Faculty of Mathematics and Computer Science

Artificial Intelligence Group

Implementation of Semiring-based Weighted Argumentation Frameworks using Answer Set Programming

Bachelor's Thesis

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

submitted by

Dominik Stummer

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

Advisor: Lars Bengel
Artificial Intelligence Group

Statement

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

Linz, 15.04.2024 [illegible]

(Place, Date) (Signature)

Zusammenfassung

Abstrakte Argumentationsgraphen bieten eine Methodik zur Modellierung und Analyse von Argumenten und deren Beziehungen zueinander. Die grundlegende Struktur eines abstrakten Argumentationsgraphen wird durch einen gerichteten Graphen dargestellt, wobei die Knoten des Graphen die Argumente repräsentieren und die gerichteten Kanten die Attacken zwischen den Argumenten darstellen. Das Konzept der abstrakten Argumentationsgraphen wird kontinuierlich durch neue Formalismen erweitert, beispielsweise durch gewichtete Argumentationsgraphen mit gewichteten Attacken. Durch das Gewichten der Attacken können wir deren Stärke repräsentieren und somit den in einem Argumentationsgraphen enthaltenen Informationsgehalt erhöhen. Die von Bistarelli et al. vorgestellten Semiring-basierten gewichteten Argumentationsgraphen ermöglichen eine Betrachtung des Gesamtgewichts von Angriffen, was zu einer besonderen Definition der Verteidigung führt. In dieser Bachelorarbeit präsentieren wir einen ersten Ansatz zur Implementierung von Semiring-basierten gewichteten Argumentationsgraphen unter Verwendung von Answer-Set-Programming, einer Form der logischen Programmierung und evaluieren die ASP-Implementierung unter der Betrachtung verschiedener Problemtypen.

Abstract

Abstract argumentation frameworks provide a methodology for modeling and analyzing arguments and their relationships to each other. The basic structure of an abstract argumentation framework is represented by a directed graph, where the nodes of the graph represent the arguments and the directed edges represent the attacks between the arguments. The concept of abstract argumentation frameworks is continuously being expanded through new formalisms, for instance through weighted argumentation frameworks with weights on attacks. By assigning weights to attacks, we can represent their strength, thus enhancing the information contained within an argumentation framework. The semiring-based weighted argumentation frameworks from Bistarelli et al. allow for a composition of weights of attacks, leading to a unique notion of defence. In this bachelor's thesis, we present an initial approach to implement semiring-based weighted argumentation frameworks using answer set programming, a form of logical programming, and subsequently evaluate the ASP-implementation considering various problem types.

Contents

1 Introduction

Argumentation Theory is an interdisciplinary research field that deals with the analysis of arguments and their characteristics. The foundations of this field extend back to 500 BC and were influenced by philosophers such as Zeno of Elea, Plato, Socrates and Aristotle [39]. In the area of artificial intelligence the concept of *abstract argumentation frameworks (AAFs)*, introduced by Dung [18] in 1995, is well-studied and applied in various areas such as legal reasoning [36], machine learning [35] and multi-agent system research [12].

An abstract argumentation framework provides a formal representation of arguments and their relationships to each other, thereby making them computable, which is particularly helpful for complex argumentation situations. The basic structure of an AAF is represented by a directed graph, where the nodes of the graph represent the arguments and the directed edges represent the attacks between the arguments. Once an argumentation situation has been formalized as an AAF, important characteristics and relationships between the arguments can be examined. In the area of abstract argumentation, the primary focus is on finding sets of arguments that are acceptable under various semantics (admissible, complete, grounded, preferred, stable).

Abstract argumentation frameworks have been extended by a variety of variants to increase their functionality and expressiveness. *Bipolar argumentation frameworks (BAFs)* [14] for example provide a support relation in addition to the attack relation, while *extended argumentation frameworks (EAFs)* [34] allow attacks on attacks. *Preference-based argumentation frameworks (PAFs)* [2] and *value-based argumentation frameworks(VAFs)* [3] both extend AAFs by modeling preferences. In PAFs, this is done through the use of a preference relation between the arguments and in VAFs by assigning values to the arguments and preferring arguments based on a hierarchy of these values.

In this bachelor's thesis, the focus lies on *weighted argumentation frameworks (WAFs)* [10], with the capability to weight attacks by assigning a numerical value to each attack. By weighting attacks instead of arguments, the relations between arguments can be represented in a more nuanced and detailed way [10]. Figure 1 demonstrates an example of a WAF with two arguments attacking each other. This

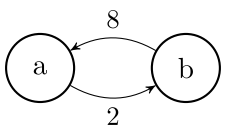


Figure 1: Example of a WAF with two arguments.

could be an argumentation between two hikers with the arguments a: “I am tired, we should rest.” and b: “It is getting dark, we should move on.” If only the arguments a and b are present, they would be both acceptable in traditional AAFs. In a WAF on the other hand, rational agents would weight the attack from b to a stronger, and therefore the argument b emerges as the only acceptable argument. Additionally, the weighting of attacks offers the possibility to attack multiple arguments with different weights from one argument. There exist several variants of WAFs that are based on AAFs and differ in their notion of defence or allow for relaxations of certain properties[10]. In this bachelor's thesis, we focus on the *semiring-based weighted argumentation frameworks* proposed by Bistarelli et al. [7], which feature a unique notion of defence and the ability to perform relaxation on the notion of defence, as well as allowing for inconsistency within an extension.

Since 2015, the *International Competition on Computational Models of Argumentation (ICCMA)* [38] has been held every two years, in which different programs and software systems compete in abstract argumentation reasoning. A regular contender in this competition is *Aspartix* [22], a software system that utilizes *answer set programming (ASP)*, a form of logic programming. Answer set programming allows for the semantics of an AAF to be declared in the form of rules and then solved automatically with an ASP solver. *Clingo* [26] is an example of such an ASP system, which is continuously being developed and can efficiently solve complex reasoning problems. The semiring-based WAFs from Bistarelli et al. [7] have already been implemented in *ConArg* [8]. However, there is currently no such implementation using ASP.

In this bachelor's thesis, we present an initial approach to implement semiring-based weighted argumentation frameworks using answer set programming. We introduce ASP-encodings for conflict-free sets, admissible sets, complete semantics, and stable semantics, which are compatible with the ASP system Clingo. Subsequently, we conduct an evaluation of the ASP-implementation, in which we address the following research questions:

Research Question 1: Impact of the Model
When utilizing different models for generating weighted argumentation frameworks, how do the instances behave in terms of acceptability under various semantics and problem types, and what is the corresponding runtime performance of the ASP-implementation?

Research Question 2: Impact of the Problem Type
To what extent do different problem types affect the runtime performance of the ASP-implementation?

Research Question 3: Impact of the Relaxations
What impact do relaxations of the properties conflict-freeness and the notion of defence have when analyzing weighted argumentation frameworks for acceptability, and what impact do the relaxations have on the runtime performance of the ASP-implementation?

Research Question 4: Impact of the Implementation
When examining weighted argumentation frameworks for acceptability under various problem types, how does the runtime performance of the ASP-implementation compare to that of other implementations?

In this work, we begin by presenting Dung's abstract argumentation frameworks [18], which also form the basis for the semiring-based weighted argumentation frameworks from Bistarelli et al. [7] presented in Section 3. We will introduce the core concepts and semantics of WAFs and also showcase the unique properties of this type of WAFs with several examples. In Section 4, we provide a brief introduction to answer set programming and explain its functionality in more detail. Section 5 is dedicated to related work, where we explore how the semiring-based approach for WAFs differs from other approaches in terms of the notion of defence, and additionally, we will examine the software systems ASPARTIX and ConArg more closely. In Section 6, we present the developed ASP-encodings for WAFs, which are compatible with the ASP system Clingo. In Section 7, we provide an evaluation of the ASP-implementation in which we address the research questions presented. Finally, in Section 8 we provide a summarizing conclusion and discuss directions for future research.

2 Abstract Argumentation Frameworks

In this section we present Dung's *abstract argumentation frameworks (AAFs)* [18] and associated definitions. The primary focus of abstract argumentation is to identify sets of arguments that are acceptable under various semantics. Key concepts in this context include the *notion of defence* and *conflict-freeness*, which serve as the foundational criteria for defining *admissible sets* of arguments. By imposing *constraints* on admissible sets, various semantics can be defined and corresponding *extensions* can be calculated. The definitions that follow in this section represent the concepts which where presented by Dung in his original paper [18].

2.1 Core Concepts

Abstract argumentation frameworks (AAFs) [18] allow for a formal representation of arguments and the attack relations between those arguments, thereby enabling an analysis of these AAFs for acceptability. Through the formal and robust representation of Dung's AAFs, we can examine complex argumentation situations under various semantics. The definition of an abstract argumentation framework is as follows:

Definition 1 ***(Abstract argumentation framework).*** *An* ***abstract argumentation framework (AAF)*** *is a pair $\langle \mathcal{A}, R \rangle$ consisting of a set of arguments $\mathcal{A}$ and a binary relation $R \subseteq \mathcal{A} \times \mathcal{A}$, called attack relation.*

For a given AAF $G = \langle \mathcal{A}, R \rangle$[1], we denote $\forall\ a_i, a_j \in \mathcal{A}$, $R(a_i, a_j)$ means that a_i attacks a_j. The formal definition of AAFs allows us to represent them as a directed graph as shown in Example 1.

Example 1 *Let $G_1 = \langle \mathcal{A}, R \rangle$ be an AAF, with the arguments $\mathcal{A} = \{a, b, c, d\}$ and the attacks $R = \{(a, b), (b, d), (c, d), (d, c)\}$. Figure 2 shows the AAF as a directed graph, with the arguments as nodes and the attacks as directed edges.*

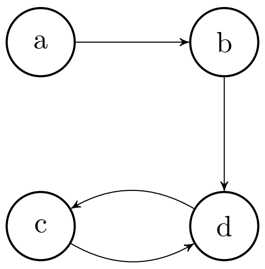


Figure 2: Example of an AAF G_1.

[1]We denote G for AAFs, as we use F for WAFs.

A prerequisite for acceptable sets of arguments is conflict-freeness, which means that the arguments within a set do not attack each other and therefore are internally consistent. The definition of *conflict-free sets* is as follows:

Definition 2 ***(Conflict-free sets).*** *A set $\mathcal{B} \subseteq \mathcal{A}$ is* ***conflict-free*** *iff no two arguments a and b in $\mathcal{B}$ exist such that a attacks b.*

Another crucial concept for the acceptability arguments is the notion of *defence*, which describes whether a set defends an argument by collectively defending the argument against all attacks.

Definition 3 ***(Defence).*** *An argument b is* ***defended*** *by a set $\mathcal{B} \subseteq \mathcal{A}$ (or $\mathcal{B}$ defends b) iff for any argument $a \in \mathcal{A}$, if $R(a, b)$ then there exists an argument $c \in \mathcal{B}$ such that $R(c, a)$.*

With the notion of defence, we can examine in Example 2, if an argument is defended by a set.

Example 2 *Consider the AAF G_1 from Figure 2. If we observe the argument d, we see that d defends itself against c but not against b. We can investigate whether d can be successfully defended together with other arguments. Since a attacks b, the set $\{a, d\}$ successfully defends against the arguments b and c.*

2.2 Semantics for Abstract Argumentation Frameworks

Since we have defined the core concepts of AAFs, we can now define semantics for AAFs, which offer the possibility to identify sets of arguments that are acceptable under the respective semantics. We denote σ to the semantics, where $\sigma \in \{adm, com, grd, prf, stb\}$ corresponds respectively to admissible, complete, grounded, preferred and stable semantics. For an AAF $G = \langle \mathcal{A}, R \rangle$, we denote with $\sigma(G)$ the set of σ-extensions of G.

First, we define *admissible sets*, which are considered the fundamental form of acceptable sets, which can defend themselves against all attacks and are conflict-free.

Definition 4 ***(Admissible sets).*** *A conflict-free set $\mathcal{B} \subseteq \mathcal{A}$ is* ***admissible*** *iff each argument in $\mathcal{B}$ is defended by $\mathcal{B}$.*

In Example 3, we determine the admissible sets of the AAF from Figure 2.

Example 3 *Consider the AAF G_1 from Figure 2. For the admissible sets we get $adm(G_1) = \{\emptyset, \{a\}, \{c\}, \{a, c\}, \{a, d\}\}$. If we examine the sets $\{a, c\}$ and $\{a, d\}$, we see that $\{a, c\}$ is admissible because a is not attacked and therefore defended and c defends itself from b. The set $\{a, d\}$ is admissible, because a defends d from b and d defends itself from c. The empty set $\emptyset$ is always admissible, because it is conflict free and has no arguments that can be attacked and therefore always satisfies the notion of defence.*

We can now define various semantics, which impose *constraints* on admissible sets. With these semantics we can calculate subsets of admissible sets, which we refer to as *extensions*. *Complete extensions* are admissible and include all arguments that are defended by the extension, which according to Dung [18, p. 329] describes a confident rational agent, who believes in everything he can defend. The *grounded extension* represents the minimal complete extension and is always unique. The *preferred extensions* are admissible sets, that are maximal with respect to set inclusion. The *stable extensions* are admissible and attack all arguments that are not included in the extension. Therefore stable extensions are able to defeat all arguments that are outside the extension.

Definition 5 ***(Semantics for AAFs).*** *Let $G = \langle \mathcal{A}, R \rangle$ be an AAF. A subset of arguments $\mathcal{B} \subseteq \mathcal{A}$ is then*

- ***complete**, iff it is admissible and each argument which is defended by $\mathcal{B}$ is in $\mathcal{B}$;*
- ***grounded**, iff it is complete and minimal w.r.t. set inclusion;*
- ***preferred**, iff it is admissible and maximal w.r.t. set inclusion;*
- ***stable**, iff it is admissible and for each argument which is not in $\mathcal{B}$, there exists an argument in $\mathcal{B}$ that attacks it.*

Since we have defined the semantics for AAFs, we can now examine the AAF from Figure 2 for acceptability. The respective admissible sets were already examined in Example 3.

Example 4 *Consider the AAF G_1 from Figure 2. For the extensions under the respective semantics we get:*

– *$com(G_1) = \{\{a\}, \{a, c\}, \{a, d\}\}$. The sets $\emptyset$ and $\{c\}$ are admissible, but not complete. $\emptyset$ is not complete, since a is not attacked, and therefore defended by $\emptyset$. $\{c\}$ is not complete, since a is not attacked and thus defended by $\emptyset$, which is why c is only complete in combination with a.*

– *$grd(G_1) = \{\{a\}\}$, since $\{a\}$ is the minimal complete extension.*

– *$prf(G_1) = \{\{a, c\}, \{a, d\}\}$, since these extensions are the maximal admissible sets.*

– *$stb(G_1) = \{\{a, c\}, \{a, d\}\}$, since these extensions are admissible and attack all arguments that are not in the respective extension.*

While the examples shown in this section are still relatively simple, the formal structure of AAFs also enables the examination of large and complex argumentation situations. The semiring-based WAFs introduced in the following section are expansions of AAFs and allow for a weighting of attack relations with a composition of weights and the relaxation of the properties conflict-freeness and notion of defence.

