al., 2018). More recently, a new semantics called weak admissibility semantics has been proposed by Baumann et al. (2020b) and Baumann et al. (2020a). Weak admissibility semantics try to distance themselves from naive semantics as defined by Gaggl and Woltran (2013) and Baroni et al. (2018), while

preserving most of the properties of the classical semantics proposed by Dung and also addressing the issues of self-defeating arguments, and the equal treatment of even- and odd-length attack cycles.

The expert reader might wonder, given the existence of so many different semantics (Baroni et al., 2018), why do we need yet another form of semantics. The answer lies in the fact that some semantics can only be applied to very specific framework structures, such semantics fail in the presence of so called zombie-arguments, i.e. arguments that have very little chance of being accepted but still prevent other arguments from being accepted (Baumann et al., 2020b).

Dvořák et al. (2021) studied computational problems involving weak admissibility semantics, the complexity analysis resulted in the conclusion that decision problems in arguing with weak-admissibility semantics belong to the class of “problems that can be solved using polynomial space of memory” also known as **PSPACE-Complete**.

Given the complexity of some the decision problems posed by weak admissibility semantics (Dvořák et al., 2021), in particular the problem of acceptability (either credulous or skeptical) with respect to a given semantics, we argue that following an empirical approach based on exploiting implicitly and explicitly available information in an AF, might help us identify beforehand certain features of the arguments of the AF that make them more prone to be part of an extension and also in deciding acceptability of an argument. This information might be useful in the implementation of solution search algorithms that use this information as heuristics to help guide the search as proposed by Craandijk and Bex (2020).

In another vein, during the past two decades, increasing availability of data and computing power has fostered major breakthroughs in the field of ML. The philosophy behind ML techniques moves away from hard-wiring rules in computational agents or computer programs. The aim of ML is to learn from experience. We teach our agent or computer program by exposing it to examples of correct behavior. The agent learns to make predictions based on the seen examples and tries to improve with each new instance seen, we say our agent is “learning”. Basically, we expose our computer program to a very large amount of data for the purpose of modeling, prediction or control (Russell and Norvig, 2021). This approach has proven to be a success in solving some challenging problems.

To the best of our knowledge and at the time of this writing only a few teams (Kuhlmann and Thimm, 2019; Craandijk and Bex, 2020; Malmqvist et al., 2020) have attempted the use of ML to approach the problem of argument acceptability (either credulous or skeptical) under classic semantics showing promising results. Furthermore, we have no antecedents about the application of ML techniques to weak-admissibility semantics. In this master thesis, we will explore the use of statistical analysis and ML techniques to determine credulous acceptability of arguments in an AF under weak admissibility semantics. But first, we will design and implement a solver that can provide the exact acceptability status of arguments in an AF in a moderate amount of time.

The goals of the present study are three-fold:

1. To develop an exact solver for credulous acceptability under weak admissibility.
2. To explore the feasibility of using statistical analysis to identify patterns of credulous acceptability under weak admissibility semantics.
3. To apply ML techniques to determine credulous acceptability under weak admissibility semantics. More specifically, we will train a Support Vector Machine (SVM) and a Graph Convolutional Neural Network (GCN), compare and evaluate its performance in the face of the problem mentioned above.

In the following chapter, we review fundamental definitions of classic semantics followed by a description of weak admissibility semantics and we highlight the differences among the two. We explore typical computational problems of this relatively new semantics and discuss their complexity; we introduce some algorithmic solutions to the problem of interest. We also present a review of studies applying ML techniques in classical semantics.

Chapter 3 offers a description of the design and implementation of the software developed to solve the problem of credulous acceptability under weak admissibility semantics. We implement an algorithm based on the ideas proposed by Dvořák et al. (2022) based on SCC-recursiveness. In Chapter 4 we provide an overview of the data set collected for this study. We analyze a set of AFs from the point of view of descriptive statistics and try to find argument features that influence credulous acceptability under weak admissibility semantics. We design a model based on classic statistical tools that allow us to identify credulously accepted arguments in an AF.

More advanced techniques are explored and summarized in Chapter 5. Two models of classification are explored and compared: Support Vector Machine (SVM) and Graph Convolutional Neural Network (GCN). Finally, in Chapter 6 we discuss the limitations of our study as well as the implications for future research. We discuss major advantages and disadvantages of each prediction model and offer a comparison in terms of binary classification metrics.

Chapter 2

Background Literature

2.1 Fundamentals of Abstract Argumentation

We start our review of literature by recalling some basic definitions widely used in abstract argumentation frameworks (Dung, 1995).

One of many Dung's contributions was the idea that arguments could be treated as abstract atomic entities whose interaction could be described in terms of attack relationships among arguments. This simplified approach had a massive impact in the field, making it the de-facto standard.

An **Argumentation Framework (AF)** is formally represented as a directed graph $F = (A, R)$. The set A is the set containing arguments in F and represents the nodes in the directed graph. The set $R \subseteq A \times A$ defines the binary relation between any two arguments $a, b \in A$ such that an attack exists among these arguments. In the directed graph, attacks between arguments are represented by arrowed edges depicting the direction of the attack.

From now on we will assume finite argumentation frameworks, that is A is finite and R is also finite.

An **attack** between two arguments in an AF F is defined as follows: an argument $a \in A$ is said to attack an argument $b \in A$ if $(a, b) \in R$. In other words, there is a directed edge from a to b in F . Conversely, if b attacks a then we say that a is attacked by b . Arguments can also attack sets of arguments in F and vice versa. We can formally express these types of attack in the following manner: assume $B \subseteq A$, let $a \in A$ and $b \in B$, if a attacks b then we say that a also attacks B . If b attacks a , we say that A is attacked by b . Let $B \subseteq A$ and $C \subseteq A$, if $b \in B$ attacks $c \in C$, we say that B attacks C in AF .

Given an argumentation framework, the goal is to find sets of arguments which are compliant with the following conditions:

1. They internally coherent, i.e. conflict-free. A set $B \subseteq A$ is said to be **conflict-free** if and only if for any two arguments $a, b \in B$ an attack between a and b does not exist in AF , i.e. $(a, b) \notin R$ and $(b, a) \notin R$. Given an AF $AF = \langle A, R \rangle$, we denote the set of conflict-free sets in AF as $cf(AF)$.

2. They also present a coherent world-view with respect to the given argumentation framework, that is, the members of set is collectively acceptable.

To comply with the conditions stated above, we use a series of rules called semantics. Dung (1995) based his development of semantics and extensions on the concept of **acceptability** and the concept of **admissibility** which we briefly review in the following sections.

2.1.1 Classic Admissibility

Dung began his treaty of admissibility by first establishing the concept of **acceptability**, he stated that given an argumentation framework $F = (A, R)$ and a set of arguments $C \subseteq A$, an argument $a \in A$ is **acceptable** with respect to C if and only if for any argument $b \in A$ that attacks a , then b is attacked by C .

Expanding on this definition of acceptability, given an argumentation framework $F = (A, R)$ and a set of arguments $C \subseteq A$, Dung proposed two fundamental conditions to identify a set of arguments as **admissible**:

1. the set of arguments C is **conflict-free**, and
2. every argument $a \in C$ is **acceptable** with respect to C .

These two conditions have profound implications in the search of arguments that can be part of an **admissible** set in the classical sense of admissibility. For a set of arguments to be admissible we first require the set to be internally unimpeachable, meaning the arguments in the set cannot attack one another. Second, the set needs to defend its arguments against every attack coming from the arguments not included in the set, reflecting approximately a law of retaliation “an eye for an eye”. In other words, any attack from any argument not included in the set to an argument in the set must be met with a counter-attack coming from any argument in the admissible set.

Four types of extensions can be derived from the conditions above:

- Preferred extension: an admissible set $C \subseteq A$ is a **preferred** extension if C is maximal with respect to set inclusion.
- Stable extension: let $C \subseteq A$ be conflict-free, C is a **stable** extension if and only if C attacks each argument which is not included in C .
- Complete extension: an admissible set $C \subseteq A$ is a **complete** extension if an only if each argument which is acceptable with respect to C is a member of C .
- Grounded extension: a set $C \subseteq A$ is a **grounded** extension if and only if C is a minimal with respect to set inclusion complete extension of AF .

Baumann et al. (2020b), Baumann et al. (2020a) and Dvořák et al. (2021) argue that classic admissibility is problematic in the presence of self-defeating arguments as well as arguments involved in an odd-length attack cycles. Such so-called zombie arguments can prevent “reasonable” arguments from being accepted even if they do not stand a chance of being accepted themselves. To address these issues they propose weak admissibility semantics and we explain it in the following section.

2.1.2 Weak Admissibility

Weak-admissibility intends to address the issues mentioned previously by relaxing the second condition of classic admissibility. Weak admissibility semantics propose that a conflict-free set only has to respond to attacks from “serious” arguments. Basically, we are allowed to ignore attacks from zombie arguments.

Dvořák et al. (2021) and Baumann et al. (2020b) use a recursive procedure based on what they call the **reduct** with respect to a set of arguments which is defined as follows:

Let $F = (A, R)$ be an argumentation framework, given $C \subseteq A$, let C^+ be the set of arguments that receive an attack from any of the arguments contained in C . Formally, let $C^+ = \{a \in A \mid b \in C \wedge (b, a) \in R\}$.

Furthermore, let C^\diamond be the **range** of C defined by $C^\diamond = C \cup C^+$. Then, the **reduct** of F with respect to C is the argumentation framework $F^C = \langle C^*, R \cap (C^* \times C^*) \rangle$, where $C^* = A \setminus (C \cup C^+) = A \setminus C^\diamond$ (Baumann et al., 2020b; Dvořák et al., 2021).

The set C is said to be **weakly admissible**, or w-admissible, if and only if the following conditions are both met:

1. C is **conflict-free**, and
2. an attacker of C does not belong to any of the weakly admissible sets of the reduct of AF with respect to C .

However, weakening the concept of admissibility has consequences in the way we now account for the defense of the arguments, particularly because only serious attacks must receive a counter-attack, that is, if C receives an attack from an argument in the reduct, either the set C contains an argument that offers a counter-attack (in which case the attacker would not appear in the reduct, because it would be part of the range) or we have to make sure that the attacker is not weakly admissible in the reduct. Formally, given an argumentation framework $F = \langle A, R \rangle$ and two sets of arguments $C, X \subseteq A$, we say C **weakly defends** X if and only if for any attacker a of X we have:

1. C attacks a , or
2. a does not belong to any w-admissible set of the reduct with respect to C , $a \notin C$, and $X \subseteq X'$, such that X' belong to an w-admissible set of F .

Besides weak-admissible extensions, denoted by $adm^w(F)$, weak-admissibility semantics recognize the existence of three types of extensions given an argumentation framework $F = (A, R)$:

1. Weakly preferred extension: a weakly admissible set $C \subseteq A$ is called **w-preferred** if C is maximal with respect to set inclusion, we say $C \in pr^w(F)$.
2. Weakly complete extension: a w-admissible set $C \subseteq A$ is called **w-complete** if and only if for any X , such that $C \subseteq X$ and X is w-defended by C , we also have $X \subseteq C$, we say $C \in co^w(F)$.
3. Weakly grounded extension: a w-complete set $C \subseteq A$ is called **w-grounded** if and only if C is minimal with respect to set inclusion, we say $C \in gr^w(F)$.

2.2 Computational Problems in Abstract Argumentation

Given an abstract argumentation framework $F = (A, R)$ and a semantics σ , the type of computational problems most commonly addressed in the abstract argumentation literature belongs to one of the following categories (Charwat et al., 2015; Lagniez et al., 2020):

- Decision problems: decide whether an argument $a \in A$ can be either credulously or skeptically accepted with respect to a given semantics.
 - Credulous Acceptability (**CD**- σ): returns *True* if the argument a belongs to any of the extensions of F under σ , returns *False* otherwise. Formally, $a \in \bigcup \sigma(F)$.
 - Skeptical Acceptability (**SD**- σ): returns *True* if the argument a belongs to all of the extensions of F under σ , returns *False* otherwise. Formally, $a \in \bigcap \sigma(F)$.
- Counting problems: compute the number of extensions in F given a semantics σ .
- Search problems: find extensions of F given a semantics σ .

Given an AF $F = (A, R)$ and a weak admissible semantics $\sigma \in \{adm^w, pr^w, co^w, gr^w\}$, the complexity analysis showed (Dvořák et al., 2021; Dvořák et al., 2022) that decision problems of credulously acceptability belong to the class of PSPACE-Complete problems. While skeptical acceptability under a semantics $\sigma \in \{pr^w, co^w, gr^w\}$ is also considered to belong the class of PSPACE-Complete problems, skeptical acceptability under weak admissibility is trivial, due to the fact that the empty set is always w-admissible.

In the course of this work we dedicate our efforts to the problem of credulous acceptance under weak admissibility semantics. Below, we survey algorithms suggested by Dvořák et al. (2021) and Dvořák et al. (2022), these algorithms were designed with the purpose of verifying whether a set of arguments was weakly admissible and serves as a building block to later determine the credulous acceptability of one argument.

2.2.1 Verifying weak admissibility of a set of arguments

Solving problems under weak admissibility semantics is difficult. In an attempt to simplify the problem at hand, Baumann et al. (2020b) recommend the elimination of all self-attacking arguments as well as their incoming and outgoing attacks. Although this pre-processing step delivers small gains, we make it mandatory and will assume in the following sections that the AF has been stripped off of all self-attacking nodes.

Given an AF $F = (A, R)$, let F be stripped-off self attacking arguments and let $S \subseteq A$, verifying weak admissibility of this set in F according to the iterative procedure described by (Dvořák et al., 2021), consists of three main steps, as illustrated in Algorithm 1.

1. Verifying that S is conflict-free, i.e. $S \in cf(F)$. If not, S is not w-admissible.
2. Compute the *reduct* of F with respect to S .
3. Iterate over all subsets T of the *reduct* that contain at least one attacker of S .
 - (a) Verify whether T is w-admissible in the *reduct* of F w.r.t S . If so, S is not w-admissible.
4. If no set T containing at least one attacker of S in the *reduct* could be found, the set S is w-admissible.

Algorithm 1 Verifying w-admissibility of a set S

Require: $S \subseteq Args(F)$

```

1: procedure VER-WADM( $S, F$ )
2:   if NOT ISCONFLICTFREE( $F, S$ ) then
3:     return False
4:    $F^S \leftarrow$  REDUCT( $F, S$ )
5:   for all  $T \in \{T \mid T \subseteq Args(F^S) \wedge T \text{ attacks } S\}$  do
6:     if VER-WADM( $F^S, T$ ) then
7:       return False
8:   return True

```

By decreasing the size of the AF — applying the *reduct*— in the last step in Algorithm 1, the recursion depth is bounded from above by the number of nodes in the original AF (Dvořák et al., 2021).

2.2.2 Deciding credulous acceptability under weak admissibility semantics

Given an AF $F = (A, R)$ and an argument $a \in A$. Let $Args(F) = A$, i.e. the set of arguments A contained in F , let $adm^w(F)$ denote the weakly admissible sets of F . One approach to solve the problem of credulous acceptability under w-admissible semantics consists of two building blocks (Dvořák et al., 2021): the first one relates to finding a set of arguments containing a and then verifying whether this set is w-admissible, see Algorithm 2.

1. Find a set $S \subseteq A$ such that $a \in S$, and S has not been previously processed.
2. Test whether $S \in adm^w(F)$, if true, a is credulously accepted under adm^w semantics and we can stop. If not test the next set S until we have no more new sets to test, then a is not credulously accepted under adm^w semantics.

Algorithm 2 Deciding credulous acceptability under adm^w semantics

Require: $a \in Args(F)$

- 1: **procedure** CREDACC-WADM(F, a)
- 2: $\mathcal{SS} \leftarrow \text{GENERATESUBSETS}(F, a)$
- 3: **for all** $S \in \mathcal{SS}$ **do**
- 4: **if** VER-WADM(F, S) **then**
- 5: **return** *True*
- 6: **return** *False*

Notice how Algorithm 2 describes a blind search for a solution (Pearl, 1984), the order in which the sets of arguments containing the argument a are tested for w-admissibility is not governed by any criterion other than the order in which they were generated. The approach mentioned above also involves another kind of decision problem: the problem of deciding or verifying whether a set of arguments $S \subseteq A$ is w-admissible. We describe a solution to this problem in the following section.

Up to now, we have presented alternatives to solving decision problems under weakly admissible semantics for one argument and for a set of arguments. For the purpose of our research and for the sake of implementing an exact solver that provides the precise credulous acceptance status of every argument in an AF, the following section presents a constructive approach to generating extensions in abstract argumentation.

2.2.3 Solving Search Problems with SCC-Recursiveness

All classic semantics lend themselves to the idea of partitioning an AF into Strongly Connected Components to generate all of its extensions component-wise (Baroni et al., 2005).

We call a directed graph **strongly connected** if given any pair of nodes in the graph, there are directed paths between them. Formally, let $F = (A, R)$ denote a directed graph, and for any two nodes $a, b \in A$, there are paths from a to b and from b to a . Then we say F is strongly connected (Tarjan, 1971; Tarjan, 1972, Definition 4).

Furthermore, we say two nodes $a, b \in A$ in F are equivalent if there is a closed path from a to a which also contains b . Let A_i denote one of the n sets of equivalent nodes in F , with $i = 1, \dots, n$. Suppose $F_i = (A_i, R_i)$ and $R_i = \{(a, b) \in R \mid a, b \in A_i\}$. We call each F_i a Strongly Connected Component (SCC) of F , provided F_i is strongly connected and for every subgraph F_j, F_k in F neither the nodes A_i nor the edges R_i are subsets of A_j, R_j , respectively (Tarjan, 1972, Lemma 8).

The nodes in a directed graph can be partitioned into SCCs, where the nodes in each component are strongly connected, i.e. every node contained in the SCC can be reached through a directed path from any other node in the same SCC (Skiena, 2020).

Similarly, the AF is represented as a directed graph and its nodes are partitioned into components, so that every node is assigned to only one component. The assignment condition depends on the node being reachable from any other node also contained in the SCC by a directed path. Partitioning a directed graph into its SCCs inevitably results in a directed acyclic graph of SCCs with one or more initial SCCs from which we can start the computation of extensions in a sequential form using a **base function** (Baroni et al., 2005).

SCC-recursiveness represents a divide-and-conquer approach to constructing extensions. Instead of dealing with the whole AF at once, we can work with single SCCs starting from the initial SCCs — SCCs with no incoming attacks by any other SCC — and working our way “down” until we have exhausted all possibilities of finding extensions (Dvořák et al., 2022). SCC-recursiveness also requires that a semantics can be characterized by a base function. The base function should be capable of generating the extensions of the AF consisting of a single SCC for the desired semantics.

Unfortunately, SCC-recursiveness cannot be applied to all weak admissibility semantics. In particular, Dvořák et al. (2022) proved that we can use SCC-recursiveness without further ado to find weakly preferred extensions in a given AF. The procedure consists of partitioning an AF into SCCs (Baroni et al., 2005). We now assume a new AF consisting of only the arguments contained in the initial SCC. Remember that this is an initial SCC, therefore, there are no incoming attacks, and for the moment we will ignore the outgoing attacks. The idea from here is to apply a base function to generate all weakly preferred sets of this SCC. After generating the weakly preferred extensions for this portion of the AF, we encounter one of the following situations:

1. At least one weakly preferred extension was found.
2. No weakly preferred extensions were found.

In the first case, in which we found at least one w -preferred extension in the initial SCC, we compute the reduct of the AF with respect to the extension found and recursively continue the process of finding w -preferred extensions in the reduced AF, each new extension found is added to the presiding one by a union of sets.

In the second case, in which no w -preferred extensions could be found in the SCC, we have to eliminate the arguments contained in the SCC, as well as their out-going attacks, from the graph representing the AF. Then, we find w -preferred extensions in the new downsized AF.

Algorithm 3 Finding Weakly Preferred Extensions with SCCs

```

1: procedure FINDWPREFEXTENSIONS( $F$ )
2:    $SCCS \leftarrow$  STRONGLYCONNECTEDCOMPONENTS( $F$ )
3:   for all initial  $SCC \in SCCS$  do
4:      $wprefext \leftarrow$  BASISFUNCTION( $F, SCC$ )
5:     if  $wprefext \neq \{\emptyset\}$  then
6:        $F^{wp} \leftarrow$  REDUCT( $F, wprefExt$ )
7:        $wprefext \leftarrow wprefext \cup$  FINDWPREFEXTENSIONS( $F^{wpe}$ )
8:     else
9:       Remove  $SCC$  from  $F$ 
10:     $wprefext \leftarrow$  FINDWPREFEXTENSIONS( $F$ )
11:  return  $wprefext$ 

```

SCC-recursiveness offers certain advantages in the computation of w -preferred extensions, because the partitioning of the AF and the subsequent reduction in size make run-time scale exponentially with the size of the largest SCC but run-time also scales in polynomial time with the size of the AF (Dvořák et al., 2022). Furthermore, certain graph structures allow the complexity of weak admissibility semantics to approach their classic counterparts, that is their complexity decreases considerably.

We cannot and should definitely not count on AFs having convenient structures and even with clever algorithms, problem solving in weak admissibility semantics continues to be hard. Maybe it is time to explore some data-centric alternatives and see how statistical tools and Machine Learning (ML) can contribute to solving problems in abstract argumentation under weak admissibility semantics.

2.3 Machine Learning in Abstract Argumentation

In Section 2.1, we established that argumentation frameworks can be defined as a directed graph with nodes representing the arguments in the AF and the directed edges characterizing the attacks. This kind of representation has enabled the application of advanced techniques in ML (Kuhlmann and Thimm, 2019; Craandijk and Bex, 2020; Malmqvist et al., 2020). These ML models share a common goal: solving the computational problem of deciding the credulous acceptability of arguments

under preferred semantics in the classical sense. Mainly, this problem can be cast as a binary classification problem.

To this effect we were able to identify two classes of implementations. On the one hand, we have implementations that partially exploit the directed nature of the graph and are based on spectral Graph Convolutional Neural Network (GCN) (Kuhlmann and Thimm, 2019; Malmqvist et al., 2020); on the other hand, we have an implementation which exploits the digraph more extensively and are based on spatial Message Passing Neural Network (MPNN) models (Kollias et al., 2022). Both classes are discussed below.

2.3.1 Graph Convolutional Neural Network (GCN) Models

Graph Convolutional Neural Network (GCN) build on the idea that graphs can be treated in a similar way as visual data. The goal is to learn a function whose input consists of two components (Kipf and Welling, 2016):

1. A description of the graph structure in matrix form. Commonly, the adjacency matrix of an undirected graph.
2. A feature matrix \mathbf{X} of size $N \times D$, where N denotes the number of nodes in the graph and D denotes the number of input features.

The learned function produces an output at the node-level denoted by a matrix \mathbf{Y} of size $N \times F$, where F denotes the number of output features per node. The learned function contains filters with a limited perceptive field which then feed into an activation layer, the neural network activates when certain patterns are present at some spatial position of the input.

Kuhlmann and Thimm (2019) set out to train a classification model based on a modified version of the Graph Convolutional Neural Network (GCN) proposed by Kipf and Welling (2016). The aim of the GCN model was to determine the credulous acceptability status of the arguments in an AF under preferred semantics. The training and test set in this experiment was partly generated by *probo* (Cerutti et al., 2014b) and partly by *AFBenchGen* (Cerutti et al., 2014a), additional test data was obtained from the International Competition on Computational Models of Argumentation (ICCMA) 2017 ¹. The results in terms of overall accuracy on the test data were rather moderate, ranging from 60 to 80 percent of correctly classified arguments. Further analysis shows that the proportion of arguments that were credulous acceptable for real and were classified as such by the model was at most 27 %. The model showed a better performance at classifying arguments as non credulously accepted, with a rate of up to 98% correctly classified arguments. However, the accuracy loss experienced by using the GCN model was to some degree compensated by the faster running time as compared to exact methods. These results support the widely known trade-off between finding optimal solutions for complex problems and the cost of arriving at those solutions (Simon, 1996; Pearl, 1984).

¹<http://argumentationcompetition.org/2017/results.html>

Later on, Malmqvist et al. (2020) devised an experiment in which the input were set to include node embeddings generated using **DeepWalk** (Perozzi et al., 2014) besides the adjacency matrix of the undirected graph derived from the AF. The model consisted of 4 to 6 convolutional hidden layers. These hidden layers contained an additional residual connection composed of a feature matrix containing the in- and out-degree of each node and the normalized adjacency matrix. The goal of this model was to classify arguments in terms of credulous and skeptical acceptability with respect to preferred semantics. The training and test set in this experimental setting was conformed by a selection of the benchmark dataset used during the International Competition on Computational Models of Argumentation (ICCMA) 2017.

Malmqvist et al.'s model with six convolutional layers showed superior accuracy as compared to Kuhlmann and Thimm's model applied to the same training/test set. Particularly, the correct classification of credulously accepted arguments went from 10% using Kuhlmann and Thimm's model to 71% using Malmqvist et al. with an unbalanced dataset. Regarding the classification of not credulously accepted arguments, classification accuracy was lower for Malmqvist et al. with 92% as compared to 97%. Regarding solution cost, Malmqvist et al. (2020) findings were consistent with Kuhlmann and Thimm (2019), the cost of training and using the model to classify arguments was relatively low when compared to the cost of computing an exact solution with a good solver.

2.3.2 Message Passing Neural Network (MPNN) Models

Message Passing Neural Networks are an alternative technique to processing graphs in ML (Gilmer et al., 2020). This technique aims at encoding information on the graph into a feature vector and consists of two phases:

Message passing phase collects a series of updates of a node in T steps. The updates consist of a hidden state and aggregated messages from the node's neighbor.

Readout phase computes a feature vector for the whole graph using a readout function.

Unlike the GCN model used in Kuhlmann and Thimm (2019) and Malmqvist et al. (2020), MPNN can be easily extended to fully harness the directed nature of attacks in an argumentation framework (Gilmer et al., 2020). Remember that in both studies, the adjacency matrix was always formed from the undirected graph of the argumentation framework. The techniques described in Kipf and Welling (2016), Kuhlmann and Thimm (2019), and Malmqvist et al. (2020) make one crucial mathematical assumption: the adjacency matrix is symmetric and thus its Laplacian is symmetric too, the Laplacian is also positive semi-definite. This assumption derives from the fact that undirected graphs produce symmetric adjacency matrices and Laplacians which can be later factorized using **eigenvalues** and **orthonormal**

eigenvectos (Strang, 2019). Assuming a positive semi-definite Laplacian is crucial for the application of spectral convolutions and ultimately the filtering step. However, the Laplacian of a directed graph cannot always be factored in the same way a Laplacian of an undirected graph can be factored (Veerman and Lyons, 2020; Kollias et al., 2022).

To circumvent this issue with directed and undirected graphs and account for the directed nature of the graph on an argumentation framework, Craandijk and Bex (2020) developed a model called Argumentation Graph Neural Network (AGNN) that applied and extended the concepts of a MPNN to directed graphs. The goal of the AGNN model was to determine credulous and skeptical acceptability for each one of the classic semantics: grounded, preferred, stable and complete semantics. The model was trained on a data set containing one million AFs with a number of arguments between five and 25. The data set was generated using the ICCMA 2017 benchmark generators. The test set consisted of a thousand AFs containing 25 arguments each. Their results when compared to Kuhlmann and Thimm (2019) were vastly superior when analyzed under the lens of the Matthews Correlation Coefficient (MCC). This measure of a classifier performance has proved more robust than the more usual measures of performance, particularly in the presence of an unbalanced number of members in each class.

The authors also claimed that their model can also be applied to larger and more complex AFs than seen during the training stage by just increasing the number of steps T .

Problems in weak admissibility semantics are hard (Dvořák et al., 2021) and the availability of working exact solvers is limited. Probably due to the relatively novelty of weak admissibility semantics, we have not seen any studies assessing the feasibility of applying ML techniques towards solving computational problems in this area. In this study we would like to address both issues. During the course of this research we design and implement an exact solver for credulous acceptability under weak admissibility semantics. The solver provide us with the necessary data to later use statistical tools and ML techniques to investigate the viability of approximate solutions with low cost.

As we have seen, two different types of ML techniques has been applied to the problem of deciding credulous acceptability under preferred semantics. The results are promising in the sense that once a model has been trained, it can perform relatively well at identifying an arguments' credulous acceptability in very short run time.

Chapter 3

Credulous Acceptability under W-Admissibility

In this chapter, we briefly discuss the two general approaches to solve problems in abstract argumentation. We also describe our own implementation as well as the caveats and assumptions that led its development.

3.1 Types of Implementation

Typically, researchers and experts have implemented solutions to computational problems in abstract argumentation using two kinds of approaches (Charwat et al., 2015; Cerutti et al., 2018):

1. reduction-based implementation, and
2. direct implementation.

Reduction-based implementations make use of sophisticated well-maintained solver tools initially designed for other problem domains. By translating a reasoning problem into an equivalent formalism appropriate to the solver in question, we can take advantage of existing software to produce outputs that will be interpreted as solutions to the original problem (Cerutti et al., 2018). This type of implementation require developers to have a solid understanding of both the formalism of abstract argumentation and the formalism of the target system.

On the other hand, direct implementations consist of software written from scratch for the specific problem at hand. They are able to incorporate certain features specific to abstract argumentation that may drive great improvement, while incorporating the same feature in a reduction-based approach may only lead to limited improvement, if at all (Cerutti et al., 2018).

Unlike classic semantics, weak admissibility semantics has not seen much developments on the implementation of tools for this particular formalism. After consulting with the proponents and specialized researchers of the theory behind

weak admissibility semantics, neither a reduction-based nor a matured direct implementation was available. One publicly available implementation of a reasoner for weakly admissibility semantics is included in the TweetyProject¹ (Thimm, 2014; Thimm, 2017) and is a direct implementation in Java developed by Lars Bengel at the Artificial Intelligence Group in Hagen.

After reviewing this implementation, I decided against incorporating it in my research for two main reasons:

1. I suspected that this implementation would only work in reasonable time for AFs containing a limited amount of arguments. This suspicion would later prove correct in our own implementation. The repeated generation of conflict-free sets even in a reduced AF requires a considerable amount of resources, slowing down the process.
2. Interfacing with this reasoner required some serious effort to first process the amount of AFs needed for the analysis and to produce an output structure conducive to further analysis and processing.

Furthermore, our research goals require that we determine the credulous acceptability status of arguments under weak admissibility semantics in a “large” number of AFs under constraints relative to time and limited computational resources. To this end, we developed a direct implementation using the ideas proposed by Dvořák et al. (2021) and Dvořák et al. (2022).

In the following sections, we will discuss an alternative statement of our problem that will contribute to the application of more refined strategies and possible improved use of computational resources.

3.2 Problem Reformulation

As stated above, our goal was to determine the credulous acceptability status under weak admissibility semantics for every argument in a number of AFs. Instinctively, we felt inclined to use some adaptation of Algorithm 2, and run it for each argument in an AF. However, this course of action would have prevented us from taking advantage of some strategies mentioned by Dvořák et al. (2022). These strategies would not be applicable to weak admissibility semantics, but taking a different perspective on the problem would allow us to use them without further ado.

There exists a conceptual link between weakly preferred extension and weakly admissible extensions, and the argument goes like this:

“For an AF $F = (A, R)$, $E \subseteq A$ is called weakly preferred (or w-pref) in F ($E \in pr^w(F)$) if and only if is \subseteq -maximal in $adm^w(F)$.” (Baumann et al., 2020a, Definition 2.7).

¹<http://tweetyproject.org/>

Because weakly preferred extensions are maximal (with respect to set inclusion) weakly admissible extensions, this implies that for any C such that $C \in \text{adm}^w(F)$ there exists some $D \in \text{pr}^w(F)$ such that $C \subseteq D$. It follows that the union of all weakly preferred extensions in any AF equals the union of all weakly admissible extensions of that AF.

In other words, let $CA_{\text{adm}^w} \subseteq A$ be the set of all credulously accepted arguments in an AF $F = (A, R)$ under weak admissibility semantics, then

$$CA_{\text{adm}^w} = \bigcup_{C \in \text{adm}^w(F)} C = \bigcup_{D \in \text{pr}^w(F)} D \quad (3.1)$$

Under this premises, we were able to restate the problem at hand, from deciding credulous acceptability of arguments under weak admissibility semantics to enumerating all weakly preferred extensions in an AF, forming the union of the weakly preferred extensions and obtaining the set CA_{adm^w} of credulously accepted arguments under weak admissibility semantics.

In summary, our main new problem is "enumerating all weakly preferred extensions in an AF". This reformulation opens up the chance of considering SCC-recursiveness as an strategy to iteratively reduce the problem size in the best cases as described in Section 3.4.

Before we move on to the thick of our problem solution, we introduce two preliminary steps to computing weakly preferred extensions. It should be noted that both preprocessing steps can also be used in the determination of other extensions under weak admissibility semantics.

3.3 Preprocessing Steps

The steps delineated in the following parts constitute an effort to reduce the problem size in order to make our solution more scalable. In the best case, these steps will greatly simplify the problem. In the worst case, they will do almost nothing for the problem size and the resources needed to solve the problem. The cost of these operations is nevertheless low and they will be carried out by default.

3.3.1 Exclusion of Self-Attacking Arguments

The first preprocessing step was initially proposed by Baumann et al. (2020b, Theorem 3.10) and guarantees that the weak admissible and weakly preferred extensions—computed with the resulting AF after this step is executed— will be consistent with the extensions computed with the original AF.

The preprocessing consists of determining all arguments with self-loops. Formally, let $F = (A, R)$ be an AF and let $L \subseteq A$ be the set containing any argument $a \in A$ such that if $(a, a) \in R$ then $a \in L$.