3 Weighted Argumentation Frameworks

In this section we present the *semiring-based weighted argumentation frameworks (WAFs)* from Bistarelli et al. [7] and associated definitions. First, we present the mathematical structure for these types of WAFs, the *c-semiring* [5], followed by the expansion of the definitions known from abstract argumentation frameworks to account for a weighting of the attacks. Subsequently, we present relaxations for conflict-free sets and the notion of defence with the parameters α and γ, which leads to the definitions of α^γ-semantics. The definitions in this section were sourced from the original paper by Bistarelli et al. [7] and from a subsequent paper by Bistarelli and Taticchi [11].

3.1 Core Concepts

Weighted argumentation frameworks extend AAFs with a weighting of the attacks, thereby enabling a more nuanced and detailed representation of the relations between arguments. There are several approaches for weighted argumentation frameworks [19, 16, 33, 7], which differ in their notion of defence and the possibility to perform relaxations. In Section 5, we will explore these approaches in more detail. In this work, we will focus on the *semiring-based weighted argumentation frameworks (WAFs)* from Bistarelli et al. [7], which introduce an extended notion of defence and the possibility to perform relaxations on conflict-freeness and the notion of defence.

The WAFs from Bistarelli et al. [7] are based on the *c-semiring* [5], an algebraic structure that provides an operation for composing weights and therefore allows us to compare the composition of the attacks with the composition of the defences to determine the acceptability of an argument. The definition of the c-semiring is as follows:

Definition 6 ***(c-semirings).*** *A commutative semiring is a tuple $\mathbb{S} = \langle \mathcal{S}, \oplus, \otimes, \bot, \top \rangle$ such that $\mathcal{S}$ is a set, $\top, \bot \in \mathcal{S}$ and $\oplus, \otimes : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ are binary operators making the triples $\langle \mathcal{S}, \oplus, \bot \rangle$ and $\langle \mathcal{S}, \otimes, \top \rangle$ commutative monoids (semi-groups with identity), satisfying*

(i) $\forall s, t, u \in \mathcal{S}. s \otimes (t \oplus u) = (s \otimes t) \oplus (s \otimes u)$ (distributivity), and

(ii) $\forall s \in \mathcal{S}. s \otimes \bot = \bot$ (annihilator).

If $\forall s, t \in \mathcal{S}.\ s \oplus (s \otimes t) = s$, the semiring is said to be absorptive. In short, ***c-semirings*** *are defined as commutative and absorptive semirings.*

Building upon the c-semiring, we can define *semiring-based weighted argumentation frameworks (WAFs)*. The instance of the c-semiring, that is applied for WAFs is the weighted c-semiring: $\mathbb{S}_{weighted} = \langle \mathbb{R}^+ \cup \{\infty\}, min, +, \infty, 0 \rangle$.

Definition 7 ***(WAF).*** *A* ***semiring-based weighted argumentation framework (WAF)*** *is a quadruple $\langle \mathcal{A}, R, W, \mathbb{S} \rangle$, where $\mathbb{S}$ is a c-semiring $\langle \mathcal{S}, \oplus, \otimes, \bot, \top \rangle$, $\mathcal{A}$ is a* *set of arguments, R the binary attack-relation on $\mathcal{A}$ and $W : \mathcal{A} \times \mathcal{A} \to \mathcal{S}$ is a binary function. Given $a, b \in \mathcal{A}$ and $R(a, b)$, then $W(a, b) = s$ means that a attacks b with a weight $s \in \mathcal{S}$. Moreover, we require that $R(a, b)$ iff $W(a, b) > \top$*[2].

Since we have defined WAFs, we can now expand the AAF from Figure 2 to a WAF by assigning weights to the attacks.

Example 5 *Let $F_1 = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$ be a WAF with the arguments $\mathcal{A} = \{a, b, c, d\}$, the attacks $R = \{(a, b), (b, d), (c, d), (d, c)\}$, the weights $W(a, b) = 2$, $W(b, d) = 4$, $W(c, d) = 5$, $W(d, c) = 3$ and the weighted c-semiring $\mathbb{S} = \langle \mathbb{R}^+ \cup \{\infty\}, min, +, \infty, 0 \rangle$. Figure 3 shows the WAF as a directed graph with the weighted attacks.*

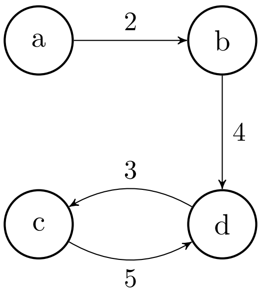


Figure 3: Example of a WAF F_1.

In WAFs, we calculate the sum of the weights of the attacks from one set of arguments to another with *w-attacks*. For this functionality, we define the $\bigotimes$ operator, which extends the $\otimes$ operator from $\mathbb{S}$ to sets of values.

Definition 8 ***(w-attack).*** *Given a WAF, $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$. A set of arguments $\mathcal{B} \subseteq \mathcal{A}$* ***w-attacks*** *a set of arguments $\mathcal{D} \subseteq \mathcal{A}$ and the weight of such an attack is $k \in \mathcal{S}$, if*

$$W(\mathcal{B}, \mathcal{D}) = \bigotimes_{b \in B, d \in D} W(b, d) = k.$$

The definition for w-attacks also allows for a composition of the attacks from a single argument to a set of arguments and from a set of arguments to a single argument.

Example 6 *When examining the WAF from Figure 3, we find for example that the total weight of the attack from $\{b, c\}$ to d is $W(\{b, c\}, d) = 9$ and the total weight of the attack from $\{a, c\}$ to $\{b, d\}$ is $W(\{a, c\}, \{b, d\}) = 7$.*

We can now define *w-conflict-free sets*, which have the same characteristic as the conflict-free sets from Section 2, since no attacks within w-conflict-free sets are allowed.

[2]In the context of a weighted c-semiring $\top$ represents 0.

Definition 9 ***(w-conflict-free sets).*** *Given a WAF, $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$. Then a set $\mathcal{B} \subseteq \mathcal{A}$ is **w-conflict-free** iff $W(\mathcal{B}, \mathcal{B}) = \top$.*

By defining the notion of *w-defence*, which builds upon w-attacks, we can determine if an argument is defended by a set against an attacking arguments. This is a fundamental prerequisite for determining acceptable sets of arguments for WAFs.

Definition 10 ***(w-defence).*** *Given a WAF, $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$. Then a set $\mathcal{B} \subseteq \mathcal{A}$ **w-defends** $b \in \mathcal{A}$ iff $\forall a \in \mathcal{A}$ such that $R(a, b)$, we have that $W(a, \mathcal{B} \cup \{b\}) \leq W(\mathcal{B}, a)$.*

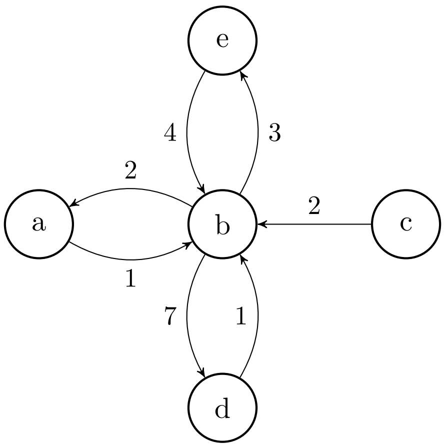


Figure 4: Example of a WAF F_2 for basic definitions.

In the Examples 7 and 8, we examine the WAF F_2 from Figure 4 for w-defence, by investigating whether the arguments a or d are w-defended by the set $\{c, e\}$.

Example 7 *Consider the WAF F_2 from Figure 4 with $\mathcal{B} = \{c, e\}$. By examining if the argument a is w-defended by $\mathcal{B}$, we get $W(b, \mathcal{B} \cup \{a\}) = 3 + 2 = 5$ and $W(\mathcal{B}, \{b\}) = 4 + 2 = 6$. $W(b, \mathcal{B} \cup \{a\}) \leq W(\mathcal{B}, b)$ is true and thus the argument a is successfully w-defended by $\mathcal{B}$.*

Example 8 *Consider the WAF F_2 from Figure 4 with $\mathcal{B} = \{c, e\}$. By examining if the argument d is w-defended by $\mathcal{B}$ against b, we get $W(b, \mathcal{B} \cup \{d\}) = 7 + 3 = 10$ and $W(\mathcal{B}, \{b\}) = 4 + 2 = 6$. We see, that the weight from the attacking argument is now higher than the weight of the defence from $\mathcal{B}$, thus the argument d is not w-defended by $\mathcal{B}$, since $W(b, \mathcal{B} \cup \{a\}) \leq W(\mathcal{B}, b)$ is false.*

WAFs can be relaxed by applying parameters that adjust the notions of w-defence and w-conflict-freeness to allow for more flexibility in the analysis of acceptable sets. Practical examples include removing the effect of troll arguments or analyzing product reviews in more detail by allowing for inner conflict within an extension [7, p. 80-83].

To allow for inconsistency within an extension, we expand w-conflict-free sets with the parameter α, to tolerate attacks within w-conflict-free sets up to a given value, which leads to the definition of *α-conflict-free sets.*

Definition 11 ***(α-conflict-free sets).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$. Then a set $\mathcal{B} \subseteq A$ is **α-conflict-free** iff $W(\mathcal{B}, \mathcal{B}) \leq \alpha$.*

We can now examine the WAF F_2 from Figure 4 for α-conflict-freeness by adjusting the parameter α to allow for inconsistency within a set.

Example 9 *Consider the WAF F_2 from Figure 4 with the parameter $\alpha = 5$ and $\mathcal{B} = \{a, b, c\}$. The set $\mathcal{B}$ is α-conflict-free, since $(W(\mathcal{B}, \mathcal{B}) = 5) \leq 5$.*

We now define *γ-defence*, which is a relaxed notion of w-defence through the parameter γ. This relaxation enables the identification of a broader set of acceptable arguments.

Definition 12 ***(γ-defence).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} = \langle \mathcal{S}, \oplus, \otimes, \bot, \top \rangle \rangle$ and $\gamma \in S$. A subset of arguments $\mathcal{B} \subseteq \mathcal{A}$ **γ -defends** $b \in \mathcal{A}$ iff $\forall a \in \mathcal{A}$ such that $R(a, b)$ we have that $W(\mathcal{B}, a) > \top$ and $W(a, \mathcal{B} \cup \{b\}) \leq (W(\mathcal{B}, a)) + \gamma)$.*

We can now examine the WAF F_2 from Figure 4 for γ-defence by adjusting the parameter γ to relax the notion of w-defence.

Example 10 *Consider the WAF F_2 from Figure 4 with the parameter $\gamma = 5$ and $\mathcal{B} = \{c, e\}$. By examining the argument d for γ-defence, we get $W(b, \mathcal{B} \cup \{a\}) = 7 + 3 = 10$ and $W(\mathcal{B}, \{b\}) = 4 + 2 = 6$. $W(a, \mathcal{B} \cup \{b\}) \leq (W(\mathcal{B}, a)) + \gamma)$ is true, since $\gamma = 4$, and thus the argument d is successfully 4-defended by $\mathcal{B}$.*

3.2 Semantics for Weighted Argumentation Frameworks

Building on the core concepts, we can now define semantics for WAFs, which also offer the possibility of relaxation through the parameters α and γ. We denote σ_r to the α^γ-semantics, where $\sigma_r \in \{\alpha^\gamma\text{-}adm, \alpha^\gamma\text{-}com_{\mathbb{C}_1}, \alpha^\gamma\text{-}com_{\mathbb{C}_2}, \alpha^\gamma\text{-}com_{\mathbb{C}_3}, \alpha^\gamma\text{-}prf, \alpha^\gamma\text{-}stb\}$ corresponds respectively to α^γ-admissible, α^γ-complete$_{\mathbb{C}_1}$, α^γ-complete$_{\mathbb{C}_2}$, α^γ-complete$_{\mathbb{C}_3}$, α^γ-preferred and α^γ-stable semantics.

We first define *α^γ-admissible sets*, which provide the foundation for examining arguments for acceptability in WAFs.

Definition 13 ***(α^γ-admissible sets).*** *An α-conflict-free set $\mathcal{B} \subseteq \mathcal{A}$ is **α^γ-admissible** iff the arguments in $\mathcal{B}$ are γ-defended by $\mathcal{B}$ from the arguments in $\mathcal{A} \setminus \mathcal{B}$.*

In Example 11, we investigate the WAF from Figure 4, with the focus on examining the effects of the relaxation parameters α and γ.

Example 11 *Consider the WAF F_2 from Figure 4 with the parameters $\alpha = 2$, $\gamma = 1$ and $\mathcal{B} = \{b, c\}$. Without relaxations, $\mathcal{B}$ would be neither α-conflict-free since c attacks b nor γ-defended since $\{b, c\}$ cannot defend against e. Due to the parameter α, the set $\mathcal{B}$ is 2-conflict-free and due to the parameter γ the set $\mathcal{B}$ can successfully defend against e, since $W(e, \mathcal{B}) = 4$ and $W(\mathcal{B}, e) + \gamma = 3 + 1$ with the condition for γ-defence $W(e, \mathcal{B}) <= W(\mathcal{B}, e) + \gamma$. Therefore, the set $\mathcal{B}$ is 2^1-admissible.*

For the α^γ-complete semantics, there exist several approaches. The first approach, presented in the original paper from Bistarelli et al. [7], is oriented towards the classical definition for complete semantics from Dung. In this semantics, although it is taken into account whether an argument is defended by a set, the weight of the counterattack from the argument to an attacking argument is not considered. The definition for *α^γ-complete$_{\mathbb{C}_1}$ semantics* is as follows:

Definition 14 ***(α^γ-complete$_{\mathbb{C}_1}$ semantics).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B} \subseteq \mathcal{A}$ is **α^γ-complete$_{\mathbb{C}_1}$** iff each argument $b \in \mathcal{A}$ that is $\boldsymbol{\gamma}$ -defended by $\mathcal{B}$ is in $\mathcal{B}$ and $W(\mathcal{B} \cup \{b\}, \mathcal{B} \cup \{b\}) \leq \alpha$.*

The second definition for *α^γ-complete$_{\mathbb{C}_2}$ semantics* is derived from [11]. The definition is slightly adjusted, as the relaxation parameters were also considered. In this definition, an argument no longer needs to be actively defended by $\mathcal{B}$, as an argument can also defend itself independently against attacking arguments.