Once we have determined the set L of all self-attacking arguments we may proceed to remove them from F . Both the self-attacking arguments L as well as all

incoming and outgoing edges to and from every argument $a \in L$ will be banned from the updated AF F^0 .

Eliminating all self-attacking arguments from an AF will be carried out exactly once and always at the point of object initialization. The succeeding steps will always assume that the AF under examination is free of self-attacking arguments.

3.3.2 Computing the Grounded Extension

The second preprocessing step is based on two important arguments. First, we argue that computing the grounded extension will be a low cost processing step that would in some cases reduce a large AF into a manageable size.

Second, Baumann et al. (2020b, Proposition 5.11) provide the theoretical basis for the computation of the grounded extension as a step towards computing w -preferred extensions. More specifically, a by-product of the mentioned Proposition is that " w -complete extension always contain the classical grounded part." Moreover, because weakly preferred extensions are maximal (with respect to set inclusion) weakly complete extensions (Baumann et al., 2020b, Theorem 5.3), we can safely assume that weakly preferred extensions also always contain the classical grounded part.

Although, this step delivers good results for graphs containing initial nodes, in the case of highly connected graphs (for example, large graphs with only one SCC) this step will not offer a serious advantage. Nevertheless, its low computational cost justifies its inclusion in our solution as a standard step.

Notice that in this procedure we are referring to the grounded extension in the classical sense and should not be confused with its weak admissibility counterpart, the weakly grounded extension. Computing the grounded extension of a given AF $F = (A, R)$ requires the following iterative steps (Baroni et al., 2005):

1. determining all initial nodes in the graph, i.e. identifying the unattacked arguments in an AF
2. eliminating the initial nodes as well as the nodes attacked by it from F , i.e. computing the reduct of F with respect to the initial nodes.
3. apply step 1 and 2 to the resulting AF until no initial nodes are found.

Once we have computed the grounded extension according to Algorithm 4, we proceed to compute the reduct of the AF with respect to the grounded extension. This will exclude from the AF, not only the grounded extension itself but also all arguments attacked by it, leaving us, in the best case, with a smaller AF than the original. The grounded extension will then be added to all the weakly preferred extensions found in the "smaller" AF.

Unlike the first preprocessing step, this step is carried out not only at the beginning of the program but also number of times throughout the weakly preferred extension construction process, as we will see in the following section.

Algorithm 4 Generating the Grounded Extension

Require: $F = (A, R)$

```

1: procedure GENERATEGROUNDEDEXTENSION( $F$ )
2:    $gext \leftarrow \{\}$ 
3:   if  $|A| == 0$  then
4:     return  $\{\}$ 
5:   for all unattached  $arg \in A$  do
6:      $ADD(gext, arg)$ 
7:    $F^{gext} \leftarrow REDUCT(F, gext)$ 
8:    $gext \leftarrow gext \cup GENERATEGROUNDEDEXTENSION(F^{gext})$ 
9:   return  $gext$ 

```

3.4 Enumerating Weakly Preferred Extensions

In Section 2.2.3 we outlined an algorithm that would find weakly preferred extensions with the aid of SCCs. The main idea is that the graph structure drives the incremental construction of extensions (Baroni et al., 2005) as each initial SCC is processed and nodes are subsequently suppressed based on the extensions found in the SCC under examination. However, there were a few loose ends which we will describe in the paragraphs ahead. First, we will tackle the task of computing SCCs. We will revisit Algorithm 3 and define the **basis function**, then we go on and integrate the concept of the grounded extension into the wider and much general problem of generating all weakly preferred extensions of an AF. We will also discuss an adaptation of Algorithm 1 presented in Section 2.2.1 for verifying weak admissibility of a set of arguments in an AF that incorporates the ideas behind SCC-recursiveness.

3.4.1 Computing Strongly Connected Component (SCC)

We found several algorithms which compute the SCCs of a directed graph in linear time. We had the suspicion that a recursive algorithm would prove problematic in view of Python’s recursion limit, so they were discarded from the beginning. Further study lead us to assume that not all of the non-recursive algorithms would provide the best performance for any graph structure. Hsu et al. (2017) showed that Tarjan’s algorithm (Tarjan, 1971) offered superior time performance for most digraph structures irrespective of the programming language (C++, Java) in which they were implemented.

We will not enter into the details of Tarjan’s algorithm, it does not belong to the scope of this work. Suffice to say that we decided on the modified version of this algorithm proposed by Nuutila and Soisalon-Soininen (1994). The reason behind this decision was that this algorithm could process “sparse graphs and graphs containing trivial components more economically”. Particularly, in the case where we iteratively reduce the size of the original AF and more SCCs with one element

appear, this algorithm remains efficient.

3.4.2 Finding Weakly Preferred Extension with SCC-Recursiveness

We already discussed the general idea behind SCC-recursiveness. Previously, we also briefly discussed efficient algorithms to compute SCCs and presented and justified our choice. Still we are missing important pieces of the puzzle. The first one relates to the so-called “basis function” which we will address now.

Algorithm 5 Finding Weakly Preferred Extensions

Require: $F = (A, R)$

```

1: procedure FINDWPPREFERREDEXTENSIONS( $F$ )
2:    $wprefext \leftarrow \{\}$ 
3:    $SCCS \leftarrow \text{STRONGLYCONNECTEDCOMPONENTS}(F)$ 
4:   for all initial  $scc \in SCCS$  do
5:      $cfsetssc \leftarrow \text{GENERATECONFLICTFREESETS}(scc)$ 
6:      $wpref \leftarrow \text{False}$ 
7:     for all  $cfset \in cfsetssc$  do
8:       if ISWADMISSIBLE( $scc, cfset$ ) then
9:          $wpref \leftarrow \text{True}$ 
10:         $F^{cfset} \leftarrow \text{REDUCT}(F, cfset)$ 
11:         $otherwprefsets \leftarrow \text{GENERATEWPPREFERREDEXTENSIONS}(F^{cfset})$ 
12:        for all  $otherset \in otherwprefsets$  do
13:           $\text{ADD}(wprefext, (cfset \cup otherset))$ 
14:       if  $wpref \neq \text{True}$  then
15:          $F \downarrow_{A \setminus scc} \leftarrow \text{REMOVE}(F, scc)$ 
16:          $otherwprefsets \leftarrow \text{GENERATEWPPREFERREDEXTENSIONS}(F \downarrow_{A \setminus scc})$ 
17:         for all  $otherset \in otherwprefsets$  do
18:            $\text{ADD}(wprefext, otherset)$ 
19:   return  $wprefext$ 

```

Algorithm 5 outlines a fine-grained version of Algorithm 3. After we have isolated one of the initial SCCs from the rest of the AF we apply the basis function. The SCC has no incoming edges, so we only need to cut loose the outgoing edges. By definition, for each of the elements in an SCC there exists a path from any of the other elements of the SCC, so we can safely assume that at this point, there will be no initial nodes on which we could profit from computing the grounded extension.

The basis function (highlighted in Algorithm 5) applied to any initial SCC under consideration consists of generating all conflict-free sets in the SCC and testing each one of them for weak admissibility. We introduce a flag indicating the existence or non-existence of a weakly admissible extension in the SCC.

In case an extension was found, we compute the reduct of the original AF with respect to the weakly admissible extension found, and continue to generate exten-

sions in the reduced AF that will later be joined together to form a preferred extension, in accordance with Dvořák et al. (2022, Theorem 5.8). After computing the reduct of the AF, we may safely assume that our new AF will contain some initial nodes. This is why instead of recursively applying the FINDWPPREFERREDEXTENSIONS procedure we use instead a generating procedure. This procedure will be explained in the next section.

In case no weakly admissible extensions were found in the SCC, we completely remove the SCC from the AF, all nodes and outgoing edges are removed from the AF. Next, we continue to generate weakly preferred extensions from the resulting restriction of AF. At this point, it is safe to assume there will be some initial nodes in the restriction.

The second missing piece of the puzzle relates to the integration of the grounded extension to the overall w-preferred extension generation process. It is time to direct our attention to this piece.

3.4.3 Generating Weakly Preferred Extensions

The last building block in our solution consists of integrating the grounded extension and with the w-preferred extensions found using SCC-recursiveness.

The first step in this procedure corresponds to computing the grounded extension as explained in 3.3.2. Once we obtain the grounded extension, we determine the reduct of the AF with respect to the grounded extension and proceed to generate the weakly preferred extensions in the reduced AF. If we have found weakly preferred extensions in the reduced AF we are ready to add to each one of them the grounded extension and return all w-preferred extensions of the AF.

Algorithm 6 Generating Weakly Preferred Extensions

Require: $F = (A, R)$

```

1: procedure GENERATEWPPREFERREDEXTENSIONS( $F$ )
2:    $wprefexts \leftarrow \{\}$ 
3:    $gext \leftarrow \text{GENERATEGROUNDEDEXTENSION}(F)$ 
4:    $F^{gext} \leftarrow \text{REDUCT}(F, gext)$ 
5:    $auxwprefexts \leftarrow \text{FINDWPPREFERREDEXTENSIONS}(F^{gext})$ 
6:   for all  $auxwpe \in auxwprefexts$  do
7:      $\text{ADD}(wprefexts, (auxwpe \cup gext))$ 
8:   output  $wprefexts$ 

```

In performing the steps depicted in Algorithm 6 we are closer to the solution. But it is time to take a step backwards and address the final missing piece of the puzzle. We are speaking about verifying weak admissibility of a set of arguments in an AF taking into account the procedures discussed previously.

3.4.4 Verifying weak admissibility of a set of arguments

We are now ready to present a modified version of Algorithm 1 that incorporates the concept of the grounded extension along with some of the functions and procedures presented in previous sections.

Our modified version of Algorithm 1 includes a series of shortcuts derived from the theory behind weak admissibility semantics. The function requires two arguments: an AF $F = (A, R)$ and a subset of the arguments of the AF $S \subseteq A$. It is for the subset S that we wish to determine whether it is weakly admissible. The procedure returns true if the subset is weakly admissible in the AF, and false otherwise.

Algorithm 7 Verifying Weak Admissibility

Require: $F = (A, R), S \subseteq A$

```

1: procedure ISWADMISSIBLE( $F, S$ )
2:   if  $S == \emptyset$  then
3:     return True
4:   if NOT ISCONFLICTFREE( $F, S$ ) then
5:     return False
6:    $att(S) \leftarrow$  ATTACKERS( $S, F$ )
7:   if  $att(S) == \emptyset$  then
8:     return True
9:    $F^S \leftarrow$  REDUCT( $F, S$ )
10:  if ( $Args(F^S) \cap att(S)$ )  $== \emptyset$  then
11:    return True
12:   $gext \leftarrow$  GENERATEGROUNDEDEXTENSION( $F^S$ )
13:  if ( $gext \cap att(S)$ )  $!= \emptyset$  then
14:    return False
15:   $auxF \leftarrow$  REDUCT( $F^S, gext$ )
16:  if ( $Args(auxF) \cap att(S)$ )  $!= \emptyset$  then
17:     $wprefexts \leftarrow$  GENERATEWPREFERREDEXTENSIONS( $auxF$ )
18:    for all  $ext \in wprefexts$  do
19:      if  $ext \cap att(S)$   $!= \emptyset$  then
20:        return False
21:  return True

```

As mentioned above, we introduce some steps that quickly help us determine weak admissibility. The first shortcut derives from the axiom “the empty set is always weakly admissible” (Dvořák et al., 2022, Proposition 4.7), in case our subset $S \subseteq A$ were empty, we can immediately assert that it is w-admissible. The second shortcut can be inferred from the definition of weak admissibility given in Baumann et al. (2020b, Definition 3.3) and was already part of Algorithm 1. A set S which does not conform to conflict-freeness can be immediately deemed not w-admissible. If the set S under examination passes the conditions posted above, we go on to create another set containing all of the arguments that attack S , this set we call $att(S)$. If

the set $att(S)$ is empty, meaning S is unattacked in F , it immediately follows that S is w -admissible in F .

At this point is where things start getting interesting, S is not empty, it is conflict-free and there are arguments in the AF attacking it. We get to the part where the reduct of F with respect to S is computed as suggested by Dvořák et al. (2021), resulting in a new AF F^S in which the arguments in S are excluded as well as all the arguments attacked by the arguments in S . In case the reduct does not contain arguments attacking S , the second condition in Baumann et al. (2020b, Definition 3.3) is met and weak admissibility can be ascribed to the set S . Should the opposite be the case and there are arguments attacking S in the reduct we call for the computation of the grounded extension in the classical sense as described in Section 3.3.2. This step finds its justification in Baumann et al. (2020b, Proposition 5.11) and it tells us that any argument attacking S that is also a member of the grounded extension of the reduct of F will automatically render S non-weakly admissible.

If after all the previous steps, the w -admissibility of S could not be determined, we reach a stage where the heavy computational legwork must be carried out. We have already computed the grounded extension of the reduct F^S and made sure that none of the arguments that attack S are in it. Remember the grounded extension is always contained in the weakly complete extensions and also in the weakly preferred extensions. In addition, as we established in Section 3.2, credulous acceptability under weakly admissible semantics is equivalent to credulous acceptability under weakly preferred semantics. Besides, the second condition for weak admissibility (Baumann et al., 2020b, Definition 3.3.) can be interpreted as the following statement:

For any attacker y of S , we have that y is not credulously accepted under weakly preferred semantics in the AF consisting of the reduct of the original AF F with respect to S , denoted by F^S .

We are not really interested in enumerating all weakly preferred extensions in the reduct F^S of the original AF F , what we intend instead is to reduce the problem at hand as much as possible and to find out whether any of the remaining attackers of S is credulously accepted under weakly preferred semantics in the AF F^S . We compute the reduct of F^S with respect to the grounded extension, denoted by $auxF$, to try to further decrease the size the problem.

As long as the new AF $auxF$ still contains arguments attacking S , we go on to generate all weakly preferred extensions of $auxF$ and check whether any of these w -preferred extensions contains an argument attacking S . As soon as the existence of an argument attacking S in a w -preferred extension is established, we can say that S is not weakly admissible in F and the procedure returns *False*. Otherwise, the procedure ends when all weakly preferred extensions of $auxF$ have been examined and no extension contains an argument attacking S , in which case, we say that S is w -admissible in F , returning *True*.

3.5 Solver Design

We chose Python 3.8 as the programming language throughout our research. This decision was made due mainly to the availability of a number of tools, libraries and frameworks that would facilitate not only the development of the solver, but would later play an important role in the analysis and processing of the generated data.

Our solver was conceived with an object-oriented perspective. Our solution instantiates an object of the `DiGraph` class defined in the Python package `NetworX` Release 2.8.8 (Hagberg et al., 2008) to characterize an AF.

Before undergoing actual processing the `DiGraph` first gets stripped off of all nodes with self-loops and their corresponding incoming and outgoing edges (see Section 3.3.1 for more information).

Out of the resulting `DiGraph` object, three data structures are extracted. The first one is a list consisting of the nodes of the `DiGraph`. Additionally, two dictionary structures are created. The keys of these two dictionaries are formed by the nodes contained in the graph unburdened of self-loops. While the first dictionary contains, for each node, the set of all predecessors of that node, the values of the second dictionary are constituted by all the successors of that node.

Most of the operations in our software involve manipulation of the three data structures mentioned above in the form of set operations such as union, intersection and difference. The solver returns the set of all weakly preferred extensions of an AF.

3.6 Experimental Evaluation

Our solver was evaluated on an Apple M1 processor with 8 GB RAM on two sets of AFs. The first set consisted of the AFs depicted in the examples presented by Dvořák et al., 2022. The second set AFs was compiled from the ICCMA website and consisted of the published benchmarks of the competition in the years 2017² and 2019³.

The first set of AFs was mainly used to validate correctness. There were a total of 14 AFs conforming this set; each AF contained between 3 to 10 arguments and between 3 and 15 attacks. The advantage of choosing this set was that the weakly preferred extensions could be computed by hand and allowed us to perform “sanity checks” at different stages of development.

The second sample set provided us with a way to test our solver more broadly with more challenging AFs, in terms of size and structure complexity. This sample originally consisted of 1,388 AFs. Some AFs were listed more than once in this dataset and we proceeded to remove duplicates from our database ending up with 908 unique AFs, see Table 3.1 for a statistical description of this sample in terms of number of nodes and edges in each AF. On further examination of the data, we

²<https://www.argumentationcompetition.org/2017/results.html>

³<https://www.argumentationcompetition.org/2019/results.html>

Argumentation Framework Composition (n = 908)		
	Arguments	Attacks
Mean	10,515.45	269,369.11
Standard Deviation	122,965.25	749,895.60
25th percentile	161.00	599.00
50th percentile	499.00	5,849.00
75th percentile	1,471.75	48,637.00
Minimum	2.00	1.00
Maximum	2,500,000.00	6,257,500.00
Skewness	16.46	4.14
Kurtosis	388.33	21.09

TABLE 3.1: ICCMA Sample

realized that the set of AFs used in the 2019 competition were essentially a subset of the one used in the 2017 edition.