Definition 15 ***(α^γ-complete$_{\mathbb{C}_2}$ semantics).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B} \subseteq \mathcal{A}$ is **α^γ-complete$_{\mathbb{C}_2}$** iff each argument $b \in \mathcal{A}$ such that $\mathcal{B} \cup \{b\}$ is α^γ-admissible belongs to $\mathcal{B}$.*

The third definition for α^γ-complete semantics is similar to Definition 15, but with the additional constraint that an argument, which is not in $\mathcal{B}$, cannot defend itself against attacking arguments without $\mathcal{B}$. In this semantics, $\mathcal{B}$ is only α^γ-complete when it contains all arguments that are defended by $\mathcal{B}$, with the weight of counterattacks from arguments against attacking arguments also taken into account. This definition also matches the functionality of ConArg [8], introduced in Section 5 and therefore we will proceed with this definition for the remainder of this work. The definition for *α^γ-complete$_{\mathbb{C}_3}$ semantics* is as follows:

Definition 16 ***(α^γ-complete$_{\mathbb{C}_3}$ semantics).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B} \subseteq \mathcal{A}$ is **α^γ-complete$_{\mathbb{C}_3}$** iff each argument $b \in \mathcal{A}$ such that $\mathcal{B} \cup \{b\}$ is w-admissible belongs to $\mathcal{B}$ under the condition that for every $c \in \mathcal{A} \setminus \mathcal{B}$ with $W(c, b) \neq \top$ we have that $W(\mathcal{B}, c) \neq \top$.*

In Example 12, we demonstrate the distinctions between these three types of α^γ-complete semantics by analyzing the WAF from Figure 5.

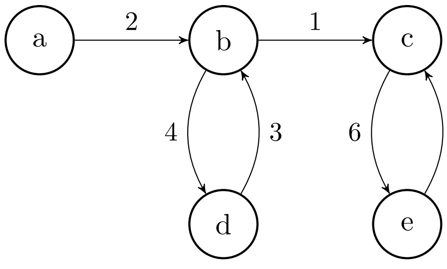


Figure 5: Example of a WAF F_3 for α^γ-complete semantics.

Example 12 *Consider the WAF F_3 from Figure 5 with the parameters $\alpha = 0$, $\gamma = 0$. For the 0^0-complete extensions, under the respective semantics we get:*

- 0^0*-com*$_{\mathbb{C}_1}(F_3) = \{\{a\}, \{a, c\}, \{a, c, d\}\}$*. $\{a\}$ is 0^0-complete$_{\mathbb{C}_1}$, since a neither 0^0-defends d against b nor 0^0-defends e against c. $\{a, c\}$ is 0^0-complete, as the set does not 0^0-defend d against b.*
- 0^0*-com*$_{\mathbb{C}_2}(F_3) = \{\{a, c, d\}\}$*. The only 0^0-complete$_{\mathbb{C}_2}$ extension is $\{a, c, d\}$. $\{a\}$ and $\{a, c\}$ are not 0^0-complete$_{\mathbb{C}_2}$, since the weight from d to b is taken into account. However, $\{a, d\}$ is also not 0^0-complete$_{\mathbb{C}_2}$, since c can defend itself independently against e without the necessity of $\{a, d\}$ attacking e.*
- 0^0*-com*$_{\mathbb{C}_3}(F_3) = \{\{a, d\}, \{a, c, d\}\}$*. With this semantics, $\{a, d\}$ and $\{a, c, d\}$ are both 0^0-complete$_{\mathbb{C}_3}$. $\{a, d\}$ is 0^0-complete$_{\mathbb{C}_3}$ since c is not attacked by $\{a, d\}$.*

As shown in Example 12, we see that the α^γ-complete$_{\mathbb{C}_1}$ semantics does not take into account the defence weight from individual arguments outside $\mathcal{B}$ against attacking arguments and the α^γ-complete$_{\mathbb{C}_2}$ semantics is too general. With the α^γ-complete$_{\mathbb{C}_3}$ semantics, the attack weight of individual arguments outside $\mathcal{B}$ against attacking arguments is considered and all attacking arguments must be attacked by $\mathcal{B}$ to defend an argument outside $\mathcal{B}$.

We now define α^γ*-preferred semantics*, which represent the maximal α^γ-admissible sets with respect to set inclusion.

Definition 17 ***(α^γ-preferred semantics).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B} \subseteq \mathcal{A}$ is* ***α^γ-preferred*** *iff $\mathcal{B}$ is maximal (with respect to set inclusion).*

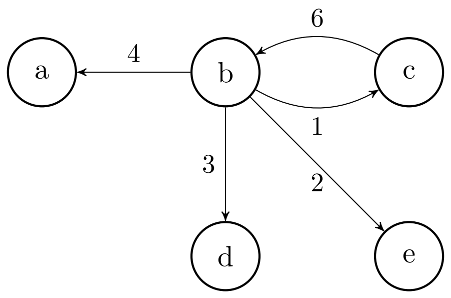


Figure 6: Example of a WAF F_4 for α^γ-preferred semantics.

In Example 13, we examine the WAF from Figure 6 under α^γ-preferred semantics. This example is particularly interesting since maximal preferred extensions of different sizes can be found.

Example 13 *Consider the WAF F_4 from Figure 6 with the parameters $\alpha = 0$, $\gamma = 0$. For the 0^0-preferred extensions, we get $\{a, c\}$ and $\{c, d, e\}$. These are the maximal 0^0-admissible sets with the respect to set inclusion, since $\{a\} \not\subset \{c, d, e\}$.*

In Example 14, we examine again the WAF from Figure 6 under α^γ-preferred semantics, but this time with a relaxation through the parameter γ.

Example 14 *Consider the WAF F_4 from Figure 6 with the parameters $\alpha = 0$, $\gamma = 1$. For the 0^1-preferred extensions, we get $\{a, c, e\}$ and $\{c, d, e\}$, because $\{a, c, e\}$ now successfully 1-defends against c, since $W(b, \{a, c, e\}) = 7$ and $W(\{a, c, e\}, b) + \gamma = 6 + 1 = 7$.*

As the last semantics dealt with in this work, we define *α^γ-stable semantics*, which state that an α^γ-stable extension must be α^γ-complete$_{\mathbb{C}_3}$ and additionally all arguments outside of the extensions must be attacked.

Definition 18 ***(α^γ-stable semantics).*** *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B} \subseteq \mathcal{A}$ is **α^γ-stable** iff for every $a \in \mathcal{A} \setminus \mathcal{B}$ there is a $b \in \mathcal{B}$ with $W(b, a) \neq \top$ and $\mathcal{B} \cup \{a\}$ is not α^γ-admissible.*

In the Examples 15 and 16, we examine the WAF from Figure 7 under α^γ-stable semantics. Without relaxations, the WAF would have no stable extensions. In Example 15, we allow for inner conflict through the parameter α. In Example 16, we relax the notion of w-defence with the parameter γ, which leads to the identification of another α^γ-stable extension.

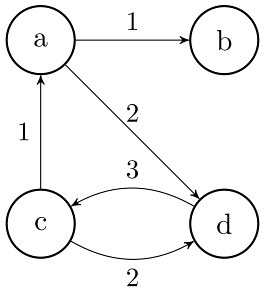


Figure 7: Example of a WAF F_5 for α^γ-stable semantics.

Example 15 *Consider the WAF F_5 from Figure 7 with the parameters $\alpha = 1$, $\gamma = 0$. For the 1^0-stable extensions, we get $\{a, c\}$, since $\{a, c\}$ is 1-conflict free, 0-defends with $W(\{a, c\}, d) = 4$ against $W(d, \{a, c\}) = 3$ and $\{a, c\}$ also attacks b.*

Example 16 *Consider the WAF F_5 from Figure 7 with the parameters $\alpha = 0$, $\gamma = 1$. For the 0^1-stable extensions, we get $\{b, c\}$, since $\{a, c\}$ is 0-conflict free, 1-defends with $W(\{b, c\}, d) + \gamma = 2 + 1 = 3$ against $W(d, \{b, c\}) = 3$ and with $W(\{b, c\}, a) + \gamma = 1 + 1 = 2$ against $W(a, \{b, c\}) = 1$.*

Since we have now defined all semantics, we can fully investigate the WAF from Figure 8 for acceptability, as we have already done in Section 2 with the respective AAF from Figure 2. The α^γ-grounded semantics is not covered in this work, as in the original paper by Bitarelli et al. [7], no definition for these semantics was presented.

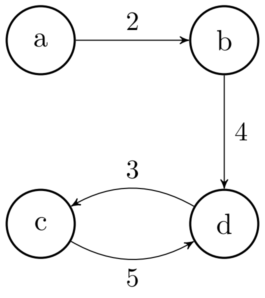


Figure 8: Reintroduction of the WAF F_1 to calculate the α^γ-extensions with various parameters.

Example 17 *Consider the WAF F_1 from Figure 8 with the parameters $\alpha = 0$ and $\gamma = 0$. By calculating the 0^0-extensions, we get:*

- 0^0-$adm(F) = \{\emptyset, \{a\}, \{c\}, \{a, c\}\}$.
- 0^0-$com_{\mathbb{C}_3}(F) = \{\{a\}, \{a, c\}\}$.
- 0^0-$prf(F) = \{\{a, c\}\}$.

– 0^0-*stb*$(F) = \{\{a, c\}\}$.

In the respective AAF without weights, $\{a, d\}$ would be admissible, complete, preferred and stable. Due to the weighting of the attacks, $\{a, d\}$ can neither 0-defend against c nor against b, since $W(c, \{a, d\}) > W((\{a, d\}), c)$ with $W(c, \{a, d\}) = 5$, $W(\{a, d\}), c) = 3$ and $W(b, \{a, d\}) > W(\{a, d\}, b)$ with $W(b, \{a, d\}) = 4, W(\{a, d\}, b) = 2$. As a result, the set $\{a, d\}$ is not 0^0-admissible and therefore also not 0^0-complete$_{\mathbb{C}_3}$, 0^0-preferred or 0^0-stable.

In Example 18 we calculate the extensions of the WAF from Figure 8 by adjusting the parameter α, which allows for inner conflict within an extension.

Example 18 *Consider the WAF F_1 from Figure 8 with the parameters $\alpha = 2$ and $\gamma = 0$. By calculating the 2^0-extensions, we get:*

– 2^0-*adm*$(F) = \{\emptyset, \{a\}, \{c\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$.

– 2^0-*com*$_{\mathbb{C}_3}(F) = \{\{a, b, c\}\}$.

– 2^0-*prf*$(F) = \{\{a, b, c\}\}$.

– 2^0-*stb*$(F) = \{\{a, b, c\}\}$.

The set $\{a, b, c\}$ is 2^0-admissible, since the attack from a to b is relaxed, resulting in $W(\{a, b, c\}, \{a, b, c\}) \leq \alpha$ with $W(\{a, b, c\}, \{a, b, c\}) = 2$ and $\alpha = 2$. $\{a, b, c\}$ is also the only 2^0-complete$_{\mathbb{C}_3}$ extension, because the sets $\{a, b\}$ and $\{a, c\}$ are not 2^0-complete$_{\mathbb{C}_3}$, since the attack from a to b is relaxed through the parameter α and b 0-defends c against d.

In Example 19 we calculate the extensions of the WAF from Figure 8 by adjusting the parameter γ, to relax the notion of w-defence:

Example 19 *Consider the WAF F_1 from Figure 8 with the parameters $\alpha = 0$ and $\gamma = 2$. By calculating the 0^2-extensions, we get:*

– 0^2-*adm*$(F) = \{\emptyset, \{a\}, \{c\}, \{a, c\}, \{a, d\}\}$.

– 0^2-*com*$_{\mathbb{C}_3}(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$.

– 0^2-*prf*$(F) = \{\{a, c\}, \{a, d\}\}$.

– 0^2-*stb*$(F) = \{\{a, c\}, \{a, d\}\}$.

Through relaxing the γ-defence with γ = 2, the set $\{a, d\}$ can successfully defend itself against the attacks from a and c, since $(W(b, \{a, d\}) - W(\{a, d\}, b)) \leq 2$ and $(W(c, \{a, d\}) - W(\{a, d\}, c)) \leq 2$, resulting in the same admissible sets and extensions as in the respective AAF without weights.

4 Answer Set Programming

In this section, we introduce *answer set programming (ASP)*, a form of logic programming and explore its functionality. In addition, we provide an overview of the software system Clingo [26], focusing on its operation and structure and conclude with a simple example to illustrate the principles and capabilities of ASP.

Answer set programming, is a type of declarative programming that originated from logic programming and non-monotonic reasoning. The history of ASP dates back to the late 1980s and early 1990s when it was first introduced as a form of logic programming by Vladimir Lifschitz and Michael Gelfond, aimed to address complex problem-solving tasks [32, p. 5].

In answer set programming, a problem is declared as a logic program in the form of rules, which consist of a head and a body. An ASP rule might be:

```
millennial(P) :- born(P, Y), Y >= 1981, Y =< 1996.
```

This rule reads as "A person P is a millennial if P is born between 1981 and 1996". Here `millennial(P)` is the head of the rule and `born(P, Y), Y >= 1981, Y =< 1996` is the body. The body specifies a condition that must be true for the head to be considered true. For more complex problems, ASP is equipped with additional types of rules such as constraints or choice rules.

The benefit of using ASP is that once a problem is formulated as a logic program, an ASP solver can then process it efficiently. First a given problem is modeled as a logic program. Then, the typical method is to initially convert the logic program into a variable-free ground format using a *grounder* and then compute the *stable models* with a *solver*. The stable models, also known as *answer sets*, provide one or multiple solutions to the given problem. These stable models can then be interpreted to get the solution for the problem.

Figure 9, sourced from [30], illustrates the workflow of answer set programming within the software-system *Clingo* [26], which combines the ASP grounder *Gringo* [28] and the ASP solver *Clasp* [27].

The declarative nature of ASP makes it applicable to a wide range of applications, especially in areas with a high degree of knowledge representation. In [25], some use cases are addressed where ASP is applied, including areas such as robotics, computational biology, bioinformatics and industrial applications. The software system ASPARTIX [22], presented in the next section, utilizes ASP to solve a wide range of problems in the area of abstract argumentation.

Listing 1 shows an example of an ASP-encoding that was extracted from [32], which calculates all prime numbers from 1 to the input value n using Clingo.