From this data, we can immediately notice the marked difference between the sample mean and the 50th percentile which indicates a positively skewed distribution on both the number of arguments and the number of attacks per AF. More specifically, our ICCMA sample contains a great amount of AFs with a number of arguments between 2 and 1,400 and a few outliers with a very large number of arguments. The skewness was estimated to be 16.46 which confirms that our data set is heavily skewed to the right. The same can be said about the number of attacks in the AFs in this case the skewness was estimated at 4.14 and is a bit less skewed than the number of arguments.

Furthermore, given the size and complexity of our sample, and because we suspect our solver is still in need of further improvement, we limited the execution time on each member of the sample to the arbitrarily chosen limit of ten minutes. If after ten minutes computation the solver had not returned a solution, the computation was aborted and the solver moved on to the next AF. The rationale behind this choice was made on the “back of the envelope” calculation that if on average each AF would take 600 seconds to process, we would need around six to seven days (night and day) to process the whole sample. In the end, we were able to obtain the w-preferred extensions for 247 AFs. This result amounts to less than a third of the sample size and might be an indicator that the ten minute average processing time was just too optimistic, since we were expecting to obtain w-preferred extensions for at least half of the sample. However, processing time is not linear and the problem is still very hard to solve.

One important observation we want to point out is that imposing a run-time limit could be a cause of bias in the data. More specifically, because our sample is composed of elements which are “easily” available, our solved sample may not be a representative sample of the ICCMA dataset. As we can see from Table 3.2, the mean of the number of arguments of the solved sample differs from the mean of the

Solved Argumentation Framework Composition (n = 247)

	Arguments	Attacks
Mean	1,759.01	376,877.18
Standard Deviation	3,438.20	867,047.49
25th percentile	25.00	97.00
50th percentile	121.00	801.00
75th percentile	1,203.50	359,955.50
Minimum	2.00	1.00
Maximum	14,665.00	5,378,385.00
Skewness	2.42	3.83
Kurtosis	5.07	17.02

TABLE 3.2: ICCMA Solved Sample Description

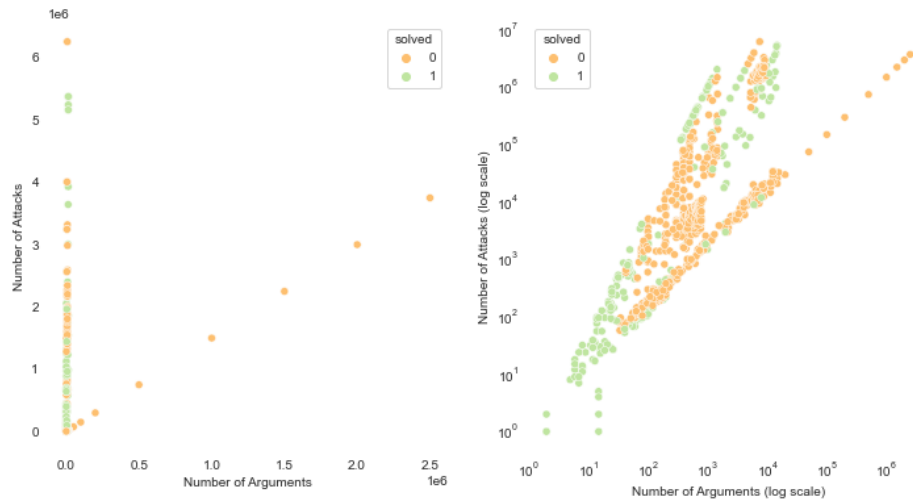


FIGURE 3.1: Composition of the ICCMA Sample

Dataset Composition			
Graph Type	Acyclic	Bipartite	Symmetric
Original ICCMA Dataset (n = 908)	0.03%	0.14%	0.03%
Solved ICCMA Dataset (n = 247)	0.06%	0.24%	0.10%

TABLE 3.3: Graph Types in Dataset

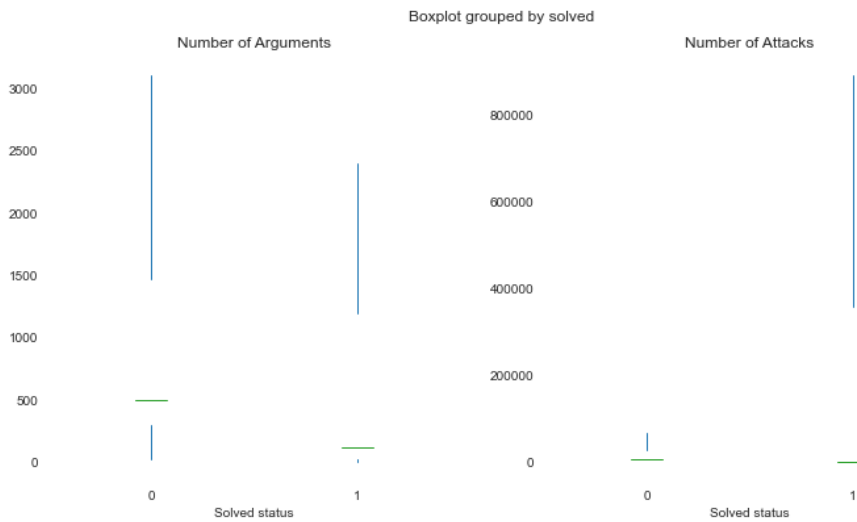


FIGURE 3.2: Differences between solved and unsolved AFs in the ICCMA sample

total sample by one degree of magnitude, amounting to a skewness coefficient of 2.42. Notice how the distribution of the number of arguments in the solved sample is still positively skewed but not as much as in the original sample. Regarding the number of attacks, we observed a similar effect, the mean in the solved sample is larger than the mean in the original dataset. In this case, the distribution of the number of attacks in the solved dataset is also skewed to the right by an estimated 3.83, slightly less skewed than the original sample. Figure 3.1 offers a depiction of how the solved set differs in distribution with respect to the unsolved set in both the linear and the logarithmic scale.

In particular, the logarithmic scale provides a more nuanced picture of how solved and unsolved AFs are distributed with respect to the number of attacks and the number of arguments.

The solved sample might also differ in other important ways from the original ICCMA sample, the structure of the graphs plays an important role how “easy” it is to compute its weakly preferred extensions, see Figure 3.2. For example, Baumann et al. (2020a) and Dvořák et al. (2022) claim that computations are easier for acyclic, symmetric and bipartite graphs. From Table 3.3 we are able to distinguish a larger

Run-time Summary (n = 247)	
Mean	18.033485 s
Standard Deviation	70.911677
25th percentile	0.004248 s
50th percentile	0.437006 s
75th percentile	3.852466 s
Minimum	0.000032 s
Maximum	538.549455 s

TABLE 3.4: Solved Dataset Run-Time

presence of these types of graphs in the solved dataset as compared to the original dataset.

Notwithstanding the biases caused by the ten minute run-time limit, we will be implementing this policy for the rest of the study. The reasons are of a mere practical nature: we need as many exact solutions for as many AFs in reasonable time to attain the goals of the study.

A closer look at the run-time also revealed a positively skewed distribution with large differences between the fastest computed AF and the slowest. The majority of AFs, around three quarters of the sample, could be processed in under four seconds with some outliers requiring more time.

3.7 Limitations

Particularly, while testing for weak admissibility in order to generate weakly preferred extensions, our implementation will compute branches in the search space more than once leading to a waste of computational resources. We were not able to devise a mechanism which would do away with this issue, so we will leave it to future research to deal with this problem.

We are also well aware that a rigorous evaluation of our solver should have included a thorough comparison against other algorithms or existing tools. In this case, we should have compared our tool against the Java implementation by Lars Bengel, which at the time of this writing was the only publicly available tool. Or we could have programmed a solution using the algorithms in Dvořák et al. (2021).

Due to time constraints we will leave the above mentioned tasks open for future work. We will also assume that our implementation delivers acceptable performance in terms of run time and considering the number of AFs we will evaluate. Therefore, making it the default option for the rest of this work.

Chapter 4

Exploratory Data Analysis

In the previous chapter we implemented an exact solver for weakly preferred semantics. This tool, despite sub-optimality, will help us determine the ground truth for the credulous acceptability of the arguments in a number of AFs. The ability to determine this information is essential in the pursue of the other goals of this study.

The next step consists of putting together a sample set of AFs with reliable information about the credulous acceptability of their arguments under weak admissibility semantics. Because in this work we deal exclusively with credulous acceptability under weak admissibility semantics, in some instances, we refer to it as just “credulous acceptability” for short.

In the following sections, we delineate the process of collecting a “large” sample set. We explain the relation between the data collected and the goals of the study, we then proceed to statistically analyze their features at the AF-level as well as at the argument level.

Ultimately, the goal of this step of the study is be, with the help of statistics, identify a set of predictor variables of credulous acceptability under weak admissibility semantics and learn patterns that influence the acceptability status of arguments in an AF.

4.1 Data Collection

The first step in our workflow consists of assembling a sample set or data set of AFs and its corresponding arguments. This step will also be carried out in preparation for later steps.

In section 3.6 we explored the capabilities of our solver on the ICCMA Benchmark for the years 2017 and 2019. A closer look at the set revealed duplicated AFs which were eliminated from the sample. We were able to compute the ground truth for credulous acceptability under weak admissibility semantics for less than a third of the unique AFs in the sample, subject to a time limit. Although not at all discouraging, ML techniques are primarily based on the premise of a large sample.

In contrast, in statistics a common rule-of-thumb is to think of a large enough sample size as one that contains at least 10 observations for each feature under examination. Should we adopt the statisticians approach, our sample with 247 AFs would be more than enough for analysis. However, from the point of view of the data scientist, this would be considered a meager sample. Mostly the sample size in ML depends largely on the complexity of the problem and accuracy requirements of the model. A non-linear model trained with a large sample yields, in general, better accuracy than the same model trained with a smaller sample (Kuhlmann et al., 2022).

Similar exploratory studies (Kuhlmann and Thimm, 2019; Craandijk and Bex, 2020; Malmqvist et al., 2020) have used very different sample sizes for their analysis. Their sample sizes range from nine hundred to over one million AFs. Already Kuhlmann et al. (2022) have discussed the discrepancy in sample sizes and sample composition of previous studies. Nevertheless, we are probably constrained by a sub-optimal ground truth solver, limited computational resources and limited time. Thus, based on previous estimations we conclude that around 3,000 AFs would constitute a handsome feasible sample size to conduct a sound analysis.

Next, we have to determine the sources and composition of the 3,000 AFs needed. Not only do we need to generate this number of AFs but we also need to be able to solve them almost surely within ten minutes, so as to comply with time constraints imposed by the nature of this work. Considering the performance of our solver on the ICCMA sample, we conclude that AFs consisting of 5 to 55 arguments would be a challenge our solver could manage. For matters of convenience, we use the AF-generator suite implemented by Craandijk and Bex (2020)¹. This generator suite includes some of the benchmarks defined at the ICCMA 2017, namely *AFBenchGen2*, *AFGenBenchGen*, *Grounded Generator*, *ScGenerator* and *StableGenerator* (Gaggl et al., 2020). These benchmarks have received wide acceptance from the expert community.

In the following section we will present a description of the AF sample generated using the above mentioned tool.

4.1.1 AF Sample Description

We were able to generate 4,615 unique AFs using the generators mentioned in the previous section. The generated AFs consisted, as planned, of between five and fifty-five arguments.

Out of the generated AF sample we were able to solve 3,307 instances in under ten minutes. As we have previously explained, the ten minute run-time influences the composition of the final experimental sample. Table 4.1 illustrates this assumption. The average number of arguments in the total sample is slightly greater than the average in the solved sample. Both samples are nearly symmetric, a conclusion we reach by comparing the mean to the 50th percentile, also called the median, and

¹<https://github.com/DennisCraandijk/DL-abstract-argumentation/>

	Number of Arguments	
	Total Sample (n = 4,615)	Solved Sample (n = 3,307)
Mean	33.81	30.25
Standard Deviation	13.85	14.03
25th percentile	23.00	18.00
50th percentile	35.00	28.00
75th percentile	45.00	43.00
Minimum	5.00	5.00
Maximum	55.00	55.00
Skewness	-0.17	0.20
Kurtosis	-1.09	-1.08

TABLE 4.1: Number of arguments in the Generated AF Sample

	Number of Attacks	
	Total Sample (n = 4615)	Solved Sample (n = 3,307)
Mean	185.56	138.35
Standard Deviation	199.61	188.68
25th percentile	47.00	36.00
50th percentile	113.00	63.00
75th percentile	260.00	154.00
Minimum	4.00	4.00
Maximum	1,529.00	1,529.00
Skewness	2.11	3.08
Kurtosis	6.07	12.07

TABLE 4.2: Number of attacks in the generated AF sample

finding no big differences in either sample. Also by examining the skewness coefficient of each sample, we notice they are both rather small, with the total sample being slightly negatively skewed as compared to the solved sample which is positively skewed. The negative kurtosis value for both samples tells us, in part as we expected, that both samples, when compared to a normal distribution, have a flattened peak and rather thin tails. We can trace this phenomena back to the fact that when the sample was generated, the number of arguments in each AF is uniformly distributed, meaning that each number in the range from five up to 55 has the same probability of being chosen as the number of arguments in the generated AF.

The distribution of the number of attacks in the generated sample is positively skewed and the skewness only gets bigger in the solved sample, as we observe in Table 4.2. The solved sample has more outliers with respect to the total generated sample, and we find a greater concentration of unsolved AFs in the upper half of the distribution of the number of attacks, as shown in Figure 4.1, solved and unsolved

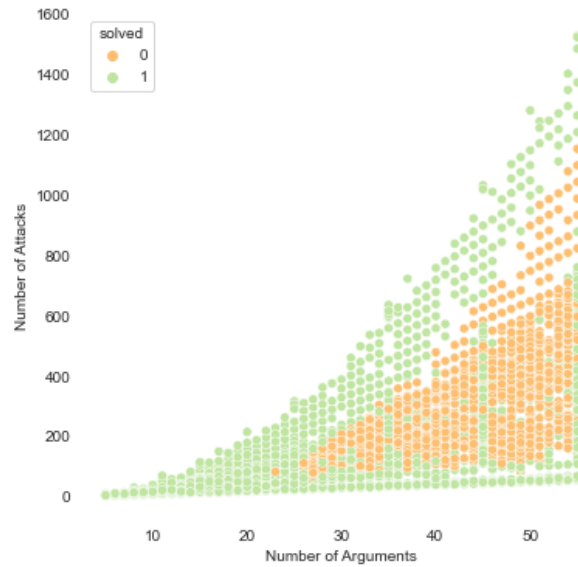


FIGURE 4.1: Composition of the Generated AFs

AFs are identified by the label "1" and "0", respectively. More specifically, AFs with 154 attacks or less constitute almost three quarters of the solved sample. Regarding the number of attacks, it can be said that the solved sample contains smaller AFs.

Comparing the solved set with the unsolved set, we reach the same conclusion as above, the AFs which could be solved within ten minutes are, in general, smaller than those whose solution would have taken longer to compute. In Figure 4.2, we notice a bias towards AFs with smaller amounts of arguments, but the bias is larger when we consider the number of attacks in both unsolved and solved set. Clearly, the solved set tends to contain more AFs with a smaller number of attacks.

So far, we have generated and solved a desirable amount of AFs. We have acknowledged the different ways in which our data may be biased. In spite of the sampling biases, we will continue with our study of the solved data set. In the following sections, we have a closer look at the arguments in the solved sample. We deal with two questions:

1. What are the properties of the arguments in the sample?
2. How is credulous acceptability affected by the properties of the arguments?

4.1.2 Experimental Data Set

To answer the questions posed in the previous and in posterior sections, we do not work with the entire solved data set at once. In preparation for the application of ML techniques, we partition the solved data set by randomly assigning each AF into

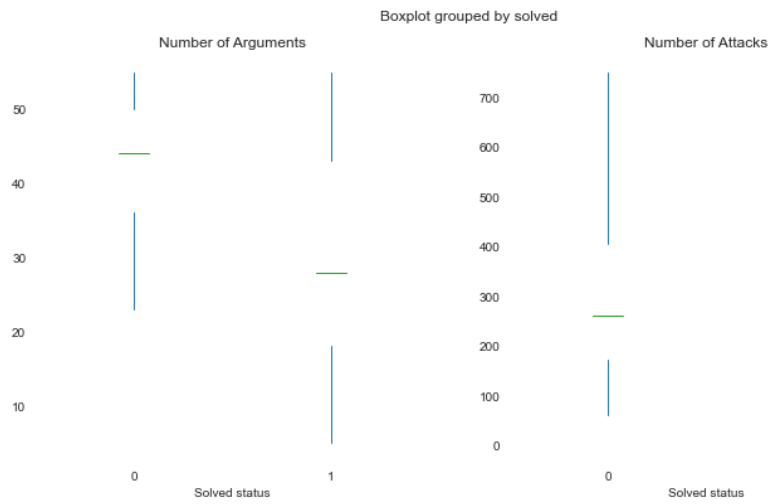


FIGURE 4.2: Differences between the unsolved and the solved AFs

Partition	Proportion	Number of AFs
Train Set	60%	1,984
Validation Set	10%	331
Test Set	30%	992

TABLE 4.3: Partitions of the solved AF Sample

one of three buckets without replacement, each bucket is assigned a different size, as depicted in Table 4.3.