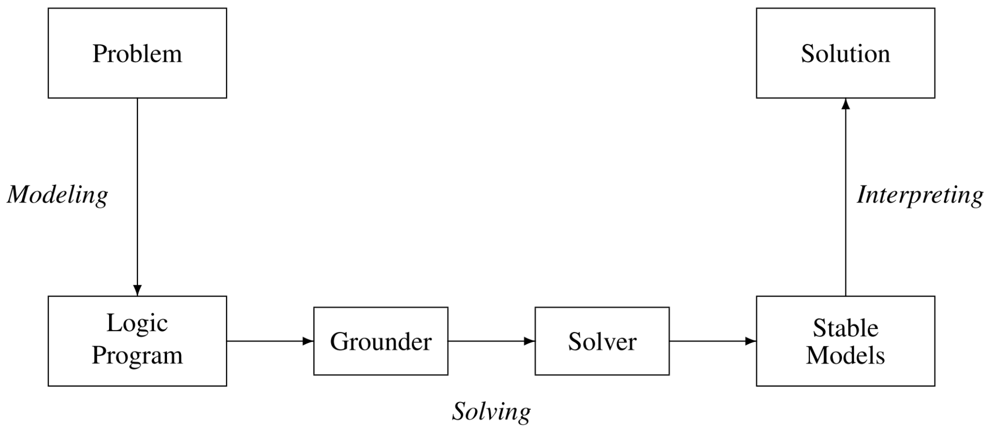


Figure 9: The workflow of ASP [30].

Listing 1 ASP-encoding for Prime numbers

```
1   % Prime numbers from 1 to n.
2
3   % input: positive integer n.
4
5   composite(N) :- N = 1..n, I = 2..N-1, N\I = 0.
6   % achieved: composite(N) iff N is a composite number from
7   % {1,...,n}.
8
9   prime(N) :- N = 2..n, not composite(N).
10  % achieved: prime(N) iff N is a prime number from {1,...,n}.
11
12  #show prime/1.
```

Below, we describe in more detail the individual components from Listing 1 for the calculation of prime numbers within the range `1..N`.

- **Line 5:** All composite numbers `N` within the range `1..n` are identified. The expression `N\I = 0` utilizes the modulo operation, which calculates the remainder when `N` is divided by `I = 2..N-1`. If there is no remainder `N` is divisible by `I` and therefore, `N` is identified as a composite number.

- **Line 9:** A number `N` is considered a prime number if it is not composite. This is determined using `not composite(N)`. This rule filters out all numbers that have been identified as composite, thereby leaving behind numbers that are prime. The specified range for `N` is `2..n`, as the number 1 is not considered a prime number by definition.

– **Line 12:** The `show prime/1` directive signals to Clingo that the output should include the results of the `prime/1` predicate. The `1` notation indicates that `prime` is a unary predicate, meaning it takes one argument.

In Example 20, we analyze the ASP-encoding for prime numbers from Listing 1 with the input n=8. The program is executed by the command ***clingo primes.lp -c n=8***.

Example 20 *For the input n=8, Clingo first identifies all composite numbers between* 1 *and* 8*, which in this instance are* $4, 6$ *and* 8*. Clingo proceeds to identify all numbers which fulfill the condition of being prime. In this case, these are all numbers from* 2 *to* 8 *that are not composite and thus the output is* $2, 3, 5$ *and* 7 *for prime.*

This example is primarily intended to illustrate the core functionalities of ASP, however, the application scope of ASP is significantly wider. Lifschitz [32] provides a solid introduction to this topic and shows a variety of practical examples and application areas. In Section 5, we will present an example from the software system ASPARTIX[22] that demonstrates how ASP can be applied in the field of abstract argumentation.

5 Related Work

In this section, we demonstrate how the semiring-based weighted argumentation frameworks from Bistarelli et al. [7] differ from other approaches by examining the notion of defence for the respective WAFs. Subsequently, we introduce ASPARTIX [22] and ConArg [8], which are well-established software systems in the field of abstract argumentation. ASPARTIX solves argumentation problems using answer set programming and supports various semantics. ConArg utilizes constraint programming and is particularly interesting as a reference system because it supports the WAFs from Bistarelli et al. [7].

5.1 Various Approaches for Weighted Argumentation Frameworks

There are various approaches for weighted argumentation frameworks with weights on attacks, which can be distinguished between *qualitative frameworks* [19, 29, 16] with preference relations over attacks and *quantitative frameworks* [33, 13, 7] with a specific numeric value for each attack [10]. The semiring-based WAFs from Bistarelli et al. [7] include an extended notion of defence, which enables the comparison of the cumulative weight of attacks. A detailed explanation on this topic can be found in [10], from which the example shown in this section is derived. Comparisons between the notions of defence are made with the WAFs from Martinez et al. [33] and Coste-Marquis et al. [16].

The main difference in the notions of defence is that in the semiring approach from Bistarelli et al. [7] the cumulative weight of multiple attacks is taken into account, while in the approaches of Martinez et al. [33] and Coste-Marquis et al. [16], only the weights of individual attacks are considered. In Example 21, sourced from [10], we demonstrate these differences by examining the WAF from Figure 10.

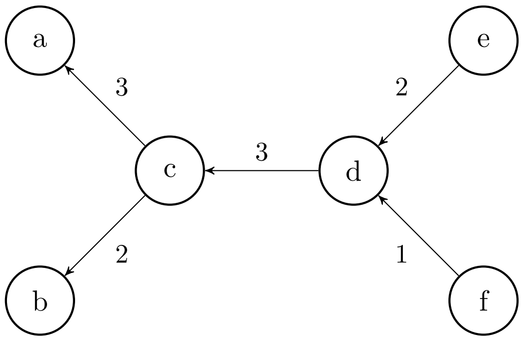


Figure 10: Example WAF for various notions of w-defence[10].

Example 21 *Consider the WAF from Figure 10. The argument c is successfully w-defended by the set $\{e, f\}$ under the approach of Bistarelli et al., using the semirings, since $W(d, c) = 3$ and $W(\{e, f\}, d) = 3$ and thus $W(d, c) <= W(\{e, f\}, d)$ is true. Under the approaches of Martinez et al. and Coste-Marquis et al., the cumulative* *weight is not considered, instead, only the strongest argument for defence counts. In this case, it would be the argument e, which attacks d with the weight of* 2*. Since the attack from d to c has a value of 3, the weight of the attack from e to d is not sufficient to successfully defend c against d. Another characteristic of the w-defence from Bistarelli et al. can be seen when we examine whether d w-defends the set* $\{a, b\}$ *against the attacks from c. Under the approaches of Martinez et al. and Coste-Marquis et al.,* $\{a, b\}$ *would be successfully defended, since again only the individual attacks are considered and thus the argument d successfully defends* $\{a, b\}$ *against c. Under the approach from Bistarelli et al., the set* $\{a, b\}$ *is not w-defended by d, since* $W(c, \{a, b\}) = 5$ *and* $W(d, c) = 3$ *and thus the condition for w-defence* $W(c, \{a, b\}) <= W(d, c)$ *is not met.*

As demonstrated in Example 21, through the semiring-based WAFs of Bistarelli et al., it is possible to consider cumulative attacks, thereby enabling an extended notion of w-defence. This allows for a more detailed examination of WAFs with respect to acceptability.

5.2 Aspartix

ASPARTIX [22] is a software system that utilizes ASP to solve reasoning problems in the area of abstract argumentation. Initiated at the Vienna University of Technology in 2008, ASPARTIX has been under continuous development and has established itself as the reference ASP system in the field of abstract argumentation [17].

Figure 11, sourced from [21], illustrates the basic workflow of ASPARTIX. The modular design of the system allows for the selection of specific ASP-encodings, which can be fed along with the input-AAF to an ASP solver for computing the extensions of the respective semantics. To solve argumentation problems, ASPARTIX is compatible with the ASP solvers Clasp [27] and *DLV* [31].

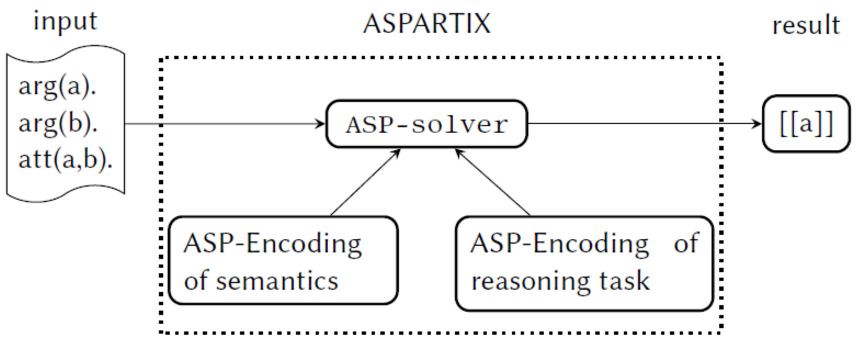


Figure 11: Basic workflow of ASPARTIX [21].

The systems supports various formalisms such as abstract argumentation frameworks (AAFs), bipolar argumentation frameworks (BAFs), preference-based argumentation frameworks (PAFs) and value-based argumentation frameworks (VAFs) [23]. The ASP-encodings, included in ASPARTIX, are continuously improved and the system also regularly participates in the ICCMA [38] Contest.

In ASPARTIX, an AAF is represented through a logic program, which serves as the input for the ASP-solver. The fact $arg(a)$. stands for an argument a and the fact $att(a, b)$. for an attack from a to b. Listing 2 shows the AAF from Figure 2 modeled as a logic program.

Listing 2 AAF modeled as a logic program.

```
arg(a).
arg(b).
arg(c).
arg(d).
att(a,b).
att(b,d).
att(d,c).
att(c,d).
```

The as a logic program modeled WAFs, discussed in Section 6, will also be based on this format with an additional weighting of the attacks. Listing 3 shows an example of an ASP-encoding to compute complete extensions, which was presented in [23].

Listing 3 ASP-encoding for complete extensions.

```
1  % Guess a set S \subseteq A
2  in(X) :- not out(X), arg(X).
3  out(X) :- not in(X), arg(X).
4
5  % S has to be conflict-free
6  :- in(X), in(Y), att(X,Y).
7
8  % The argument x is defeated by the set S
9  defeated(X) :- in(Y), att(Y,X).
10
11 % The argument x is not defended by S
12 not_defended(X) :- att(Y,X), not defeated(Y).
13
14 % admissible
15 :- in(X), not_defended(X).
16
17 % Every argument which is defended by S belongs to S
18 :- out(X), not not_defended(X).
```

Below, we examine the ASP-encoding from Listing 3 in detail. In the encoding, each subset of an input AAF is checked to see if it meets the requirements of completeness. If any of these requirements are not fulfilled, another subset is checked.

- **Lines 2-3:** A random subset S is guessed from the input AAF.

- **Line 6:** The set S is checked for conflict-freeness, with the constraint, that no attacks within S are allowed.
- **Line 9:** All arguments, which are `defeated` by S are identified. This is the case, when an argument is attacked by S.
- **Line 12:** An argument is `not_defended`, if it is attacked by an argument that is `not_defeated`.
- **Line 15:** The set S is examined for admissibility by checking if an argument contained in S is `not_defended`. If that is the case, S is not admissible, which is a basic requirement for complete extensions and thus S would also not be complete.
- **Line 18:** It is checked whether S is a complete extension. S is not a complete extension if an argument outside of S is defended, which in this case is expressed by `not not_defended(X)`.

The encodings for WAFs in Section 6 are based on the foundations of the ASPARTIX encodings. Even though the encodings differ in details, ASPARTIX nonetheless provides a stable basis for implementing various expansions of AAFs.

5.3 ConArg

ConArg [8] is a software tool for modeling and solving problems in the field of abstract argumentation, which was primarily developed by researchers from the University of Perugia in Italy. It is based on constraint programming and therefore uses the open source C++ toolkit *Gecode* [37]. The stand alone version of ConArg can be executed on both Windows and Linux. With the *ConArgLibrary* [6] the core functionalities of ConArg are also available as a C++ library. ConArg supports various semantics, including the semiring-based WAFs from Bistarelli et al. [7] and is therefore particularly interesting to use as a reference system.

For WAFs, the semantics

- α^γ-admissible
- α^γ-complete
- α^γ-grounded
- α^γ-preferred
- α^γ-stable

are supported, under the problem types:

- Enumerate
- Credulous
- Skeptical

The ConArg input format for WAFs is based on the ASPARTIX format, with additional weights on attacks. This adaptation is shown in Listing 4, where the WAF from Figure 3 is modeled for ConArg.

Listing 4 Input WAF for ConArg.

```
arg(a).
arg(b).
arg(c).
arg(d).
arg(e).
att(a,b):-2.
att(b,d):-4.
att(d,c):-3.
att(c,d):-5.
```

The selection of the specific semantics and the parameters for relaxation in ConArg is done through command-line arguments when calling the program. For example the command ***conarg -w weighted -e complete -s d -a 1 -g 2 inputWAF.dl*** tests the argument d of an input WAF for skeptical acceptance with the parameters $\alpha = 1$ and $\gamma = 2$.

6 ASP Encodings for Weighted Argumentation Frameworks

In this section, we introduce ASP-encodings for weighted argumentation frameworks, which are based on the definitions from Section 3 for α-conflict-free sets, α^γ-admissible sets, α^γ-complete$_{\mathbb{C}_3}$ semantics and α^γ-stable semantics. For the development of the encodings, we adopted approaches from ASPARTIX [22]. The α^γ-preferred semantics is not addressed in this work, as the undertaken approaches did not calculate some α^γ-preferred extensions. Here, an approach with labeling semantics would be better to also make use of Clingo heuristics for the maximization of a set. To verify the correctness of the encodings, experiments were conducted for all semantics under various parameters and the results were compared with the respective results of ConArg. The subsequent ASP-encodings for weighted argumentation frameworks build upon each other, meaning that from conflict-freeness to stable semantics, all lines are also contained in the subsequent encodings. For a clearer illustration, only the parts that are specific to each encoding will be shown in the respective listing.

6.1 Modeling Weighted Argumentation Frameworks

In our approach for modeling WAFs as logic programs, we have oriented towards the ASPARTIX format, with the expansion of attacks with an additional numeric weight as the third element. Through this type of modeling, we can directly use the attack fact in the subsequent ASP-encodings for WAFs, without adding more fact for the specific weights.

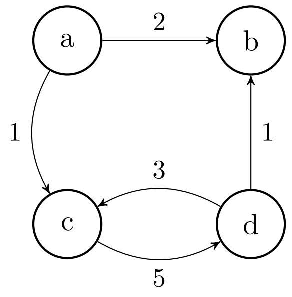


Figure 12: Example for modeling WAFs in ASP.

Listing 5 shows the WAF from Figure 12 modeled as a logic program. The fact `att(a,b,2)` expresses that the argument a attacks the argument b with an attack weight of 2. In addition, Clingo provides the possibility to express, for example through `att(b,d,_)`, that only the attack between the arguments b and d is considered, regardless of its weight.