The random assignment without replacement of AFs into sub-sets, i.e. an AF is assigned to one and only one sub-set, bestows each sub-set with representativity with respect to the population of solved AFs. For the purposes of statistical analysis carried out in this chapter, we focus only on the train set, this set be also referred to as the experimental set. The other two partitions will be ignored for the time being, until their reappearance at the end of this chapter and again in Chapter 5.

4.2 Argument-level Analysis

The arguments in an AF are conceptualized as atomic entities, only the attack relation with other arguments in the host AF is relevant to its study (Dung, 1995). With this idea in mind and the characterization of AFs as directed graphs, we collected four quantitative properties of arguments represented as node in a directed graph:

- in-degree, or number of incoming attacks
- out-degree, or number of outgoing attacks
- number of nodes in the graph, or total number of arguments in the host AF
- number of edges in the graph, or total number of attacks in the host AF

The properties mentioned above were accompanied by two additional pieces of information:

- whether the argument was self-attacking, and
- whether the argument in the AF was credulously accepted under w-admissibility semantics.

The properties mentioned above were organized in tabular form to allow for the use of statistical procedures. Each argument was characterized in terms of these properties or features and the exploratory analysis is presented in the next section.

4.2.1 Exploratory Analysis

Consolidating the arguments of the 1,984 AFs in the training set resulted in an argument sample of nearly sixty thousand arguments. Table 4.4 summarizes measures of location and dispersion of our sample.

For now we will focus on the in-degree and the out-degree features of the arguments. Three quarters of the arguments in the sample have up to 6 incoming attacks or are the sources of outgoing attacks. Half the arguments in the sample have 3 or less incoming or outgoing attacks. We observe some extreme values for both measures at the upper extreme of the scala, i.e. arguments that are attacked by 35 other

Arguments in Experimental Data Set (n = 59,983)				
	In-degree	Out-degree	Arguments in AF	Attacks in AF
Mean	4.55	4.55	36.82	182.42
Standard Deviation	5.28	5.10	13.18	224.58
25th percentile	1.00	1.00	25.00	49.00
50th percentile	3.00	3.00	38.00	81.00
75th percentile	6.00	6.00	48.00	241.00
Minimum	0.00	0.00	5.00	4.00
Maximum	35.00	37.00	55.00	1487.00
Skewness	1.88	1.95	-0.30	2.48
Kurtosis	3.65	4.14	-1.01	7.29

TABLE 4.4: Argument Properties Summary

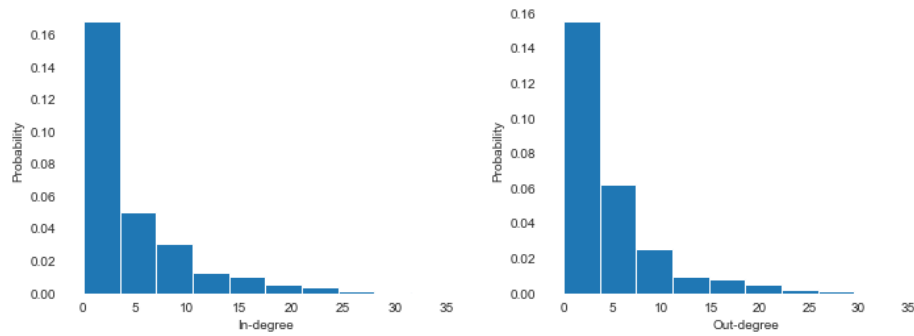


FIGURE 4.3: In- and Out-degree Histogram

	Credulously Accepted	
	No	Yes
Self-attacking		
No	55.74%	40.42%
Yes	3.84%	0.00%

TABLE 4.5: Self-attacking and Credulously accepted arguments

Arguments in Experimental Data Set (n = 57,681)		
	In-degree	Out-degree
Mean	4.47	4.41
Standard Deviation	5.27	5.02
25th percentile	1.00	1.00
50th percentile	2.00	3.00
75th percentile	6.00	6.00
Minimum	0.00	0.00
Maximum	34.00	37.00
Skewness	1.90	2.01
Kurtosis	3.65	4.45

TABLE 4.6: Data set summary: excluding self-attacking arguments

arguments. We also notice both distributions are positively skewed by observing how the sample mean differs from the 50th percentile or median. The sample mean is larger than the median, thus we can say the distributions are non-symmetric because they have longer right tails, a phenomenon we can graphically appreciate from Figure 4.3. We can conclude that the distribution of the features in-degree and out-degree is nonnormal, they do not have a bell-shaped distribution. They are also non-symmetric and have heavier tails than the normal distribution (kurtosis is in both cases larger than 3).

We continue our analysis of the argument sample with the additional properties of an argument: whether it is a self-attacking argument and whether it is credulously accepted under weak admissibility semantics. Both features were coded as boolean values, and their relative frequencies are presented in Table 4.5. Self-attacking arguments make up for nearly four percent of the arguments in our sample. From the definition of weak admissibility, we can tell that none of them will be a member in a weak admissible set, as observed in the table. Because these arguments are never weakly admissible, we can exclude them from further analysis.

The resulting data set containing no self-attacking arguments keeps compliance with some of the observations made above relative to their distribution, with one exception in the in-degree dimension: the mean and the median drift further apart, as can be observed from Table 4.6, an indicator of an asymmetric distribution. Other than that, the distribution of both features continues to be non-symmetric, has heav-

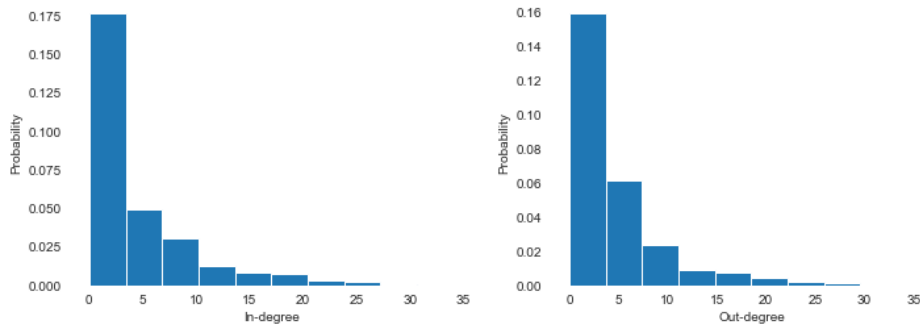


FIGURE 4.4: Histogram in- and out-degree excluding self-attacking arguments

	$CD-adm^w$
In-degree	-0.53
Out-degree	-0.17
Arguments in AF	-0.01
Attacks in AF	-0.24

TABLE 4.7: Spearman correlation coefficient of argument features

ier tails than a normal distribution and is definitely not bell-shaped (Figure 4.4), pointing out towards a non-normal distribution.

Letting aside single distributions, we want to explore how the credulously accepted arguments differ from the non-accepted arguments and Figure 4.5 provides us with some confirmation that in some of the argument features, there are differences between the non-credulously accepted arguments (grouped under the label 0.0) and the credulously accepted arguments (grouped under the label 1.0).

We are also interested in exploring how strong is the relationship between the argument features and credulous acceptability under weak admissibility semantics, denoted by $CD-adm^w$ for short. That is to say, we want to quantify the linear relationship between the features collected and credulous acceptability. The Spearman correlation coefficient offers a robust measure of this relationship in the presence of outliers. The outliers in most of the features can be observed in Figure 4.5 and are indicated as small circles. The correlation coefficients are listed in Table 4.7 and suggest a negative relation between the in-degree and credulous acceptability under weak admissibility semantics, i.e. as the in-degree is larger, the possibility of credulous acceptability decreases. The same can be said for the out-degree and the number of attacks in the AF but their influence is less strong. On the other hand, the number of arguments in the AF does not seem to exercise influence on the credulous acceptability of an argument.

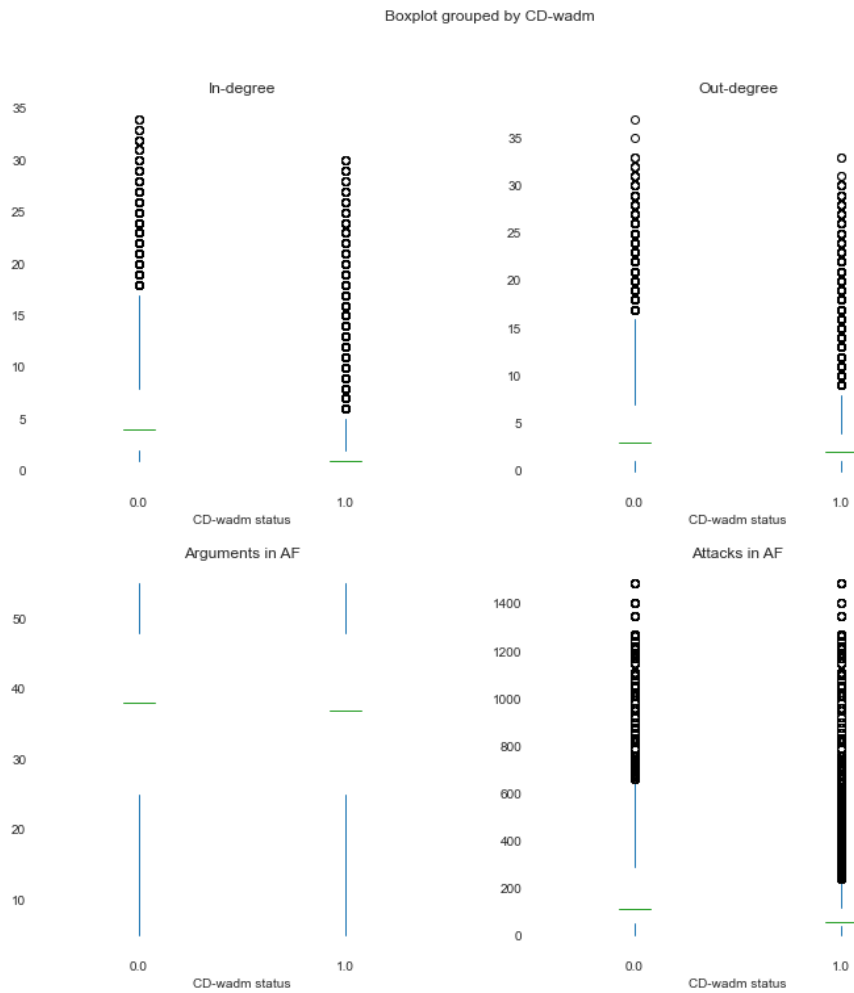


FIGURE 4.5: Differences between non-accepted and accepted arguments

The analysis carried out has helped us to generate some insight into credulous acceptability and how it is affected by the features we have collected on each argument. Sometimes, it is also useful to add new features that are functions of existing features. This activity is part of what statisticians and data scientist call “data transformation”. We delineate this process in the next section.

4.2.2 Data Transformation

So far, we have postponed the in-depth discussion of the number of arguments and attacks in the host AF as features describing an argument. We consider that incorporating them directly as a feature may not convey as much information as incorporating them in the form the denominator in a ratio with the in-degree and the out-degree as numerators.

The rationale behind the incorporation of the number of arguments and attacks into a ratio stems from a rather intuitive idea. Let $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$ be two AFs and suppose the number of arguments in F_1 and F_2 is denoted by $|A_1|$ and $|A_2|$, respectively. The number of attacks is denoted by $|R_1|$ and $|R_2|$, respectively. Let a and b be two arguments such that $a \in A_1$ and $b \in A_2$. We denote the in-degree and out-degree of an argument a as $in_degree(a)$ and $out_degree(a)$, respectively. Moreover, let $|A_1| = 3$ and $|A_2| = 30$, $|R_1| = 3$ and $|R_2| = 30$, and suppose $in_degree(a) = in_degree(b) = 2$ and $out_degree(a) = out_degree(b) = 1$. We see that in terms of in-degree and out-degree, both arguments are similar, they differ on the number of arguments and attacks of the AF in which they live. To be able to better describe an argument a , we propose, for example, that the proportion of arguments attacking it with respect to the total number of arguments in the AF, is an informative feature of the surroundings in which a lives. In this manner, the ratio in-degree to number of arguments for a would be $in_degree_to_num_args_af(a) = \frac{in_degree}{|A_1|} = 1 \div 3 = 0.3333$ and for b $in_degree_to_num_args_af(b) = \frac{in_degree}{|A_2|} = 1 \div 30 = 0.033$. The extra feature offers a more differentiated description of both arguments.

Due to the considerations exposed above we compute the ratio of the in-degree and the out-degree with respect to both the number of arguments and the number of attacks in the respective AF. From here on, the number of arguments and attacks will be excluded from the analysis in favor of the four ratios defined roughly as follows:

$$\text{In-degree to arguments ratio} = \frac{in_degree}{\text{NumberofArgumentsinAF}}$$

$$\text{In-degree to attacks ratio} = \frac{in_degree}{\text{NumberofAttacksinAF}}$$

$$\text{Out-degree to arguments ratio} = \frac{out_degr}{\text{NumberofArgumentsinAF}}$$

$$\text{Out-degree to attacks ratio} = \frac{out_degree}{\text{NumberofAttacksinAF}}$$

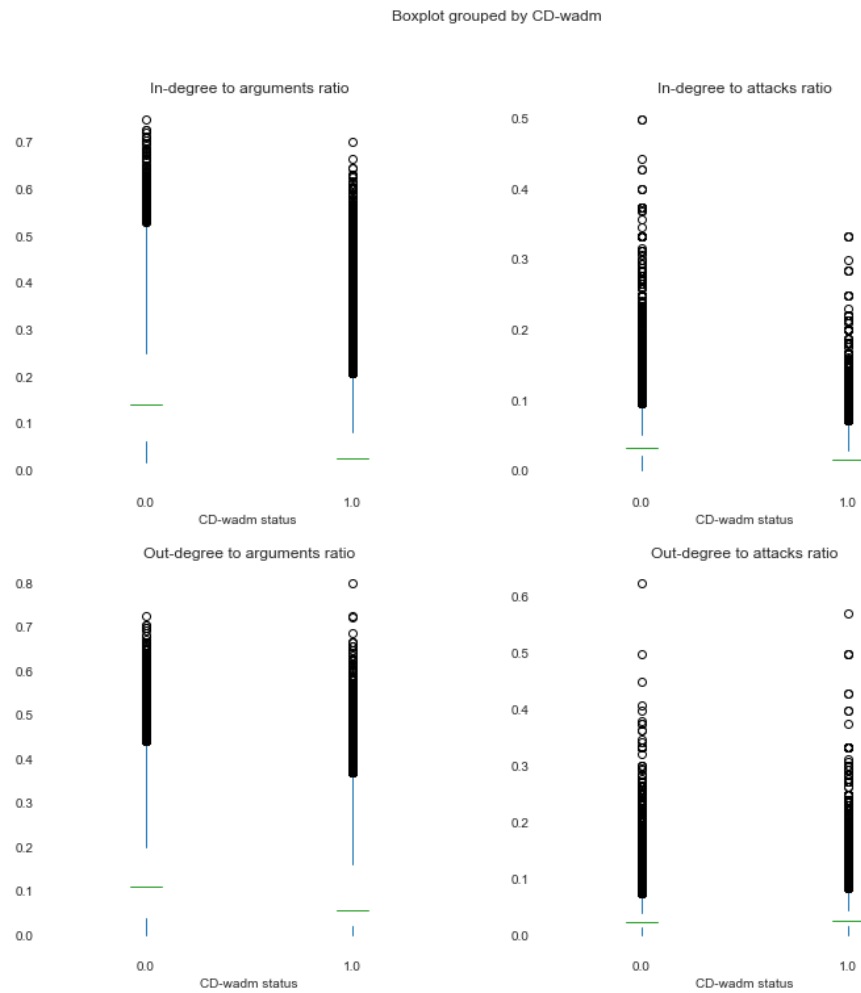


FIGURE 4.6: Differences between credulously accepted status: ratio features

Ratios	CD- adm^w
In-degree to arguments	-0.51
In-degree to attacks	-0.45
Out-degree to arguments	-0.15
Out-degree to attacks	0.08

TABLE 4.8: Spearman correlation coefficient of argument ratio features

Figure 4.6 provides visual insights of how credulously accepted arguments differ from non-accepted arguments in terms of the new computed features. For all new dimensions, we can see differences in their distribution indicative that they may have some prediction power on acceptability.

It is important to gain an idea of how strong is the relationship between the new features and credulous acceptability. We obtain this measure of relationship by means of the Spearman correlation coefficient, as shown in Table 4.8. We observe a moderate negative correlation between the in-degree ratios and credulous acceptability. Whereas the relationship between the out-degree ratios and acceptability tends to be rather weak.

As a consequence of the results above, we decide to keep the new ratio features for further analysis and discard the number of arguments and attacks as predictive features of credulous acceptability. Our next step is to explore whether the selected properties of arguments have prediction power concerning the determination of credulous acceptability under weak admissibility semantics. We are also interested in how much prediction power do the selected properties possess when deciding whether an argument is credulously accepted or not. The following section aims at answering both questions with the help of a technique called logistic regression.

4.3 Logistic Regression

With the intention to continue to model the influence of the argument features on credulous acceptability, we choose logistic regression.

Logistic regression aims at describing and testing relationships between a categorical variable—in our case the dichotomy between non-credulous acceptability and credulous acceptability—and the argument features (Peng et al., 2002). Other regression techniques fall out of consideration due to their strict requirements regarding statistical assumptions of normality, linearity and continuity.

Running a logistic regression on our data results in the computation of the odds of the dichotomous variable, in our case credulous acceptability, from the values of the argument features. The odds of an argument being credulously accepted are the ratio of the probability of the argument being credulously accepted to the

probability of the opposite.

To carry out this experiment we follow the guidelines provided by Peng et al. (2002).

4.3.1 Experimental Setup

Our initial hypothesis on the training data set is to determine whether the likelihood of an argument being credulously accepted is related to the specified argument features. Our argument sample comprises 57,681 elements, and does not include self-attacking arguments. Missing values are not present in this sample. The argument features—also referred to as independent or exogenous variables—are specified as follows:

- In-degree: a numerical variable containing integer values
- In-degree to arguments ratio: numerical variable containing floating-point values
- In-degree to attacks ratio: numerical variable containing floating-point values
- Out-degree: a numerical variable containing integer values
- Out-degree to arguments ratio: numerical variable containing floating-point values
- Out-degree to attacks ratio: numerical variable containing floating-point values

The variable we are interesting in predicting—also called the dependent or endogenous variable—is credulous acceptability and is encoded as one when the argument is credulously accepted and zero when it is not.

Once we determine whether our endogenous variable is affected by the independent or exogenous variables, we would also like to know how strong is this effect. The logistic regression analysis was carried out using the Python library `statsmodels` version 0.13.5.

Results

Using the training set—self-attacking arguments excluded—we fit a logistic regression model to explain the predicted odds that an argument is credulously accepted. The model included six independent variables and an intercept. The results of logistic regression shows that

	coef	std err	z	$P > z $	[0.025	0.975]
in_degree	-0.3734	0.009	-40.088	0.000	-0.392	-0.355
in_degree_to_num_arg_ratio	12.0582	0.403	29.925	0.000	11.268	12.848
in_degree_to_num_att_ratio	-51.7930	0.824	-62.846	0.000	-53.408	-50.178
out_degree	0.0679	0.008	8.381	0.000	0.052	0.084
out_degree_to_num_arg_ratio	-5.8970	0.361	-16.315	0.000	-6.605	-5.189
out_degree_to_num_att_ratio	15.4766	0.624	24.811	0.000	14.254	16.699
const	1.2454	0.022	55.472	0.000	1.201	1.289
Model Evaluation						
	Likelihood ratio chi-squared statistic	17,177.49	0.000	df	7	
	Wald test (chi-squared) statistic	11,228.11	0.000	df	7	
	F-test	1,612.59	0.000	df	7	
	Dependent var.:	CD-adm^w	No. Observations:	57,681		

TABLE 4.9: Logistic Regression Analysis of credulous acceptability

$$\begin{aligned}
\text{Predicted logit of } (\mathbf{CD-}adm^w) &= 1.245384 \\
&- 0.373446 * in_degree \\
&+ 12.058229 * in_degree_to_num_arg_ratio \\
&- 51.792977 * in_degree_to_num_att_ratio \quad (4.1) \\
&+ 0.067944 * out_degree \\
&- 5.896998 * out_degree_to_num_arg_ratio \\
&+ 15.476552 * out_degree_to_num_att_ratio
\end{aligned}$$

Equation 4.1 shows the linear relationship between the independent variables and the natural logarithm of the odds of an argument being credulously accepted.

With the results of the logistic regression model obtained from the data we can start to address the first hypothesis. The null hypothesis in this context is defined as: the constant term —the value 1.245384 in Equation 4.1— alone offers a good fit to the data without the inclusion of the independent variables. The alternative hypothesis is defined as: the independent variables offer a better fit to the data than the intercept alone.

The likelihood ratio statistic, the Wald statistic and the F-test provided the answer to the previous question (see Table 4.9). The conclusion of all three tests is that we can reject the null hypothesis at both the 0.05 and 0.01 significance level for our training data set. In other words, we have no reason to believe that a model with only the constant term will be more adequate to describe the natural logarithm of the odds of credulous acceptability of an argument.

Regarding the coefficients in our model, we can ask whether they are actually different from zero, since a coefficient with value zero suggests a variable which could be excluded from the model. From Table 4.9, we can reject the null hypothesis

	Dependent variable $CD\text{-}adm^w$
in_degree	0.688
in_degree_to_num_arg_ratio	1,172,513.249
in_degree_to_num_att_ratio	0.000
out_degree	1.070
out_degree_to_num_arg_ratio	0.003
out_degree_to_num_att_ratio	5,264,788.936
const	3.474

TABLE 4.10: Odd ratios

that a coefficient in our model equals zero at the 0.05 and at the 0.01 significance level. Apparently, all the variables selected for the model plus a constant term are linearly related to the logarithm of the odds of an argument being credulously accepted.

There is little useful information that we can gain from our model as it is. The output of our model is given in the logarithmic scale, a trait that is generally hard to interpret. We can however, reach other conclusions when transforming to the odd ratios.

From Table 4.10 we find that increasing the in-degree by one unit while keeping every thing else constant will turn an argument 0.688 times less likely to be credulously accepted while increasing the out-degree by one unit (all other variables held constant) will make an argument 1.070 times more likely to be credulously accepted. We can also observe two variables which when all other variables are held constant, have apparently immense effects on how many times an argument is more likely to be accepted. However, the ratio variables, by their very nature are constrained to be in the closed interval zero to one. Thus, we will never observe increments of one, the increments in the magnitude of these variables will be, in general, very small.

When performing a statistical analysis, like the one we have carried out so far, the ensuing step is to assess how well does the model predicts the outcomes in the data set used. Mainly, because in statistics, it is assumed that the experimental data set is representative of the population and that any conclusion inferred from the analysis will generalize to the population. A point in which statistical analysis and ML differ.

In the previous sections, we described the population under study as the set of unique AFs created using the generator tool provided by Craandijk and Bex (2020) containing between 5 and 55 arguments and solved in under ten minutes by the tool described in Chapter 3. Out of the population, we made a randomized partition and obtained the current experimental data set. This process gives us enough reason to assume our experimental data set is representative of the population.

It should not surprise the reader that the model assessment to be presented is not

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.77	0.79	0.78	33434
YES	0.70	0.68	0.69	24247
Accuracy			0.74	57681
macro avg	0.73	0.73	0.73	57681
weighted avg	0.74	0.74	0.74	57681
MCC = 0.4672				

TABLE 4.11: Logistic Model Assessment

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.77	0.78	0.78	17310
YES	0.68	0.66	0.67	11833
Accuracy			0.73	29143
macro avg	0.73	0.72	0.72	29143
weighted avg	0.73	0.73	0.73	29143
MCC = 0.4486				

TABLE 4.12: Logistic Model Assessment on the Test data set

carried out in a test set, but in the same set used to generate the model. Clarifications being made, we proceed with the assessment of the model.

The metrics we use to assess the model are typical of any study of binary classification using logistic regression plus the phi coefficient also known in ML as Matthews Correlation Coefficient (MCC).

The model assessment using the metrics shown in Table 4.11, indicate that our model does a moderately good job in predicting credulous acceptability from the selected argument features. In all dimensions of the assessment we do better than randomly assigning credulous acceptability to arguments in an AF.

The model's accuracy is moderate, but it offers a great advantage over an exact solver: the computational cost of computing both the model and making predictions is negligible.

For the sake of comparability with the other techniques applied in this work, we re-assess our model on the test data set created previously. As we can observe from Table 4.12, the logistic regression model performs better than random classification on the test set, while still keeping resource consumption at a negligible level.

Chapter 5

Machine Learning (ML) Experiments and Analysis

In the previous chapter, we explored how statistics can help us to identify features of arguments that influence their credulous acceptability under weak admissibility semantics. The results obtained were to some degree promising when we account for the resources consumed. Our logistic regression model was able to predict credulous acceptability with 73% overall accuracy and a MCC of 0.4486, showing it was better than random guessing.

In this chapter, we are interested in exploring whether “more advanced” techniques would improve the above mentioned baselines. The chapter is divided in two parts. The first part investigates Support Vector Machine (SVM)’s potential to make predictions about credulous acceptability using the variables defined in the previous chapter.

The second part is dedicated to the analysis of capabilities of GCNs to address our problem of interest.

5.1 Support Vector Machine (SVM)

From the previous parts of this study we have gained a rough idea of the patterns in our data. We have been getting ready for some more advanced techniques. To this effect we would like to start with one “simple” but powerful classification technique in ML: Support Vector Machine (SVM).

SVM is a product of statistical learning theory which proposes the choice of a hyper-plane that separates the space through the middle maximizing the distance between the nearest sample and the separating hyper-plane. This technique requires that the samples to be classified be represented as high-dimensional vectors in a high-dimensional space (Noble, 2006).

Although SVM is simple and has been replaced during the last decades by more complex models, it can lead to good results in some practical cases (Russell and

Norvig, 2021). Not only can it deliver adequate performance, it also possesses an advantage none of the newer more complex ML techniques has: it can be analyzed from the point of view of a linear separating hyperplane while also being able to represent complex functions (Hearst et al., 1998; Noble, 2006).

When trying to linearly separate data points belonging to different classes, different candidate hyperplanes could do the job (Strang, 2019). If the training data is linearly separable, any of the candidate hyperplanes will induce zero misclassifications. However, it does not necessarily mean that the model will perform equally good in the presence of unseen data. The objective here is to make the model generalizable to unseen data coming from the same distribution as the training data (Russell and Norvig, 2021). This is where the **maximum margin hyperplane** appears.

The preferred hyperplane or maximum margin hyperplane is selected, so that it maximizes the distance between the hyperplane and any of the data points in either class. The distance to the nearest data point is called the **margin** of the hyperplane (Noble, 2006). This is the distance that we are trying to maximize in order to find the maximum margin hyperplane. The vectors stretching perpendicularly from the separating hyperplane to the nearest data points are called the **support vectors**. Thus the name Support Vector Machine (SVM).

In designing a SVM we are trying to balance two different objectives. On one hand, we want to minimize the number of data points which are misclassified. To keep an account of the misclassified data points, it is standard practice to use the **hinge loss**. On the other hand, we want to maximize the distance between the separating hyperplane and the closest data point in either class. By applying some algebraic tricks we can reformulate the problem of maximizing the distance from a hyperplane to the closest data point into a minimization problem. By introducing a parameter C in the final objective function we can balance between the importance of the loss and the importance of the size of the margin. The parameter C is also called a **hyperparameter** that can be varied at the designer's discretion to change the behavior of the model (Goodfellow et al., 2016).

In most practical cases, the data will not be linearly separable, as is the case in our data set. This is when the so-called **kernel trick** comes to the rescue (Strang, 2019; Russell and Norvig, 2021). The trick is based on the idea that if we map our data into higher dimensions, eventually we will find a linear separator. A kernel is a function that maps features into a higher dimensional space in a non-linearly fashion. Unfortunately, there is no bullet-proof method to choosing the right kernel function other than trial and error.

SVM does not make assumptions on the distribution of the data, neither on the training set nor on the test set. It assumes, however, that both data sets are drawn from the same distribution. In other words, the process to generate both sets are the same.

5.1.1 Experimental Setup

To carry out the experiments in this section, we use the tools provided by the Python library `scikit-learn` Release 1.2.1. The implementation of SVM in this tool set comes in handy to all the experiments proposed.

The training data set, the validation set and the test data set as well as the selected features have been already discussed in Sections 4.1.2 and 4.2.1. The argument features are the same as the independent variables used in the logistic regression model and we leave out arguments with self-loops. Additionally, the features for this experiment have been centered and scaled, following the recommendation made in the `scikit-learn` documentation¹. This preprocessing of the data can have positive effects on the performance of the algorithm. Because we have previously observed that our samples contain many outliers, we make use of the `RobustScaler` on the training data set. The same centering and scaling model was later applied to both the validation and the test set.

The objective function in our experiment is based on the `scikit-learn`'s implementation of a support vector classifier. This implementation uses a **Least Squares SVM**. The difference with respect to a classical SVM resides in the fact that the hinge loss in the objective function is squared. This change in the formulation of the problem makes an SVM a lot more effective when working with large amounts of data. The second term of the objective function—the dot product of the coefficients of the maximal margin hyperplane—was the standard in the SVM definition (Gareth et al., 2021).

We run a series of experiments varying the value of the parameter C . In each run the parameter C could assume one of the following values: 10^{-4} , 10^{-3} , 10^{-2} , 10^{-1} , 1 , 10^2 , 10^3 or 10^4 . As previously explained, this parameter drives the importance we give towards minimizing the number of missclassifications or accepting more missclassifications in the training set hoping that the SVM will perform better in unseen data.

As explained previously, when dealing with a data set which is not linearly separable, it is hard to tell which kernel would be appropriate to separate the data points into their respective classes by mapping each data point into a higher dimension. For this reason, we tested four different kernels on the training data. The kernels are defined as follows (Hearst et al., 1998; Hofmann et al., 2008; Strang, 2019; Gareth et al., 2021):

1. Linear Kernel: $\mathbf{K}(x_i, x_j) = \langle x_i, x_j \rangle + const$
2. Polynomial Kernel: $\mathbf{K}(x_i, x_j) = (\langle x_i, x_j \rangle + const)^d$
3. Radial Basis Function (RBF) Kernel: $\mathbf{K}(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
4. Sigmoid Kernel: $\mathbf{K}(x_i, x_j) = \tanh(\gamma \langle x_i, x_j \rangle + const)$

¹<https://scikit-learn.org/stable/modules/svm.html#tips-on-practical-use>

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.78	0.77	0.77	17310
YES	0.67	0.68	0.67	11833
Accuracy			0.73	29143
macro avg	0.72	0.72	0.72	29143
weighted avg	0.73	0.73	0.73	29143
MCC = 0.4454				

TABLE 5.1: Linear SVM Results ($C = 0.0001$)

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.77	0.86	0.81	17310
YES	0.75	0.62	0.68	11833
Accuracy			0.76	29143
macro avg	0.76	0.74	0.74	29143
weighted avg	0.76	0.76	0.76	29143
MCC = 0.4954				

TABLE 5.2: 4th Degree Polynomial SVM Results ($C = 1.0$)

where x_i and x_j are two data points or observations in the data set and $\langle x_i, x_j \rangle$ is the inner product between a pair of training observations. The polynomial kernel offers the possibility of varying the degree d . In our experiments, we tested polynomial kernels within the degrees ranging from two to four.

Another adjustable parameter is introduced in the RBF and the Sigmoid kernel: the constant γ , also called the gain or the spread. In the RBF kernel, the spread γ when taking small values the SVM will show near to linear behavior, when taking larger values we might overfit the model. In the experiments γ can take the following values: $10^{-2}, 1/6, 10^{-1}, 1.0, 10, 10^2$ or $1/(6 * \sigma^2)$, where σ^2 is the sample variance and 6 is the number of properties of an argument.

The “best model” for each kernel is chosen according to the highest MCC achieved on the validation data set.

Results

Tables 5.1 to 5.4 illustrate the best results for each kernel experiment based on a particular setting of hyperparameters. On average, SVM models performed at the same level as the logistic regression model. In practice, all five models are better at predicting non-credulous acceptability than its counterpart, the credulously accepted class. They are also better at determining the credulous acceptance status of

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.77	0.86	0.81	17310
YES	0.75	0.61	0.68	11833
Accuracy			0.76	29143
macro avg	0.76	0.74	0.74	29143
weighted avg	0.76	0.76	0.76	29143
MCC = 0.4978				

TABLE 5.3: RBF SVM Results ($C = 0.1, \gamma = 1.0$)

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.79	0.74	0.76	17310
YES	0.65	0.71	0.68	11833
Accuracy			0.75	29143
macro avg	0.72	0.72	0.72	29143
weighted avg	0.73	0.73	0.73	29143
MCC = 0.4410				

TABLE 5.4: Sigmoid SVM Results ($C = 0.01, \gamma = 0.01$)

an argument than if we threw a fair coin and decided credulous acceptability on the basis of the result.

5.2 Graph Convolutional Neural Network (GCN)

We have previously explained some of the basic ideas underlying GCN models. In Section 2.3.1, we enumerated the inputs and outputs of this technique as implemented by (Kipf and Welling, 2016).

The basic premise of the GCN model is that a graph can be treated as a two dimensional structure. A Convolutional Neural Network (CNN) resembles the way a human recognize patterns in an image, by identifying edges, contours, color clusters, and so on (Gareth et al., 2021; Strang, 2019). In a similar way, a Convolutional Neural Network (CNN) can be helpful in learning to recognize credulously accepted arguments in an AF.