Listing 5 WAF modeled as a logic program.

```
arg(a).
arg(b).
arg(c).
arg(d).
att(a,b,2).
att(a,c,1).
att(c,d,5).
att(d,b,1).
att(d,c,3).
```

6.2 Conflict-freeness

In Listing 6, we present the ASP-encoding for α-conflict-free sets. In the encoding each subset S of an input WAF is checked to see if the weight of the attacks within S is greater than the input parameter α. If that is the case, the respective S is not α-conflict-free and another subset is checked.

Listing 6 ASP-encoding for α-conflict-free sets.

```
1  % Guess a set S \subseteq A
2  in(X) :- not out(X), arg(X).
3  out(X) :- not in(X), arg(X).
4
5  % Sum of weights for all attacks within the set S.
6  totalInternalConflictWeight(Wconf) :- Wconf = #sum{ W,X,Y : in(X),
       in(Y), att(X, Y, W)}.
7
8  % alpha-conflict-free
9  :- totalInternalConflictWeight(Wconf), Wconf > alpha.
```

In the following, we provide a description of the ASP-encoding by explaining the lines of the code divided according to functionality.

- **Lines 2-3:** A random subset S is guessed from the input WAF.
- **Line 6:** The total internal conflict weight of the attacks within S is calculated. This is done by the `#sum` aggregate function.
- **Line 9:** The set S is checked for α-conflict-freeness. If the sum of weights of the attacks within S is larger than the value of the input parameter `alpha`, then S is not α-conflict-free.

6.3 Admissibility

In Listing 7, we present the ASP-encoding for α^γ-admissible sets. In addition, the full ASP-encoding also contains the ASP-encoding for α-conflict-free sets from Listing 6. To verify a set S for admissibility, it is checked whether S can successfully defend against all attacks from outside S. Therefore, the total weight of an attacking argument is compared with the total attack weight of S against the argument, with additional consideration of the input parameter `gamma`, to relax the notion of w-defence.

Listing 7 ASP-encoding for α^γ-admissible sets.

```
1   % Sum of weights from an argument x outside S against S
2   attWeightToSet(X, WattArg) :- out(X), WattArg = #sum{ W,Y : in(Y),
        att(X, Y, W)}.
3
4   % Sum of weights from S against an argument x
5   attWeightFromSet(X, WattSet) :- out(X), WattSet = #sum{ W,Y : in(Y),
        att(Y, X, W)}.
6
7   % The argument x is defeated by S
8   defeated(X) :- in(Y),out(X), att(Y,X,_), attWeightToSet(X, WattArg),
        attWeightFromSet(X, WattSet), WattArg <= WattSet + gamma.
9
10  % The argument x is not defended by S
11  not_defended(X) :- out(Y), att(Y,X,_), not defeated(Y).
12
13  % alphaGamma-admissible
14  :- in(X), not_defended(X).
```

Below, we examine the ASP-encoding for α^γ-admissible sets in detail. Key concepts are the calculation of the total attack weight from an argument to the set S, the calculation of the attack weight from S against an attacking argument, checking whether an argument is defeated by S and finally checking whether an argument in S is not successfully defended by S against all attacks. If an argument in S is not successfully defended, then S is not α^γ-admissible.

– **Line 2:** The composite weight of all attacks from an argument x outside S against S is calculated by using the `#sum` aggregate function.

– **Line 5:** The composite weight of all attacks from the set S against an argument x outside S is calculated by using the `#sum` aggregate function.

– **Line 8:** An argument x outside S is `defeated`, if the total weight of the attacks from the argument x is less or equal the total weight from S to the argument x plus the input parameter `gamma`.

– **Line 11:** An argument x is `not_defended` by S, if the attacking argument y is `not_defeated` by S.

– **Line 14:** The set S is examined for admissibility by checking if an argument contained in S is `not_defended`. If that is the case, S is not α^γ-admissible.

6.4 Complete Semantics

In Listing 8, we present the ASP-encoding for α^γ-complete semantics. The full encoding also includes the ASP-encoding for α-conflict-free and α^γ-admissible sets from Listings 6 and 7, since both are prerequisites for α^γ-complete extensions. To examine a set S for α^γ-completeness, we check whether an argument outside S is γ-defended by S, with the defence weight of the respective argument also taken into account. If such an argument exists, S is not α^γ-complete.

Listing 8 ASP-encoding for α^γ-complete extensions.

```
1   % Defence weight from one argument outside S to another
2   defWeight(X,Y,W) :- out(X), out(Y), att(X,Y,W).
3   defWeight(X,Y,0) :- out(X), out(Y), not att(X,Y,_).
4
5   % The argument x is not outside defended
6   not_outsideDefended(X) :- out(Y), out(X), att(Y,X,Watt),
        defWeight(X,Y,Wdef), attWeightFromSet(Y,Wset), attWeightToSet(Y,
        WattArg), Watt + WattArg > Wdef + Wset + gamma.
7   not_outsideDefended(X) :- out(Y), out(X), att(Y,X,_),
        attWeightFromSet(Y,Wset), Wset = 0.
8
9   % alphaGamma-complete
10  :- out(X), not not_outsideDefended(X),
        totalInternalConflictWeight(Wconf),attWeightToSet(X,
        WattArg),attWeightFromSet(X, WattSet), (Wconf + WattArg +
        WattSet) <= alpha.
```

In the following, we examine the ASP-encoding for α^γ-complete semantics in detail. The calculation of the `def_weight` is done for arguments outside S. This weight is taken into account when checking if an argument is `not_outsideDefended`. If an argument that attacks another argument outside S is not attacked by S, then this argument is always `not_outsideDefended`. The set S is not α^γ-complete, when there is an argument, which is `not not_outsideDefended`, and by adding this argument, S is still α-conflict-free.

- **Lines 2-3:** The `def_weight` from one argument outside S to another is calculated. If there is no defence, expressed by `att(X,Y,_)`, the defence weight is 0.
- **Lines 6-7:** It is checked whether an argument x is `not_outsideDefended`. For this purpose, it is examined whether the argument x, together with the set S, can successfully defend itself against all attacking arguments, while also taking into account the value from the input parameter `gamma`. This verification is carried out by `Watt + WattArg > Wdef + Wset + gamma`. In Line 7, it is additionally checked if all arguments y, which attack an argument x, are also attacked by S. If there is no attack from S to an argument y, then `Wset=0` is fulfilled and the argument x is thus `not_outsideDefended`.
- **Line 10:** The set S is examined for α^γ-completeness by checking if an argument

contained in S is `not not_outsideDefended`. In addition, by adding the argument x, α-conflict-freeness must still be fulfilled, which is verified by `(Wconf + WattArg + WattSet) <= alpha`.

6.5 Stable Semantics

In Listing 9, we introduce the ASP-encoding for α^γ-stable semantics. The encodings from the previously introduced listings are also included in the full encoding, with the additional condition that all arguments which are not contained in S must be attacked by S.

Listing 9 ASP-encoding for α^γ-stable extensions.

```
% The argument x is attacked by S
attackedByS(X) :- out(X), in(Y), att(Y,X,_).

% alphaGamma-stable
:- out(X), not attackedByS(X).
```

– **Line 2:** An argument outside S is `attackedByS` if the condition `att(Y,X,_)` is met, since the weight is not relevant.

– **Lines 5:** The set S is α^γ-stable, if all Arguments that are outside S are attacked by S.

6.6 Discussion

The provided encodings are intended as a foundation but can still be further optimized, for example, as introduced in [20] through preprocessing techniques for calculating credulously accepted arguments under complete semantics. Furthermore, for the case $\alpha = 0$, the calculation of the internal conflict weight could be simplified, since in this case generally no attacks within a w-conflict-free set are allowed. For an ASP-encoding for preferred semantics, an approach with labeling semantics would be better, to also make use of Clingo heuristics for the maximization of a set. For an implementation of α^γ-grounded semantics, a definition for w-grounded semantics is presented in [9], in which the w-grounded semantics is defined as the maximal (with respect to set inclusion) w-admissible extension included in the intersection of w-complete extensions. However, if relaxations are to be allowed, this must also be considered in the implementation.

7 Evaluation of the ASP Implementation

In this section, we examine the ASP-implementation for weighted argumentation frameworks in detail. For this purpose, we have generated test instances of weighted argumentation frameworks using AF-Benchgen2 [15] with an additional weighting of the attacks, and tested these under various semantics and problem types (EE, EC, ES, DC, DS). Additionally, we examined instances from ICCMA 2019 [4], which we also converted into weighted argumentation frameworks, by assigning weights to the attacks.

7.1 Generation of Test Cases

To generate test instances, the AAF Generator AF-Benchgen2 [15] was utilized, for the creation of a total of 300 AAFs. The models supported by AFBenchgen2 for random graphs are Barabási-Albert [1], Watts–Strogatz [40], and Erdős–Rényi [24]. For each model, 100 AAFs were generated, which we weighted by assigning a random integer value between 1 and 10 to the attacks, based on a uniform distribution. Additionally, for the problem types DC and DS, randomized test arguments were generated, selected within the range corresponding to the number of arguments for the respective instance.

Below are the parameters listed with which the test instances were generated. The descriptions of the parameters were taken from the description integrated in AF-BenchGen2. For the model Erdős–Rényi, we chose a high value for ER_probAttacks to analyze how the ASP-encodings behave with a high number of attacks.

– BA_WS_probCycles = 0.5. Probability that an argument is part of a cycle (used with BarabasiAlbert and WattsStrogatz only).

– ER_probAttacks = 0.5. Probability of having an attack between two arguments (used with ErdosRenyi only).

– WS_baseDegree = 2. Base degree for each node (used with WattsStrogatz only).

– WS_beta = 0.5. Probability to 'rewire' an edge (used with WattsStrogatz only).

In addition to the generated test instances, we also utilized the AAFs from ICCMA 2019 [4], with an additional weighting of the attacks through a random integer value between 1 and 10, based on a uniform distribution. To be able to perform experiments for WAFs under time restrictions, we limited the maximum number of arguments to 300, thus examining 186 of the 326 instances.

7.2 Experimental Design

In the experiments, we explore the generated instances of WAFs under various semantics, relaxation parameters, and problem types. Initially, we analyze the instances in terms of acceptability, and then we compare the runtime performance of the ASP-implementation with the runtime performance of ConArg. The types of problems we consider are:

Definition 19 *Given a WAF $F = \langle \mathcal{A}, R, W, \mathbb{S} \rangle$ and a semantics $\sigma \in \{\alpha^\gamma$-adm, α^γ-com, α^γ-prf, α^γ-stb$\}$, the following problem types are defined:*

- ***EE-σ**: Enumerate all σ-extensions of F.*
- ***DC-σ**: For a given argument $a \in \mathcal{A}$, decide whether a is in at least one σ-extension of F.*
- ***DS-σ**: For a given argument $a \in \mathcal{A}$, decide whether a is in all σ-extensions of F.*
- ***EC-σ**: Enumerate all $a \in \mathcal{A}$, which are in at least one σ-extension of F.*
- ***ES-σ**: Enumerate all $a \in \mathcal{A}$, which are in all σ-extensions of F.*

The configuration for Clingo must be set accordingly to solve the specific problem type. To evaluate an argument a for DC-σ, the constraint $: -not\ in(a)$ is added, and if the program is satisfiable, the argument is credulously accepted. For DS-σ, :- in(a) is added, and if the program is unsatisfiable, it means an answer set was found that does not include the argument, therefore, the argument is not skeptically accepted. For the problem types EC and ES, the Clingo heuristics brave and cautious, are applied respectively. The commands for each problem type are listed below.

- EE-σ: ***clingo inputWAF.apx semantics.dl filter.lp 0***.
- DC-σ: ***clingo inputWAF.apx semantics.dl filter.lp 1***.
- DS-σ: ***clingo inputWAF.apx semantics.dl filter.lp 1***.
- EC-σ: ***clingo inputWAF.apx semantics.dl filter.lp -e brave***.
- ES-σ: ***clingo inputWAF.apx semantics.dl filter.lp -e cautious***.

In the experiments we investigate the research questions defined in Section 1. The experiments are divided into the sections *Admissibility*, *Complete Semantics*, and *Stable Semantics*. To demonstrate the properties of the ASP-implementation, the generated instances of the models Barabási-Albert, Watts–Strogatz, and Erdős–Rényi were either examined individually or suitable combinations were chosen. For the experiments with the ASP-implementation, the CPU Time was taken directly from the Clingo output. For the experiments with ConArg the runtime was recorded using Python, since ConArg does not output a runtime. For conducting the experiments with the instances from ICCMA 2019, a timeout of 600s was chosen. The experiments were carried out on a server with the operating system Ubuntu 20, with an octacore Intel Xeon E5-2643V3 processor and 48GB RAM.

7.3 Results

In this section, we present our experimental results. For the generated instances with the models Barabási-Albert, Watts-Strogatz, and Erdős–Rényi, we apply the short forms B, W and E, respectively. In Section 7.3.1, we investigate the impact of the model by individually examining the instances B, W and E for the problems EE and EC under various semantics. In Section 7.3.2, we examine the impact of the problem type on the ASP-implementation with a combination of the instances B and W under various semantics. In Section 7.3.3 we investigate the impact of the relaxations. Therefore, we apply various relaxations and examine their impact on acceptability under various semantics, and also investigate the effect of the relaxations on the runtime performance of the ASP-implementation. In Section 7.3.4, we examine the impact of the implementation by comparing the runtime performance of the ASP-implementation with the runtime performance of ConArg with the generated instances and with instances from ICCMA 2019.

7.3.1 Impact of the Model

To investigate the impact of the model in terms of acceptability under various semantics, we have individually analyzed the instances B, W and E in these experiments. For the investigation of admissibility, we selected the problem EC, because for the problem EE the output would be excessively large. For the α^γ-complete and α^γ-stable semantics, we chose the problem EE, as here the number of extensions is within feasible limits.

Admissibility

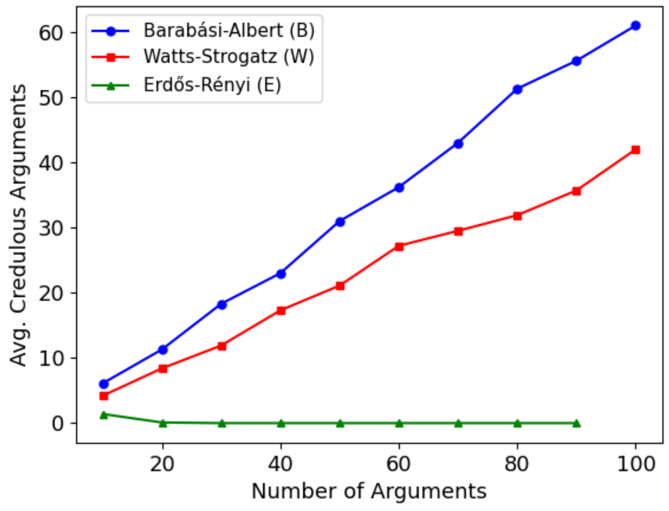
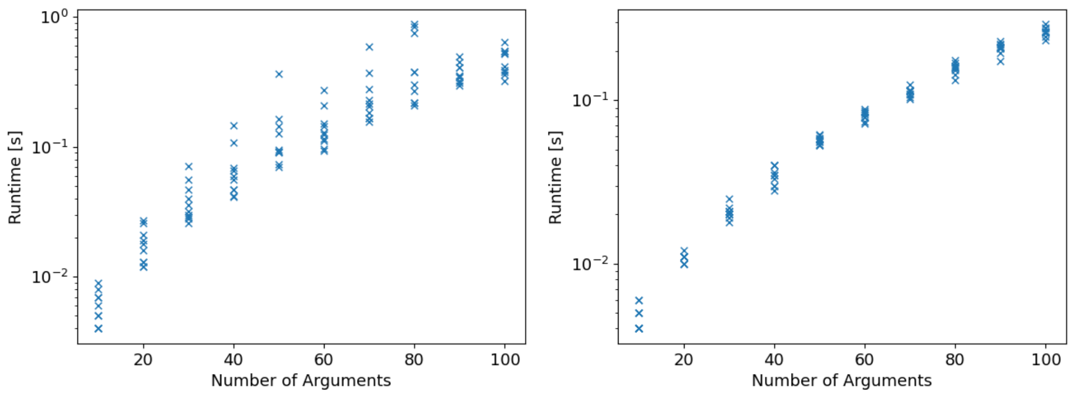


Figure 13: Average credulously accepted arguments of the problem EC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for different models.

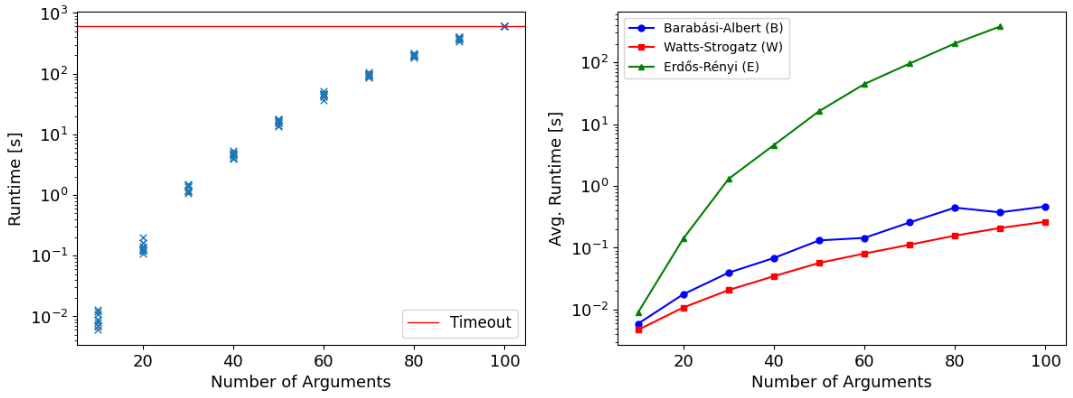
In Figure 13 we see the average number of credulously accepted arguments under α^{γ}-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$. The instances B have the highest average number of credulously accepted arguments, followed by the instances W. For the instances E, due to the high number of attacks, credulously accepted arguments were found only for instances with 10 arguments. In Section 7.3.3, we will revisit this issue and examine the same instances with relaxations.

The runtime for solving the problem EC is shown in Figure 14. We see that the calculation of credulously accepted arguments for the instances W was the fastest, and also the runtime for the instances B was only slightly higher. The by far highest runtime is for the instances E, which can be attributed to the high number of attacks between the arguments, and the fact that no credulously accepted arguments were found for instances with more than 10 arguments.



(a) Instances Barabási-Albert (B).

(b) Instances Watts-Strogatz (W).



(c) Instances Erdős–Rényi (E).

(d) Average runtime.

Figure 14: Runtime comparison of the problem EC under α^{γ}-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for different models.

Complete Semantics

As illustrated in Figure 15, we observe that, for the average number of 0^0-complete extensions, the instances B and W behave similarly. Furthermore, the average number of 0^0-complete extensions increases exponentially with the number of arguments. For the instances E, only the empty set is output for instances with 20 arguments or more, which is not surprising since also no credulously accepted arguments were found under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$. For the instances E, only instances up to 70 arguments were successfully computed, as for a larger number of arguments the runtime limit of 600 seconds was reached.

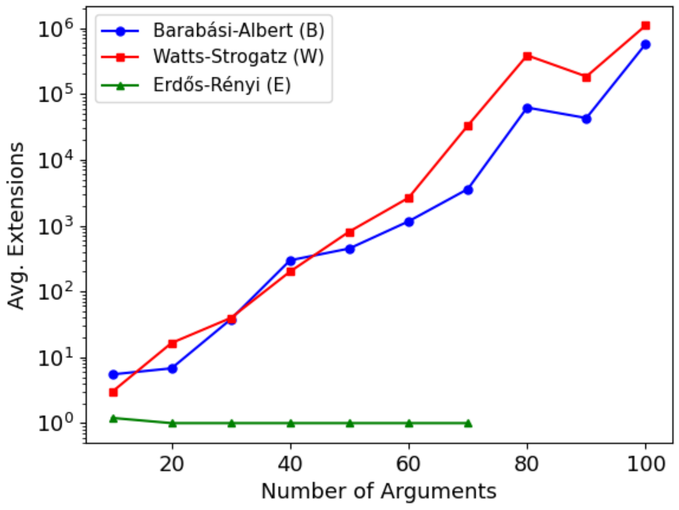
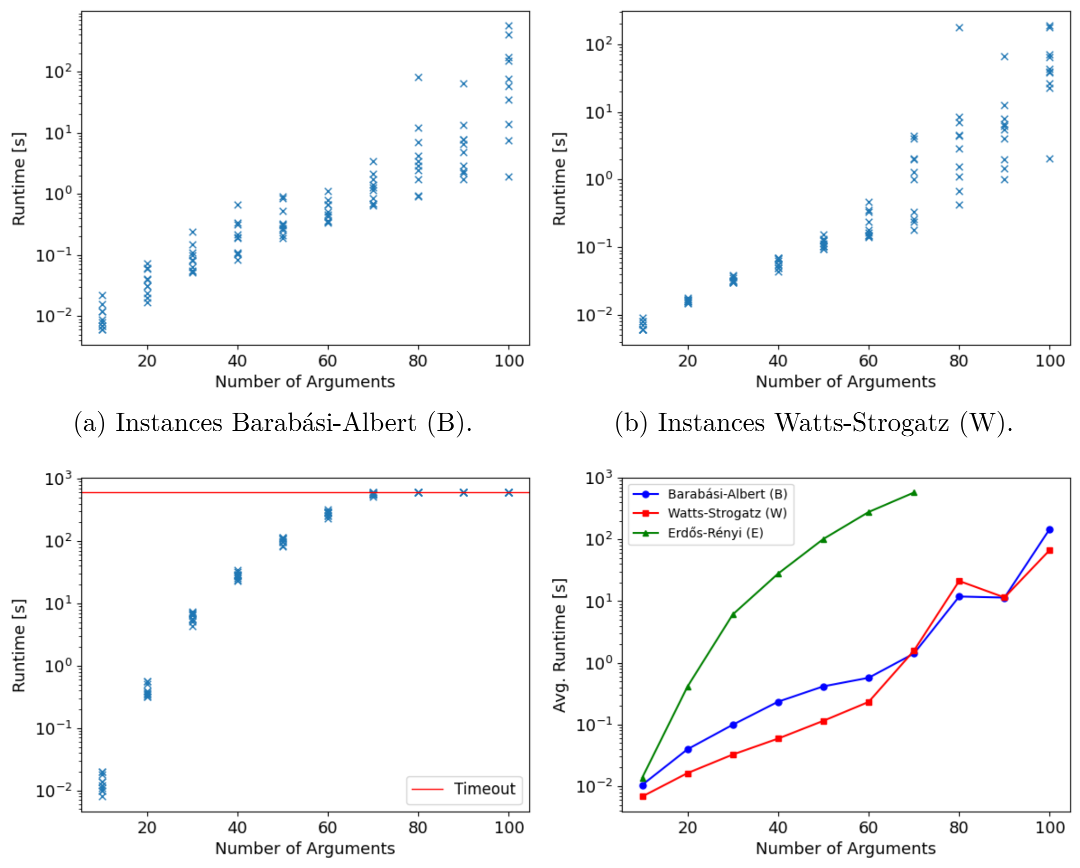


Figure 15: Average 0^0-complete extension of the problem EE for different models.

In Figure 16, the runtime for calculating the 0^0-complete extensions of the problem EE for the different instances is shown. We observe that for the instances B and W, all 0^0-complete extensions are calculated within the runtime limit, but with the runtime increasing exponentially due to the high number of 0^0-complete extensions. The by far highest runtime, is for the instances E, where the runtime limit of 600 seconds is reached for instances with 70 arguments or more, which we already observed for the problem EC under α^γ-admissible semantics. To calculate the average runtime, the experiment for 70 arguments was repeated with increased runtime limits.



(a) Instances Barabási-Albert (B).

(b) Instances Watts-Strogatz (W).

(c) Instances Erdős–Rényi (E).

(d) Average runtime.

Figure 16: Runtime comparison of the problem EE under α^{γ}-complete semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for different models.

Stable Semantics

In Figure 17, we see the average number of 0^0-stable extensions of the problem EE for the instances B, W and E. We observe that for all instances, the number of 0^0-stable extensions is low, and no extensions are found beyond 40 arguments. Therefore, in Section 7.3.3, we will apply relaxation to obtain a larger number of solutions.

The runtime for computing 0^0-stable extensions is shown in Figure 18. Since almost no 0^0-stable extensions were found, the runtime for the instances B and W is significantly reduced compared to the runtime for the calculation of 0^0-complete extensions. For the instances E, we see similarly high runtimes as observed for the calculation of the 0^0-complete extension, reaching the timeout of 600s from 70 arguments onwards. To calculate the average runtime for the instances E with 70 arguments, we repeated the experiment for these instances without time limits.

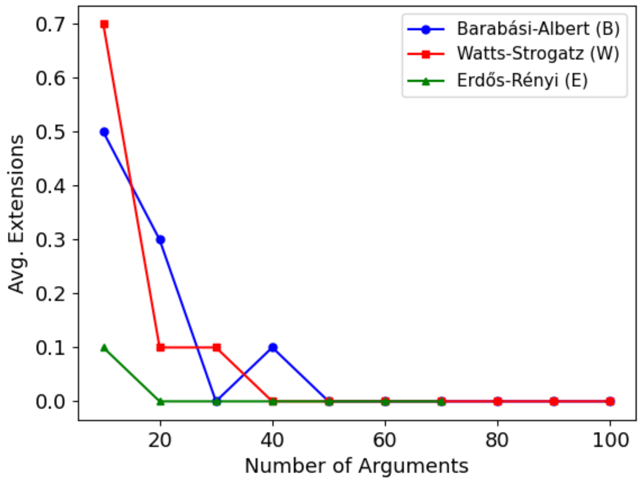
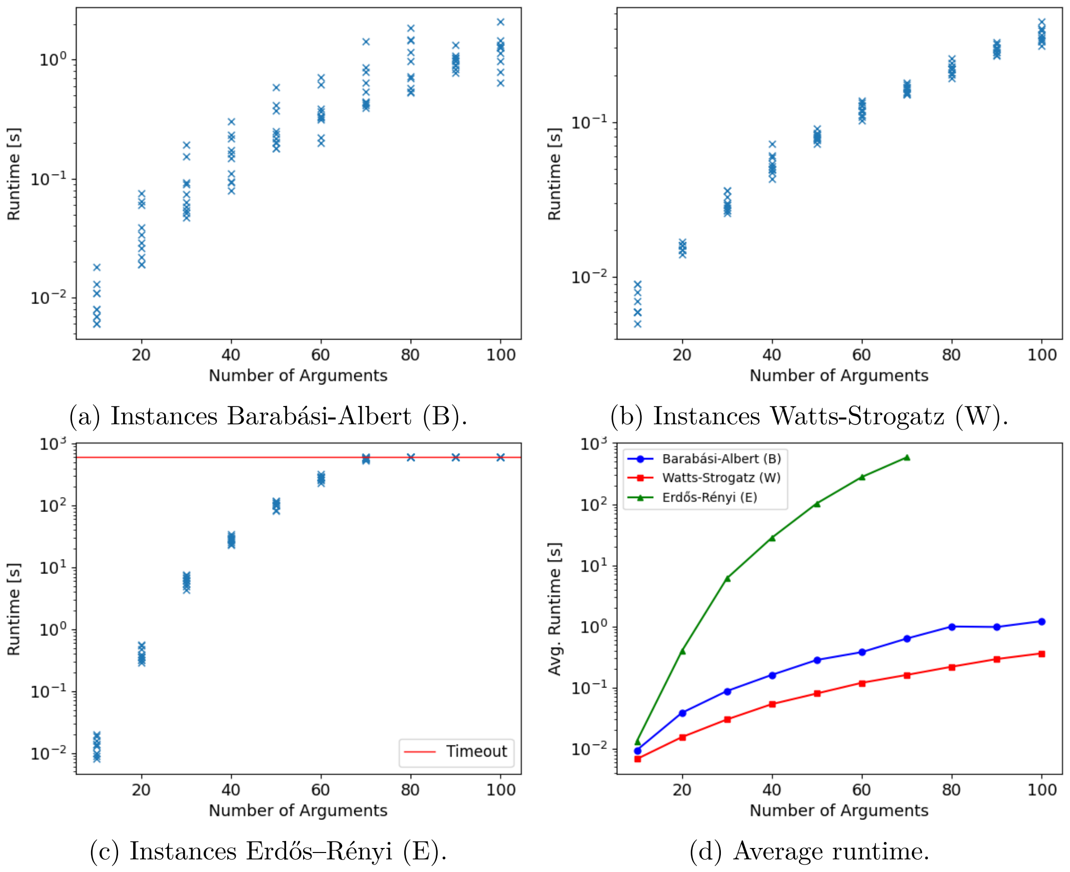


Figure 17: Average 0^0-stable extension of the problem EE for different models.



(a) Instances Barabási-Albert (B).

(b) Instances Watts-Strogatz (W).

(c) Instances Erdős–Rényi (E).

(d) Average runtime.

Figure 18: Runtime comparison of the problem EE under α^γ-stable semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for different models.