Previous studies (Kuhlmann and Thimm, 2019; Kuhlmann et al., 2022; Malmqvist et al., 2020) have used variations of Kipf and Welling’s GCN model. For example, Kuhlmann and Thimm (2019) proposed an GCN architecture with consisting of one convolution layer followed by a single neuron with a linear function and an activation layer based on the rectified linear unit $ReLU(z)$ function. The input was conformed by the adjacency matrix of the undirected graph of an AF and a fea-

ture matrix consisting of the in- and out-degree of the arguments in the AF. The later has given this model its nickname: FM2.

Other studies have used more complex architectures (Malmqvist et al., 2020). Besides the adjacency matrix of the undirected graph, additional graph embeddings have been added to the inputs. Another important change to the architecture was the addition of more convolution layers and residual connections. The final activation layer was in this case based on a sigmoid function.

In this part of the study we are interested exploring the feasibility of the GCN architecture as proposed initially by Kuhlmann and Thimm (2019) in determining the credulous acceptability of single arguments in an AF.

The details of this experiment are explained below.

5.2.1 Experimental Setup

The training, validation and test data sets are known from Section 4.2.1. In previous analysis, we have ignored the self-attacking arguments. However, for this experiment we will take no provisions in this respect.

We adapted the code provided by Craandijk and Bex (2020) to meet our requirements. The adaptation consisted mainly of adding weakly preferred semantics as an option and include additional performance metrics so as to make it comparable to previous experiments.

The convolutional layer `GCNConv` included in the Python package `torch_geometric` implements one-to-one the convolution operation as described by Kipf and Welling (2016). As such it was carried out on the data conformed by the adjacency matrix and the two dimensional feature matrix consisting of the in- and out-degree of each argument.

In this experiment the GCN was trained batch-wise, i.e. AFs were processed in batches of size 32, whereas in the original setting all adjacency matrices in the training set were diagonally stacked to form a large input matrix. Batch size is one of those parameters which can be adjusted to change the behavior of the model. This experiment was carried out fixing the batch size to 32 because the training sample was exactly divisible by this number.

We did only one run of the experiment with learning rate set to maximum of 0.0001, hidden dimension was 128 and there was no dropout layer. The best model is chosen based on the smallest loss in the validation set. Loss was quantified using binary cross-entropy between the predicted likelihood of being credulously accepted and the true status of an argument.

Other adjustable parameters were set to the standard values of the convolution and readout functions provided by the `torch` package.

We allowed the FM2 model to look for a classifier in a thousand iterations or epochs. At the end of each epoch, the model obtained was tested on the validation set. Only the best performing model with respect to the loss was temporarily stored. Later on, in case another model showed smaller loss on the validation set would the

	Precision	Recall	F1-score	Support
CD-adm^w				
NO	0.73	0.88	0.80	18357
YES	0.73	0.50	0.59	11833
Accuracy			0.73	30190
macro avg	0.73	0.69	0.70	30190
weighted avg	0.73	0.73	0.72	30190
MCC = 0.4160				

TABLE 5.5: FM2 Model Results

stored model be replaced by the better model.

Results

The results of this experiment on the test set are displayed in Table 5.5. The MCC indicates a moderate prediction quality in line with the overall accuracy. The model does a better job at correctly classifying non-credulously accepted arguments than it does at correctly identifying the counter-class: credulously accepted arguments. Overall, in spite of its complexity, the performance of this model on the test set is the weakest when compared to the models computed so far.

Chapter 6

Discussion and Conclusion

Classic semantics in argumentation are problematic in the face of particular attack structures, i.e. self-attacking arguments or arguments in odd-length attack cycles (Dung, 1995; Baroni et al., 2018). Weak Admissibility semantics suggest an alternative that is neither too strict nor too permissive in the definition of collectively acceptable arguments or extensions. Research has shown that computational problems in this semantics were hard and complex to solve.

In the beginning of this research we set out to accomplish three goals. They were aimed at gaining a better understanding of computational problems in weak admissibility semantics from an empirical perspective. We took a particular interest in credulous acceptability under weak admissibility semantics.

In the following sections we summarize and discuss our results in light of our own findings with respect to each of the three goals formulated in Chapter 1.

6.1 Weak Admissibility Semantics

We started out by implementing an interpretation of the algorithms proposed by Dvořák et al. (2021) and Dvořák et al. (2022). We targeted our implementation to enumerate weakly preferred extensions in a given AF.

To achieve our target, we made use of theoretically founded techniques aimed at reducing the problem size and —to a lesser extend— also the problem complexity Dvořák et al. (2022).

We showed how enumerating weakly preferred extensions constitutes a building block in the solution of the problem of deciding credulous acceptability of the arguments in an AF under weak admissibility semantics. Still, there remains some work to do to implement solvers for the other problems in weak admissibility semantics.

By the standards of the ICCMA 2017, our solver did a rather humble job in solving less than a third of the total benchmark in under 10 minutes. Unfortunately, we did not have the resources to obtain a solution for every AF in the benchmark suite.

We concluded that the solved set differed in many different dimensions from the unsolved set, particularly we detected a non linear interaction between arguments, attacks and the performance of the solver in terms of run-time.

Future research could do away with the 10 minute limit we imposed on run-time and analyze the behavior of the solver in light of number of arguments, attacks, SCCs, graph types, and other features of the AF. This step could help us identify bottlenecks or help us direct the code optimization process (Vallati et al., 2019).

Having a completely solved data set would also increase the chance that we find better heuristics that may complement an exact solver.

6.2 Statistical Analysis and Logistic Regression

Many statisticians and data scientists recommend performing a descriptive analysis of the data available before jumping into any modeling. This advice is often overlooked. We tend to go directly into formulating models which attempt to recreate the relation between input and output without further analysis of the data.

The above recommendation was the reason for the second goal of the study. We explored whether “simpler” techniques would be useful in identifying determinants of credulous acceptability under weakly admissible semantics.

However, we felt that the small portion of solved AFs we had computed in the previous steps would not be enough to comply with the comparability requirements in this and in subsequent experiments. A different compilation of AFs was generated and solved to continue the study. We have already explained the way in which our data set may be biased. We have to face the fact that the bias on the data set is one of the major weakness of our study. This weakness was mainly induced by the ten minute limit on run-time. Future work, should probably be better off by eliminating this constraint.

It was important to gain an idea of how the data was distributed, i.e. what features are shared among arguments, how did they differ from one another. From simple exploratory statistics we discovered discrepancies between credulously accepted arguments and their non-accepted counterparts. Furthermore, we incorporated relevant AF information to create more informative features that proved also influential in the credulous acceptability of arguments under weak admissibility.

Our analysis resulted in the definition of six argument features that had a relationship to credulous acceptability. With our selected feature were able to compute “simple” and more “complex” models, but there are other characteristics at the AF-level and at the argument-level that were left unexplored. For example, the number of SCCs in the host AF, size of the largest SCC, and other measures mentioned by Vallati et al. (2019). These features could be also incorporated to the argument features by translating them into ratios, i.e. a new feature could reflect the probability of an argument of being a member in the largest SCC of the host AF. Thus providing more distinctive features at the argument level.

More distinctive features could render the arguments linearly separable in higher dimensions, a fact from which an SVM's performance would profit.

We computed a logistic model that not only predicted credulous acceptability under weak admissibility semantics better than chance. The logistic model also offered us some degree of interpretability. We learned that the ratio of the in-degree to the number of arguments in an AF can greatly influence the likelihood of an argument being credulously accepted. The same can be said about the ratio out-degree to number of attacks.

To the best of our knowledge, this type of analysis has not been performed previously neither in the realm of weak admissibility semantics nor in the realm of classic semantics. We showed that, at least for our data set, "simple" techniques perform at the level of more complex techniques without compromising interpretability and with a rather tiny resource consumption.

Future work could —on the basis of a completely solved ICCMA data set— reproduce our experiments and establish whether similar performance results can be achieved in this and maybe other data sets. In short, future research should take on the question of general feasibility of simpler prediction models on both classic and weakly admissible semantics.

We want to bring to the readers attention that weak admissibility semantics will, in general, result in more credulously accepted arguments than their classic counterparts. We conjecture that the expectation of the number of credulously accepted arguments under weak admissibility semantics will be larger than under classic semantics. Further research could shed some light into this conjecture and possibly analyze how arguments under each semantics differ.

The number of credulously accepted arguments in our sample sets was smaller than the number of non-accepted arguments. It is worth noting that we did not take any provisions that would incorporate and mitigate this potential issue into the models we computed. We will leave it to future research to address this situation.

6.3 Machine Learning (ML) Models

ML techniques enable us to learn directly and automatically from the data. ML models can make really good predictions without an explicit model structure. Also these models are not subject to the constraints posed to other regression methods. On the other hand, training these models is also more difficult, it requires large amounts of data and one has to pay attention to how the hyperparameters are tuned. Suddenly, we may find ourselves testing a numerous combination of parameters. As a result, interpretability is sacrificed. We know what went into the model, we know what came out, but we are unable to explain which features played a role in the output.

With the six argument features found in the descriptive and exploratory phases, we modeled a SVM using four different kernels and varying, for each model, the regularization parameter. For the RBF kernel and the Sigmoid kernel we also var-

ied the gain parameter. The best results were achieved by the SVM with an RBF kernel. The RBF kernel performed marginally better than the other kernels. Its performance was very similar to the fourth degree polynomial kernel, however the RBF kernel was faster to compute. When compared to the results of the logistic model, it got better at identifying non credulously accepted arguments, but the same cannot be said about credulously accepted arguments. The accuracy of the model was marginally better than the other models computed but only because more non credulously accepted arguments were correctly identified.

When using even more complex techniques such as the FM2 model based on GCNs, we found that the overall accuracy was not better than the simpler models. The FM2 model's ability to correctly identify non-credulous acceptability was better than in the RBF-SVM model. From all the models computed, the FM2 model was the weakest at correctly identifying credulously accepted arguments. In our view, the results of this model were rather meager compared to its resource consumption in terms of training time and computational resources.

We did not carry out any type of formal assessment regarding resource consumption when computing models and testing them on unseen data. However, we perceived large discrepancies in the time we had to wait to obtain a model. Logistic regression returned almost immediately a moderately good prediction model. Other techniques took minutes or even hours to compute also moderately good models. We cannot help but ask ourselves, whether there is some merit to the statistician's advice: "look first at the data".

In another vein, other ML models have shown significantly better results when applied to classic semantics (Craandijk and Bex, 2020). In this work, we assessed only two models. One of them had not been previously used in the context of neither classic semantics nor weak admissibility semantics —the SVM model. The second model, FM2 (Kuhlmann and Thimm, 2019), is better known for its mild results in classic semantics and in this work. It is only natural to ask oneself whether the AGNN model would do a good job predicting credulous acceptability under weak admissibility semantics. However, this question will also have to be addressed by future research.

6.4 Recommendations for Future Work

Regarding the data collection process, we faced the challenge of not sharing a common AF repository with other researchers, a situation that Kuhlmann et al. (2022) have already pointed out. Except for the set of AFs from the ICCMA 2017 competition, every researcher has had to invest resources in the generation of their own AF sample set. In this research we benefited in this respect, from previous work by Craandijk and Bex (2020), but not without being well aware that this work missed some other important benchmarks which have been in further development since the ICCMA 2017 (Gaggl et al., 2020).

Future work might surely benefit from the integration of all available and widely

accepted AF generation tools in one standard toolbox. Better yet would be the online availability of a common repository of AFs on which researchers can train and test their models. This step would make research in this field a bit more rigorous and comparable.

Worthy of consideration is also the implementation of a “hybrid” solver in which an exact algorithm is complemented with a approximate solution based on the results of this work. It has been remarked in this work and in other studies (Kuhlmann et al., 2022; Kuhlmann and Thimm, 2019; Craandijk and Bex, 2020; Malmqvist et al., 2020), that some models deliver results in reasonable time at the expense of accuracy. However, in our opinion, the further development of exact solvers should not be at odds with the use of approximate solutions that under the right circumstances and at the right point may significantly decrease the search space (Pearl and Kim, 1982).

In this work, we did not more than re-purpose the model proposed by Kipf and Welling (2016) and adapted by (Kuhlmann and Thimm, 2019) to the problem of determining credulous acceptability under weak admissibility semantics. In section 2.3.1, we explained how the GCN model consists of the adjacency matrix of the undirected AF graph. In section 2.3.2 we described how this model would crumble down if the undirected adjacency matrix would be substituted by the adjacency matrix of the directed graph. However, more recent research by Kollias et al. (2022) have studied how to encode information from the adjacency matrix of a directed graph for the purposes of training a GCN for classification. Future research should consider this findings and incorporate them in new predictive models.

Appendix A

Weak Admissibility Exact Solver

A.1 Algorithm using SCC-recursiveness

```
"""
```

```
File: wadmsolver.py
```

```
A class containing the methods to generate weakly preferred  
extensions according to the  
definitions presented by Baumann et al. (2020,2020a) using  
the algorithms proposed in Dvorak et al. (2021, 2022).
```

```
The argumentation framework (AF) is contained in a digraph as  
defined by the Python module 'networkx'.
```

```
Author: Carla I. Sanchez Aguilar, Artificial Intelligence  
Group, Fernuniversitaet in Hagen
```

```
"""
```

```
import networkx as nx
```

```
class WAdmSolverSCC:
```

```
    af = nx.DiGraph()
```

```
    def __init__(self, af: nx.DiGraph):
```

```
        """
```

```
        Args:
```

```
            af: An AF represented as a DiGraph object
```

```
        """
```

```
        self.af = af.copy()
```

```
        # Eliminate all self-attacking arguments. First
```

```

        preprocessing step.
self.af.remove_nodes_from(nx.nodes_with_selfloops(af)
)
self.nodes = list(self.af.nodes)
self.edges = set(self.af.edges)
self.successors = {} # Arguments attacked by an
argument
self.predecessors = {} # A dictionary containing the
defeaters of each argument.

for node in self.nodes:
    self.successors[node] = set(self.af.successors(
node))
    self.predecessors[node] = set(self.af.
predecessors(node))

def successors_of(self, af_nodes, nodes_set):
    """
    Computes the set of arguments in AF attacked by the
    set of arguments 'nodes_set'.
    :param af_nodes: A list of the nodes contained
in the AF.
    :param nodes_set: A subset of 'af_nodes'
    :return: The set of all nodes
attacking 'nodes_set' in AF.
    """

    successors_over_all = set()
    for node in nodes_set:
        successors_over_all = successors_over_all | self.
successors[node]
    return successors_over_all & set(af_nodes)

def predecessors_of(self, af_nodes, nodes_set):
    """
    Finds the arguments in 'af_nodes' attacking the set
    of arguments 'nodes_set'
    :param af_nodes: A list containing the
arguments of an AF.
    :param nodes_set: A set of arguments in AF.
    :return: A set of arguments in '
af_nodes' attacking any argument in the set '
node_set'
    """

```

```

"""
defeaters_overall = set()
for node in nodes_set:
    defeaters_overall = defeaters_overall | self.
        predecessors[node]
return defeaters_overall & set(af_nodes)

def strongly_connected_components(self, af_nodes):
"""
Generates the SCCs of the AF conformed by af_nodes.
We borrowed some code from the Python
package 'networkx' and adapted it to accept a list of
the nodes in AF. We decided for the
non-recursive implementation of Tarjan's algorithm
adapted from Nuutila et al. (1994) because it
works well with both sparse graphs and with highly
connected graphs.
:param af_nodes: A list of the nodes contained
in the AF.
:return: The set of strongly connected
components of AF.
"""

preorder = {}
lowlink = {}
scc_found = set()
scc_queue = []
i = 0 # Preorder counter
# We want to obtain the successors of each node
# contained in 'af_nodes', successors should be part
# of 'af_nodes'
successors = {node: iter(self.successors[node] & set(
    af_nodes)) for node in af_nodes}
successors_dict = {node: (self.successors[node] & set
    (af_nodes)) for node in af_nodes}
for source_node in af_nodes:
    if source_node not in scc_found:
        stack = [source_node]
        while stack:
            v = stack[-1]
            if v not in preorder:
                i = i + 1
                preorder[v] = i

```

```

done = True
for w in successors[v]:
    if w not in preorder:
        stack.append(w)
        done = False
        break
if done:
    lowlink[v] = preorder[v]
    for w in successors_dict[v]:
        if w not in scc_found:
            if preorder[w] > preorder[v]:
                lowlink[v] = min([lowlink[v], lowlink[w]])
            else:
                lowlink[v] = min([lowlink[v], preorder[w]])
    stack.pop()
    if lowlink[v] == preorder[v]:
        scc = {v}
        while scc_queue and preorder[scc_queue[-1]] > preorder[v]:
            k = scc_queue.pop()
            scc.add(k)
        scc_found.update(scc)
        yield scc
    else:
        scc_queue.append(v)

def reduct(self, nodes_af, nodes_set):
    """
    Computes the reduct of the AF arguments 'nodes_af'
    with respect to a set of arguments 'nodes_set'.
    :param nodes_af: A list containing the
        arguments of an AF.
    :param nodes_set: A set of arguments from which
        to compute the reduct in AF.
    :return: A list of arguments contained
        in the reduct of the AF w.r.t. 'nodes_set'.
    """
    nodes_set_plus = self.successors_of(nodes_af,
        nodes_set)
    nodes_star = nodes_set | nodes_set_plus