7.3.2 Impact of the Problem Type

In this section, we investigate the impact of the problem type on the combined instances of B and W, as these exhibited similar behavior in the previous experiments. Therefore, a total of 200 instances were examined. The instances E were not analyzed in this section, as only few extensions can be found without relaxation, resulting in similar runtime behavior for all problem types.

Admissibility

In Figure 19, the average runtime for the problems EC and DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$ is illustrated. We observe that both problem types exhibit similar runtime behavior, which is quite interesting because with the problem EC, all credulously accepted arguments are calculated and with the problem DC, only a specific argument is checked for acceptability. The problems EE, ES, DS were not considered since for EE the output is too large, and for the problems ES and DS, no skeptically accepted arguments can be found, as they would have to be contained in the empty set.

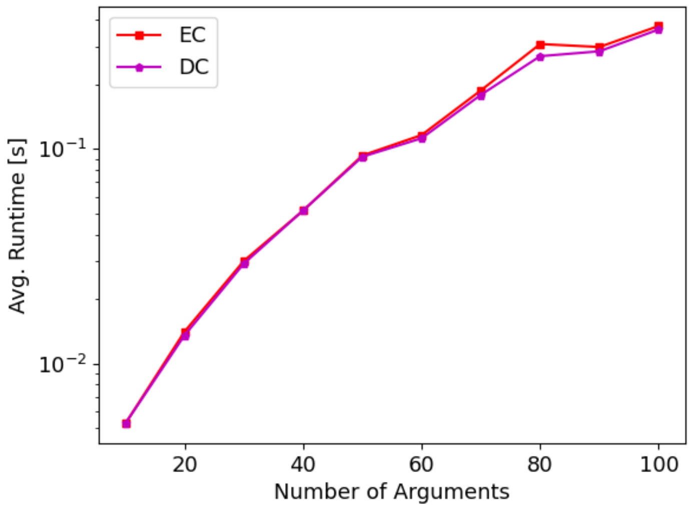


Figure 19: Runtime comparison between the problems EC and DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for the combined instances B and W.

Complete Semantics

In Figure 20, we see the average runtime of various problem types under α^γ-complete semantics with the parameters $\alpha = 0$ and $\gamma = 0$. The problems EC, ES, DC and DS show similar runtime behavior, as has already been observed for the problems EC and DC under α^γ-admissible semantics. To accelerate the computation time of the problems DC and DS, it would be advantageous to employ preprocessing techniques to avoid evaluating all arguments, and thus achieving a faster runtime. The problem EE has a higher runtime in comparison to other problems, which can be attributed the high number of 0^0-complete extensions, as illustrated in Figure 15.

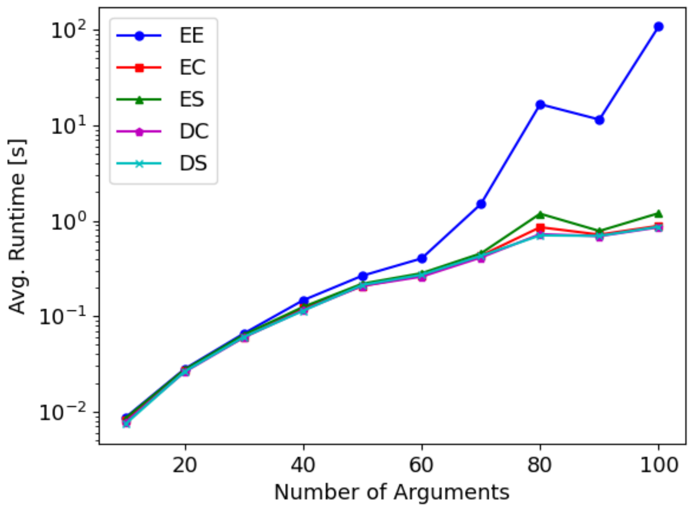


Figure 20: Runtime comparison between various problem types under α^γ-complete semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for the combined instances B and W.

Stable Semantics

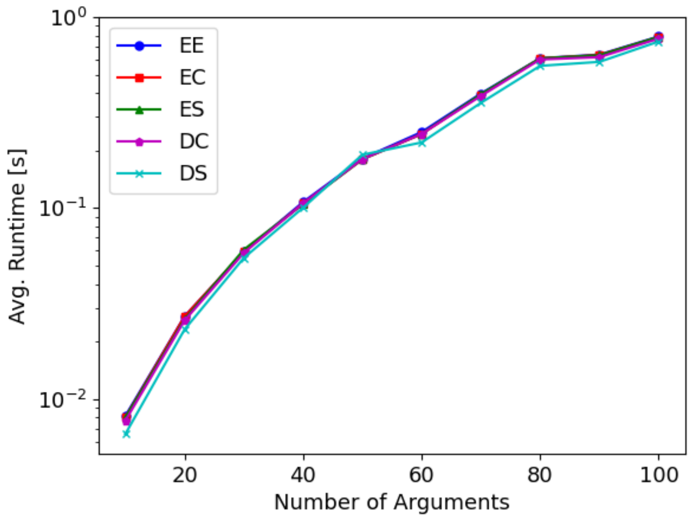


Figure 21: Runtime comparison between various problem types under α^γ-stable semantics with the parameters $\alpha = 0$ and $\gamma = 0$ for the combined instances B and W.

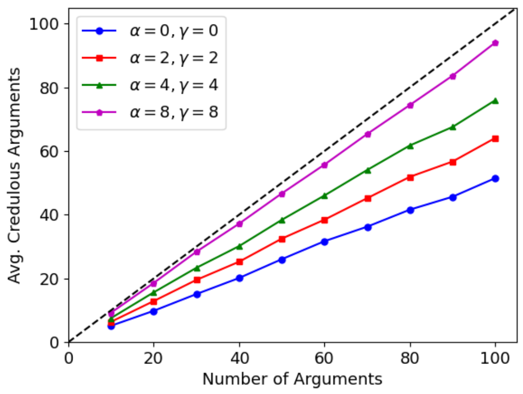
In Figure 21, the runtime of various problem types under α^γ-stable semantics with the parameters $\alpha = 0$ and $\gamma = 0$ is illustrated. We observe a similar runtime across all problem types. The problem EE shows no differences in runtime, since for the parameters $\alpha = 0$ and $\gamma = 0$, as can be seen in Figure 17, the number of 0^0-stable extensions is low, and thus the number of extensions has no impact on the runtime.

7.3.3 Impact of the Relaxations

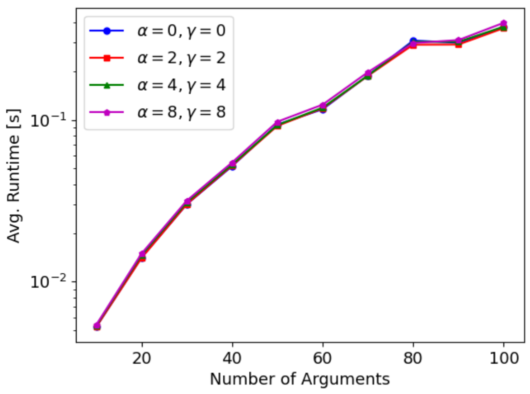
In this section, we investigate the impact of the relaxations under various semantics. We analyzed the combined instances of B and W with the problem EC, and the instances E with the problem EE. For the parameters α and γ, we have chosen suitable values to illustrate the effects of the relaxations. Due to the high runtime of the problem EE with the instances E, we only analyzed instances up to 60 arguments in the associated experiments. The effects of the individual parameters were not analyzed separately, as this would require an extended investigation, which is out of the scope of this work.

Admissibility

In Figure 22, we see a comparison of the problem EC with various relaxation parameters under α^γ-admissible semantics, with the combined instances B and W. We see that the number of credulously accepted arguments increases continuously with the increase of the relaxation parameters. For the parameters $\alpha = 8$ and $\gamma = 8$, almost all arguments are credulously accepted. In terms of runtime, the relaxations do not have significant effects, and a consistently fast runtime is achieved for all instances.



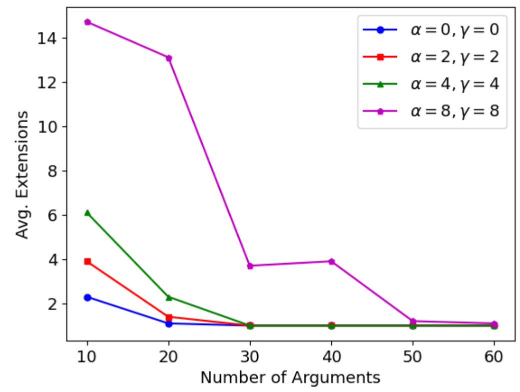
(a) Average credulously accepted arguments.



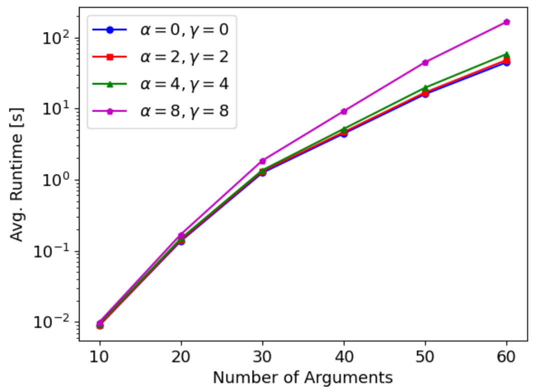
(b) Average runtime.

Figure 22: Relaxation comparison of the problem EC under α^γ-admissible semantics, with the combined instances B and W.

In Figure 23, a comparison of the problem EE with various relaxation parameters for the instances E is illustrated. We see that as the parameters increase, the number of average α^γ-admissible extensions also steadily increases. However, beyond a certain number of arguments, even with relaxations, the empty set is identified as the only α^γ-admissible extension. The relaxations result in a longer runtime, which is evident for the relaxation parameters $\alpha = 8$ and $\gamma = 8$.



(a) Average extensions.

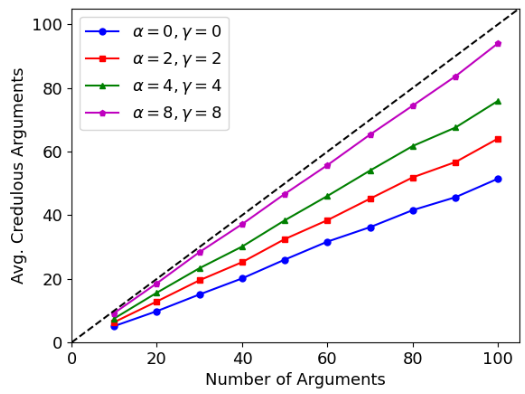


(b) Average runtime.

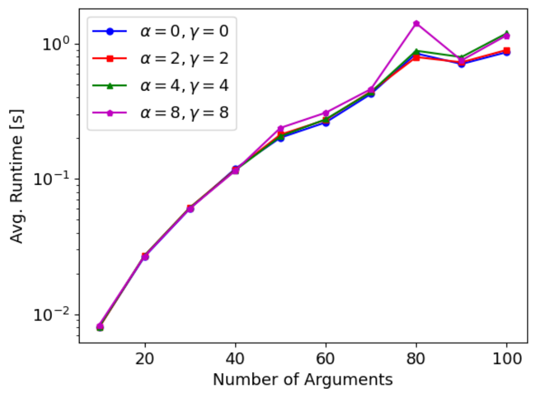
Figure 23: Relaxation comparison of the problem EE under α^γ-admissible semantics, with the instances E.

Complete Semantics

In Figure 24, we see a comparison of the problem EC with various relaxation parameters under α^γ-complete semantics with the combined instances of B and W.



(a) Average credulously accepted arguments.

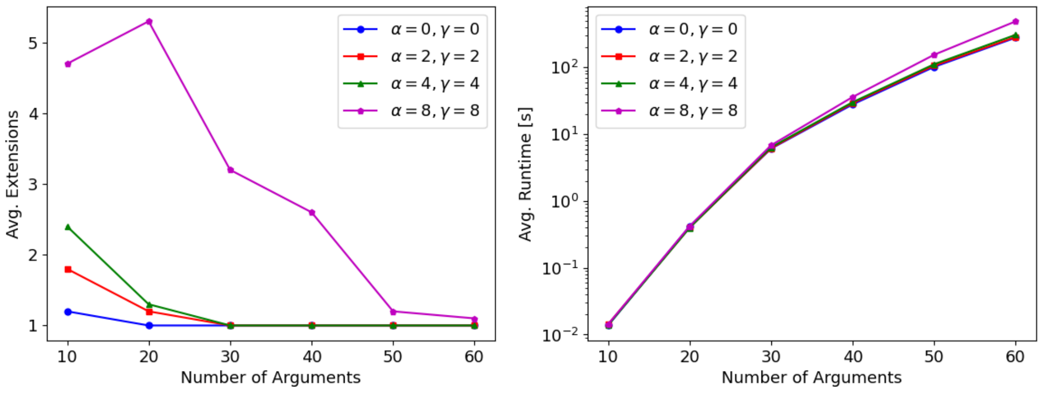


(b) Average runtime.

Figure 24: Relaxation comparison of the problem EC under α^γ-complete semantics, with the combined instances B and W.

The average number of credulously accepted arguments matches with the average number of credulously accepted arguments for α^γ-admissible semantics, as shown in Figure 22, since for both semantics under the problem EC the same constraints are present. However, we see that the runtime for α^γ-complete semantics is higher. Therefore, when investigating the problems DC and EC, the α^γ-admissible semantics should be chosen instead of the α^γ-complete semantics.

In Figure 25, a comparison with various relaxation parameters of the problem EE under α^γ-complete semantics with instances E is illustrated. Due to the additional constraints of α^γ-complete semantics, we observe a lower average number of extensions than for α^γ-admissible semantics. The runtime is consistently higher compared to α^γ-admissible semantics, and we can again observe an increased runtime for the parameters $\alpha = 8$ and $\gamma = 8$.

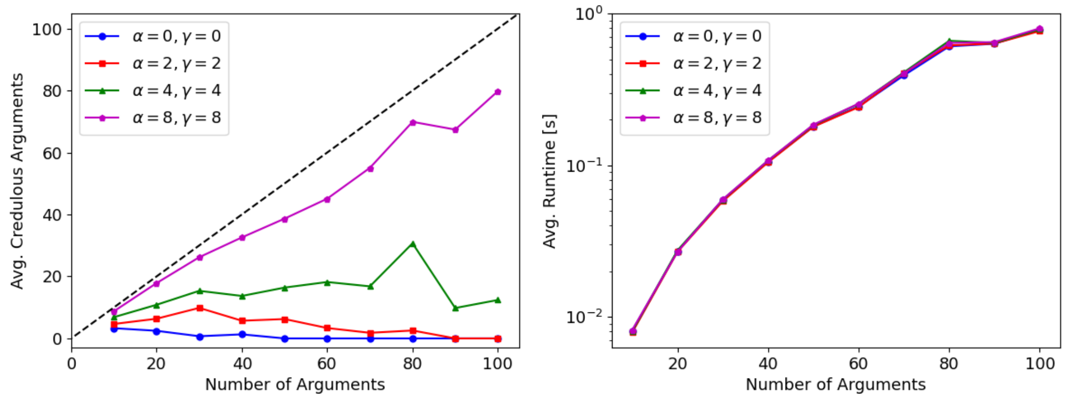


(a) Average extensions. (b) Average runtime.