```

```

    return set(nodes_af) - nodes_star

def generate_cf_sets(self, af_nodes):
    """
    Computes all the conflict-free sets in AF, notice we
    assumed AF contains no self-loops.
    :param af_nodes:      A list of the nodes contained
                          in AF, none of the nodes attacks itself.
    :return:              The set of all conflict-free
                          sets in AF.
    """
    cf_sets = set()
    if len(af_nodes) > 0:
        cf_sets.add(frozenset([af_nodes[0]]))
        # Compute a new AF which does not include neither
        # the current argument, nor the arguments
        # attacked by it.
        aux_af = self.reduct(af_nodes, set([af_nodes[0]]))
        compatible_nodes = set()
        for node in aux_af:
            if node not in self.predecessors_of(af_nodes,
                                                set([af_nodes[0]])):
                compatible_nodes.add(node)

        if len(compatible_nodes) > 0:
            aux_cf_sets = self.generate_cf_sets(list(
                compatible_nodes))
            for aux_set in aux_cf_sets:
                cf_sets.add(frozenset(set([af_nodes[0]]
                                          | aux_set)))

        other_cf_sets = self.generate_cf_sets(af_nodes
                                              [1:])

        cf_sets = cf_sets | other_cf_sets

    return cf_sets

def is_w_adm(self, af_nodes, cf_set):
    """
    Checks whether the set of arguments 'cf_set' is
    weakly admissible in AF.
    """

```

```

:param af_nodes:      A list containing the
                      arguments of AF.
:param cf_set:       A set of conflict-free
                      arguments in AF.
:return:             True if the set 'cf_set' is
                      weakly admissible in AF, False otherwise.
"""
# we assume the test for conflict-free sets has been
  carried out.
if len(cf_set) == 0:
    return True # The empty set is w-admissible

# Find the defeaters of the arguments contained in '
  cf_set' in the argumentation framework 'af'.
defeaters = self.predecessors_of(af_nodes, cf_set)
# When there are no defeaters of 'cf_set', the set is
  w-admissible.
if len(defeaters) == 0: # The set 'cf_set' has no
  defeaters in 'af'.
    return True

reduct_af = self.reduct(af_nodes, cf_set)
# Find the defeaters of 'cf_set' that are present in
  the reduct.
defeaters_reduct = reduct_af & defeaters
# If there are no defeaters of 'cf_set' in the reduct
  , the set is w-admissible.
if len(defeaters_reduct) == 0:
    return True
# Compute the grounded extension of the reduct.
red_gr_ext = self.generate_grounded_ext(reduct_af)
# In case the grounded extension contains a defeater
  of cf_set, the set is not w-admissible.
if len(defeaters_reduct & red_gr_ext) > 0:
    return False

# Compute the reduct of the previous AF w.r.t. to the
  grounded extension
aux_af = self.reduct(reduct_af, red_gr_ext)
# In case there are still defeaters of 'cf_set' in
  this reduced AF
if len(defeaters_reduct & aux_af) > 0:

```

```

        for w_pref_set in self.generate_w_pref_sets(list(
            aux_af)):
            if len(defeaters_reduct & w_pref_set) > 0:
                return False

    return True

def find_w_pref_sets_scc(self, af_nodes):
    """
    Finds the w-preferred sets using strongly connected
    components.
    :param af_nodes:      A list containing the
        arguments of AF.
    :return:              The set of w-preferred
        extensions of AF.
    """
    w_pref_sets = set()
    sccs = list(self.strongly_connected_components(
        af_nodes))

    if len(af_nodes) == 0:
        w_pref_sets.add(frozenset())

    for scc in sccs:
        # Sort out initial SCCs, i.e. SCCs which have no
        # incoming edges.
        if len(self.predecessors_of(af_nodes, scc) - scc)
            == 0:
            scc_cf_sets = self.generate_cf_sets(list(scc)
                )
            w_adm_set_in_scc = False # A flag denoting
                the existence of w-adm sets in the scc
            for scc_cf_set in scc_cf_sets:
                if self.is_w_adm(list(scc), scc_cf_set):
                    w_adm_set_in_scc = True # A w-adm
                        set was found in the scc
                    reduct_af_nodes = self.reduct(
                        af_nodes, scc_cf_set)
                    if len(reduct_af_nodes) == 0:
                        w_pref_sets.add(scc_cf_set)
                else:
                    # Compute the reduced AF which
                    # only should include the nodes

```

```

        in the reduct
        for other_set in self:
            generate_w_pref_sets(
                reduct_af_nodes):
                w_pref_sets.add(scc_cf_set |
                    other_set)

    if not w_adm_set_in_scc:
        # A w-admissible extension was not found
        # in this SCC, go on looking for w-
        # admissible sets in the AF
        # resulting from removing the scc
        new_af_nodes = set(af_nodes) - scc
        for other_set in self:
            generate_w_pref_sets(new_af_nodes):
                w_pref_sets.add(frozenset() |
                    other_set)

    return w_pref_sets

def generate_grounded_ext(self, af_nodes):
    """
    Generates the grounded extension of an AF. This
    function was adapted from Wolfgang Dvorak's
    solver for Dung's semantics.
    :param af_nodes: A list of containing the
        arguments of the AF.
    :return: A set of arguments conforming
        the grounded extension of AF.
    """
    gr_set = set()

    if len(af_nodes) == 0:
        return set()

    # Find undefeated nodes.
    for node in af_nodes:
        if len(self.predecessors_of(af_nodes, [node])) ==
            0:
            gr_set.add(node)

    if len(gr_set) == 0:
        return set()

```



```

# Compute a new af in which both undefeated nodes and
# nodes attacked by these are not in.
reduced_af = self.reduct(af_nodes , gr_set)

gr_set = gr_set | self.generate_grounded_ext(
    reduced_af)

return gr_set

def generate_w_pref_sets(self , af_nodes):
    """
    Given an argumentation framework, this function
    generates weakly preferred sets on the premise of
    first
    carrying out a second pre-processing step which
    consists of finding the grounded extension thus in
    the best cases
    reducing the size of the set of arguments contained
    in the AF. It then uses an approach based on SCCs
    to find
    weakly preferred sets that will complement the
    grounded extension and form weakly preferred
    extensions.

    :param af_nodes:      A list of containing the
        arguments of the AF.
    :return:              A set containing all weakly
        preferred extensions found in the AF.
    """
    w_pref_sets = set()

    # Compute the grounded extension , second
    # preprocessing step
    gr_ext = self.generate_grounded_ext(af_nodes)
    # Eliminate the grounded extension and the arguments
    # it defeats (reduct) from the af.
    reduced_af = self.reduct(af_nodes , gr_ext)
    # We need to find the w-preferred extensions in the
    # remaining reduced AF.
    aux_w_pref_sets = self.find_w_pref_sets_scc(
        reduced_af)
    # Add the grounded extensions to the w-preferred sets
    # found in the reduct.

```

```
for w_pref_set in aux_w_pref_sets:
    w_pref_sets.add(frozenset(w_pref_set | gr_ext))

return w_pref_sets

def generate_w_pref_extensions(self):
    """
    Returns the w-preferred sets in the AF.
    """
    return self.generate_w_pref_sets(self.nodes)
```

Bibliography

- Atkinson, K., P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G. Simari, M. Thimm, and S. Villata (Oct. 2017). “Towards Artificial Argumentation”. In: *AI Magazine* 38.3, pp. 25–36. DOI: 10.1609/aimag.v38i3.2704. URL: <https://ojs.aaai.org/index.php/aimagazine/article/view/2704>.
- Baroni, P., M. Caminada, and M. Giacomin (Feb. 2018). “Abstract argumentation frameworks and their semantics”. In: *Handbook of Formal Argumentation*. Ed. by P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre. Vol. 1. College Publications, pp. 157–234. URL: https://users.cs.cf.ac.uk/CaminadaM/publications/HOFA_semantics.pdf.
- Baroni, P., M. Giacomin, and G. Guida (2005). “SCC-recursiveness: a general schema for argumentation semantics”. In: *Artificial Intelligence* 168.1, pp. 162–210. ISSN: 0004-3702. DOI: 10.1016/j.artint.2005.05.006. URL: <https://www.sciencedirect.com/science/article/pii/S0004370205000962>.
- Baumann, R., G. Brewka, and M. Ulbricht (Sept. 2020a). “Comparing Weak Admissibility Semantics to their Dung-style Counterparts – Reduct, Modularization, and Strong Equivalence in Abstract Argumentation”. In: *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 79–88. DOI: 10.24963/kr.2020/9.
- (Apr. 2020b). “Revisiting the Foundations of Abstract Argumentation Semantics Based on Weak Admissibility and Weak Defense”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03, pp. 2742–2749. DOI: 10.1609/aaai.v34i03.5661. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/5661>.
- Bench-Capon, T. and P. E. Dunne (2007). “Argumentation in artificial intelligence”. In: *Artificial Intelligence* 171.10. *Argumentation in Artificial Intelligence*, pp. 619–641. ISSN: 0004-3702. DOI: 10.1016/j.artint.2007.05.001. URL: <https://www.sciencedirect.com/science/article/pii/S0004370207000793>.
- Cerutti, F., S. A. Gaggl, M. Thimm, and J. P. Wallner (Feb. 2018). “Foundations of Implementations for Formal Argumentation”. In: *Handbook on Formal Argumentation*. Ed. by P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre. College Publications, pp. 688–767. URL: https://iccl.inf.tu-dresden.de/w/images/d/d5/HOFA_2017.pdf.

- Cerutti, F., M. Giacomin, and M. Vallati (2014a). "Generating Challenging Benchmark AFs." In: vol. 14, pp. 457–458.
- Cerutti, F., N. Oren, H. Strass, M. Thimm, and M. Vallati (2014b). "A Benchmark Framework for a Computational Argumentation Competition." In: *COMMA*, pp. 459–460.
- Charwat, G., W. Dvořák, S. A. Gaggl, J. P. Wallner, and S. Woltran (2015). "Methods for Solving Reasoning Problems in Abstract Argumentation – A Survey". In: *Artificial Intelligence Journal* 220.0, pp. 28–63. DOI: 10.1016/j.artint.2014.11.008.
- Craandijk, D. and F. Bex (2020). "Deep Learning for Abstract Argumentation Semantics". In: *CoRR abs/2007.07629*. arXiv: 2007.07629.
- Dung, P. M. (1995). "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games". In: *Artificial Intelligence* 77.2, pp. 321–357. ISSN: 0004-3702. DOI: 10.1016/0004-3702(94)00041-X. URL: <https://www.sciencedirect.com/science/article/pii/000437029400041X>.
- Dvořák, W., M. Ulbricht, and S. Woltran (2021). "Recursion in Abstract Argumentation is Hard—On the Complexity of Semantics Based on Weak Admissibility". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 7, pp. 6288–6295.
- (2022). "Recursion in Abstract Argumentation is Hard—On the Complexity of Semantics Based on Weak Admissibility". In: *Journal of Artificial Intelligence Research* 74, pp. 1403–1447.
- Gaggl, S. A., T. Linsbichler, M. Maratea, and S. Woltran (2020). "Design and results of the Second International Competition on Computational Models of Argumentation". In: *Artificial Intelligence* 279, p. 103193. ISSN: 0004-3702. DOI: 10.1016/j.artint.2019.103193. URL: <https://www.sciencedirect.com/science/article/pii/S0004370218302029>.
- Gaggl, S. A. and S. Woltran (2013). "The cf2 argumentation semantics revisited". In: *Journal of Logic and Computation* 23.5, pp. 925–949. DOI: 10.1093/logcom/exs011.
- Gareth, J., D. Witten, T. Hastie, and R. Tibshirani (2021). *An introduction to statistical learning*. 2nd. Vol. 112. Springer.
- Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2020). "Message passing neural networks". In: *Machine learning meets quantum physics*. Springer, pp. 199–214.
- Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.
- Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, pp. 11–15.

- Hearst, M., S. Dumais, E. Osuna, J. Platt, and B. Scholkopf (1998). "Support vector machines". In: *IEEE Intelligent Systems and their Applications* 13.4, pp. 18–28. ISSN: 2374-9423. DOI: 10.1109/5254.708428.
- Hofmann, T., B. Schölkopf, and A. J. Smola (2008). "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3, pp. 1171–1220. DOI: 10.1214/009053607000000677.
- Hsu, D. F., X. Lan, G. Miller, and D. Baird (2017). "A Comparative Study of Algorithm for Computing Strongly Connected Components". In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp. 431–437. DOI: 10.1109/DASC-PiCom-DataCom-CyberSciTec.2017.85.
- Kipf, T. N. and M. Welling (2016). "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR abs/1609.02907*. arXiv: 1609.02907.
- Kollias, G., V. Kalantzis, T. Idé, A. Lozano, and N. Abe (2022). *Directed Graph Auto-Encoders*. DOI: 10.48550/ARXIV.2202.12449. arXiv: 2202.12449.
- Kuhlmann, I. and M. Thimm (2019). "Using Graph Convolutional Networks for Approximate Reasoning with Abstract Argumentation Frameworks: A Feasibility Study". In: *Scalable Uncertainty Management*. Ed. by N. Ben Amor, B. Quost, and M. Theobald. Cham: Springer International Publishing, pp. 24–37. ISBN: 978-3-030-35514-2.
- Kuhlmann, I., T. Wujek, and M. Thimm (2022). "On the Impact of Data Selection when Applying Machine Learning in Abstract Argumentation". In: *Proceedings of the 9th International Conference on Computational Models of Argument (COMMA'22)*.
- Lagniez, J.-M., E. Lonca, J.-G. Mailly, and J. Rossit (Dec. 2020). "Solver Requirements". In: *The Fourth International Competition on Computational Models of Argumentation*. URL: <https://www.argumentationcompetition.org/2021/SolverRequirements.pdf>.
- Malmqvist, L., T. Yuan, P. Nightingale, and S. Manandahr (2020). "Determining the Acceptability of Abstract Arguments with Graph Convolutional Networks". In: *SAFA@COMMA*, pp. 47–56.
- Noble, W. S. (2006). "What is a support vector machine?" In: *Nature biotechnology* 24.12, pp. 1565–1567.
- Nuutila, E. and E. Soisalon-Soininen (1994). "On finding the strongly connected components in a directed graph". In: *Information Processing Letters* 49.1, pp. 9–14. ISSN: 0020-0190. DOI: 10.1016/0020-0190(94)90047-7. URL: <https://www.sciencedirect.com/science/article/pii/0020019094900477>.
- Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc. DOI: 10.5555/525.
- Pearl, J. and J. H. Kim (1982). "Studies in Semi-Admissible Heuristics". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-4.4*, pp. 392–399. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1982.4767270.

- Peng, C.-Y. J., K. L. Lee, and G. M. Ingersoll (2002). "An Introduction to Logistic Regression Analysis and Reporting". In: *The Journal of Educational Research* 96.1, pp. 3–14. DOI: 10.1080/00220670209598786. eprint: <https://doi.org/10.1080/00220670209598786>.
- Perozzi, B., R. Al-Rfou, and S. Skiena (2014). "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, pp. 701710. ISBN: 9781450329569. DOI: 10.1145/2623330.2623732.
- Russell, S. and P. Norvig (2021). *Artificial Intelligence - A Modern Approach*. 4th. Pearson series in artificial intelligence. Pearson Education. ISBN: 978-0-13-461099-3.
- Simon, H. A. (1996). *The sciences of the artificial*. 3rd Edition. MIT press.
- Skiena, S. S. (2020). *The Algorithm Design Manual*. 3rd. Texts in Computer Science. Springer Verlag.
- Strang, G. (2019). *Linear algebra and learning from data*. Vol. 4. Wellesley-Cambridge Press Cambridge.
- Tarjan, R. (1971). "Depth-first search and linear graph algorithms". In: *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pp. 114–121. DOI: 10.1109/SWAT.1971.10.
- (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2, pp. 146–160. DOI: 10.1137/0201010. eprint: <https://doi.org/10.1137/0201010>.
- Thimm, M. (2014). "Tweety: A comprehensive collection of java libraries for logical aspects of artificial intelligence and knowledge representation". In: *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.
- (2017). "The Tweety library collection for logical aspects of artificial intelligence and knowledge representation". In: *KI-Künstliche Intelligenz* 31.1, pp. 93–97.
- Vallati, M., F. Cerutti, and M. Giacomini (2019). "Predictive models and abstract argumentation: the case of high-complexity semantics". In: *The Knowledge Engineering Review* 34, e6. DOI: 10.1017/S0269888918000036.
- Veerman, J. J. P. and R. Lyons (2020). *A Primer on Laplacian Dynamics in Directed Graphs*. DOI: 10.48550/ARXIV.2002.02605. arXiv: 2002.02605.