Figure 25: Relaxation comparison of the problem EE under α^γ-complete semantics, with the instances E.

Stable Semantics

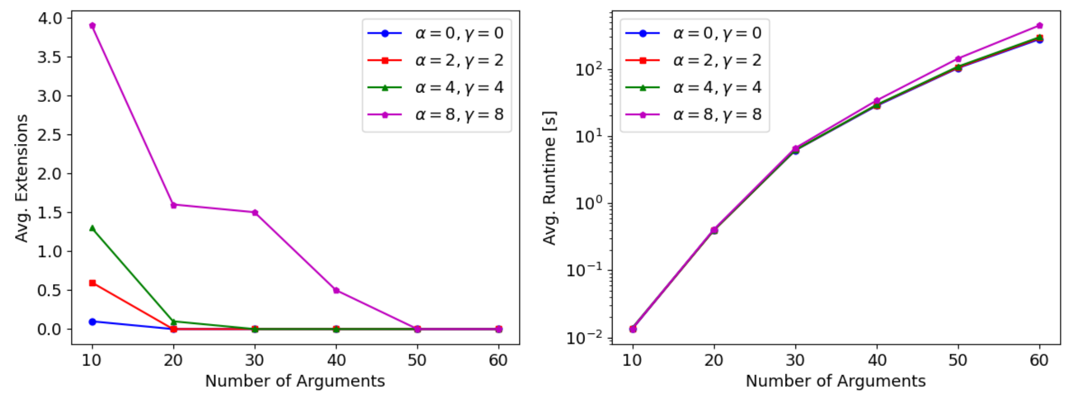
In Figure 26, we see a comparison with various relaxation parameters for the problem EC under α^γ-stable semantics with a combination of the instances B and W. We observe that the average number of credulously accepted arguments is significantly lower compared to α^γ-admissible semantics. However, if the parameters α and γ are chosen to be large enough, in this case $\alpha = 8$ and $\gamma = 8$, we see that the number of credulously accepted arguments is close to those of the α^γ-admissible semantics with the same relaxations. An effect of the relaxation parameters on the runtime for the problem EC under α^γ-stable semantics is not noticeable for these instances.



(a) Average credulously accepted arguments. (b) Average runtime.

Figure 26: Relaxation comparison of the problem EC under α^γ-stable semantics, with the combined instances B and W.

The effect of the relaxation parameters under α^γ-stable semantics for the problem EE with the instances E is illustrated in Figure 27. Without relaxations, the number of α^γ-extensions is almost zero. With the relaxation parameters $\alpha = 8$ and $\gamma = 8$, 8^8-stable extensions can be determined for instances with up to 40 arguments. Since α^γ-stable semantics involve additional constraints compared to α^γ-complete semantics, the number of extensions is accordingly lower. Regarding runtime, only for the parameters $\alpha = 8$ and $\gamma = 8$ a slight increase can be observed.



(a) Average extensions. (b) Average runtime.

Figure 27: Relaxation comparison of the problem EE under α^γ-stable semantics, with the instances E.

7.3.4 Impact of the Implementation

In this section, we investigate the impact of implementation by comparing the runtime performance of the ASP-implementation with the runtime performance of ConArg. We examine the instances B, W and E under various problems and semantics with suitable values for relaxations. Additionaly, we examine instances from ICCMA 2019, under α^{γ}-admissible semantics for the problem DC.

Admissibility

In Figure 28, the runtime comparison between the ASP-implementation and ConArg for the problem DC under α^{γ}-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$ is illustrated. We observe that for the combined instances of B and W, both implementations show overall fast runtime performance. However, ConArg was able to demonstrate better runtime performance except for a single instance with 90 arguments. Due to the large output for the instances B and W, the problem EE was not investigated under α^{γ}-admissible semantics for these instances, but we will consider this problem later in this section under α^{γ}-complete and α^{γ}-stable semantics.

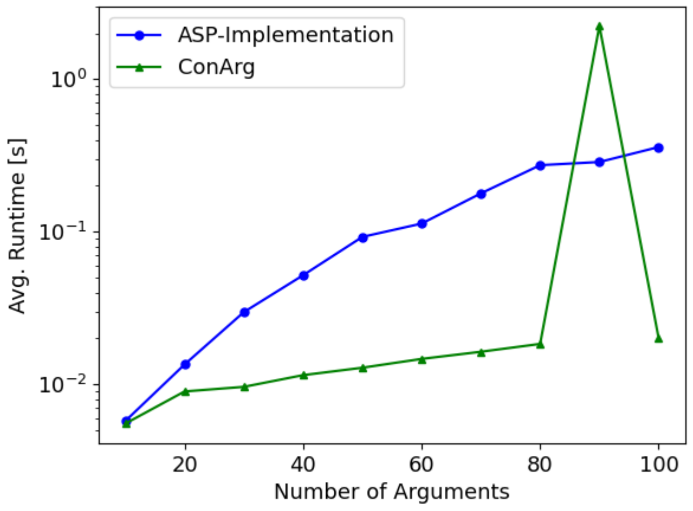
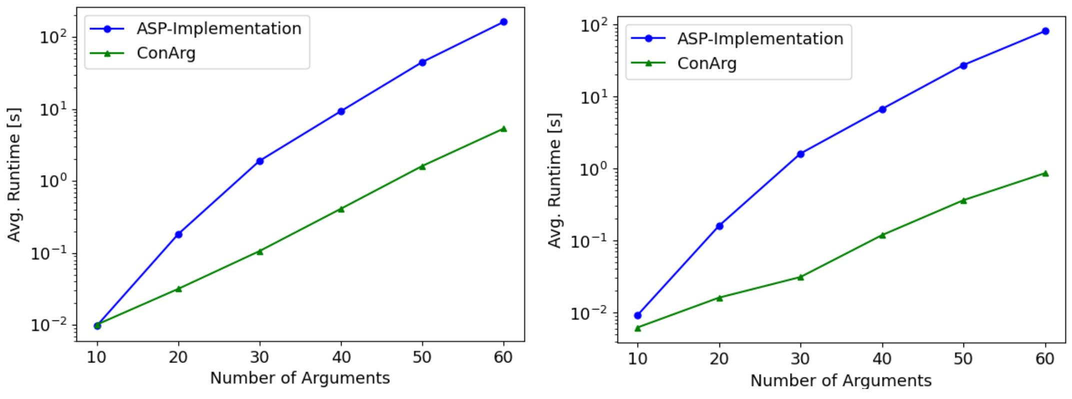


Figure 28: Runtime comparison between the ASP-implementation and ConArg for the problem DC under α^{γ}-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$, with the combined instances B and W.

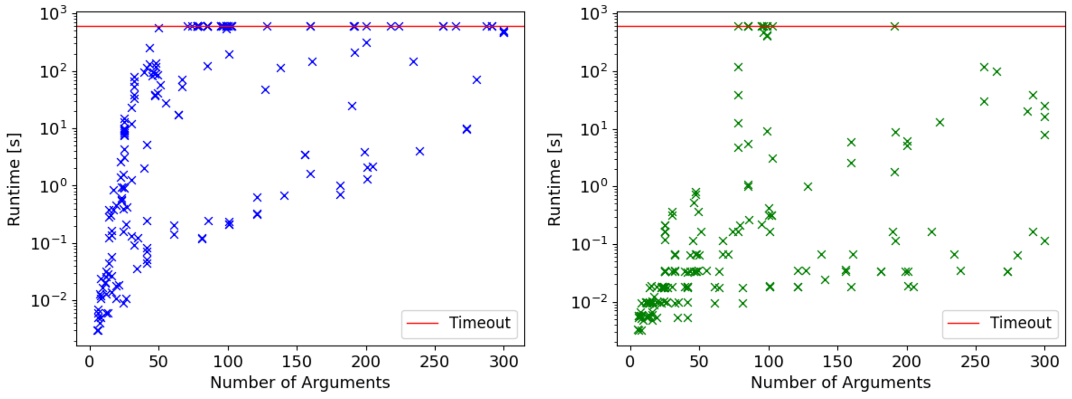
For the investigation of the instances E under α^{γ}-admissible semantics, we chose suitable relaxation parameters. In Section 7.3.3, we found that with the parameters $\alpha = 8$ and $\alpha = 8$, we obtain a suitable number of extensions. In Figure 29, we see that for both problems EE and DC, ConArg shows a consistently better runtime performance than the ASP-implementation. The extended runtimes, which we observed for the instances E are thus only encountered with the ASP-implementation.



(a) Average runtime EE. (b) Average runtime DC.

Figure 29: Runtime comparison between the ASP-implementation and ConArg for the problems EE and DC under α^γ-admissible semantics with the parameters $\alpha = 8$ and $\alpha = 8$, with the instances E.

To further investigate the runtime performances of the ASP-implementation and ConArg under α^γ-admissible semantics for the problem DC, we examined instances from ICCMA 2019, with additional weights of the attacks ranging from 1 to 10. For the execution of the experiment, we chose a timeout limit of 600 seconds and considered instances with up to 300 arguments.



(a) ASP-implementation (b) ConArg

Figure 30: Runtime of the ASP-implementation and ConArg with the ICCMA instances for the problem DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$.

In Figure 30, we see the runtimes of the ASP-implementation and ConArg for calculating the problem DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$. With the ASP-implementation, 142 out of the 186 instances were successfully calculated. For 37 instances, the timeout limit was reached and for 7 instances Clingo terminated with an error. With ConArg, the timeout was reached 11 times, thus 175 out of the 186 instances were successfully calculated. We observe that the ASP-implementation already exhibits high runtimes for instances with less than 50 arguments compared to ConArg, which shows runtimes of under 1 second for these instances.

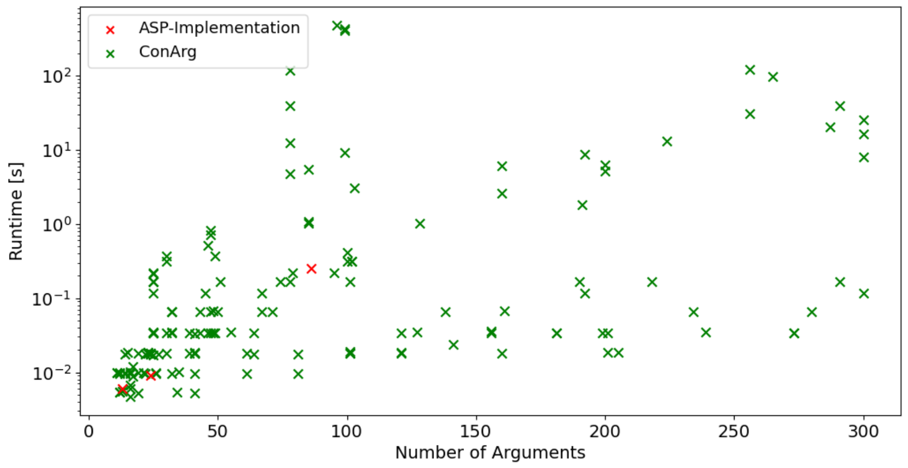


Figure 31: Runtime comparison of the ASP-implementation and ConArg with the ICCMA instances for the problem DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$.

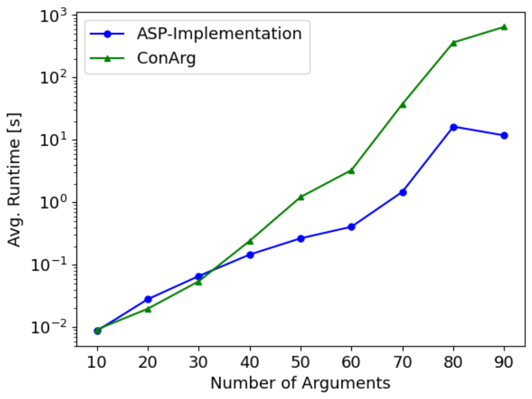
In Figure 31, we see the runtime comparison between the ASP-implementation and ConArg for the problem DC under α^γ-admissible semantics with the parameters $\alpha = 0$ and $\gamma = 0$. The faster runtime is always shown, starting from a count of 10 arguments to exclude effects of runtime measurement. We observe that ConArg consistently exhibits faster runtime. Only the computation of 3 instances could be performed faster with the ASP-implementation under α^γ-admissible semantics.

After evaluating the instances B, W, and E, as well as the instances from ICCMA, we can conclude that ConArg, for the problem DC under α^γ-admissible semantics, consistently shows better runtime performance than the ASP-implementation, except for a few individual instances.

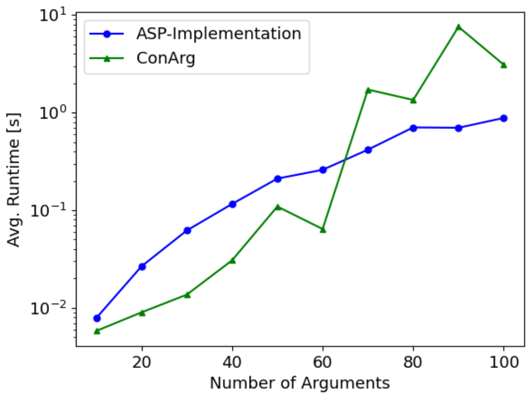
Complete Semantics

To compare the runtime performance of the ASP-implementation with the runtime performance of ConArg under α^γ-complete semantics, we considered the problems EE and DC for the combined instances B and W and for the instances E. For the respective instances and problem types, we chose suitable relaxation parameters.

In Figure 32, we see the runtime comparison between the ASP-implementation and ConArg for the problems EE and DC under α^γ-complete semantics with the param-



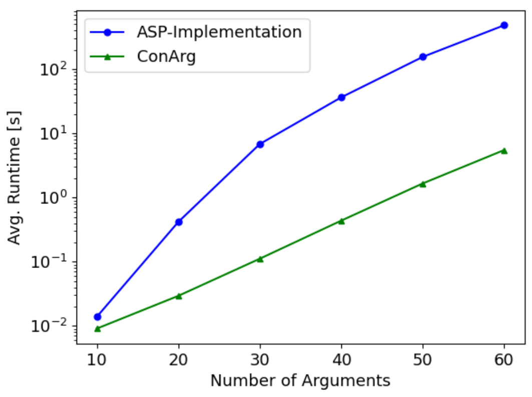
(a) Average runtime EE.



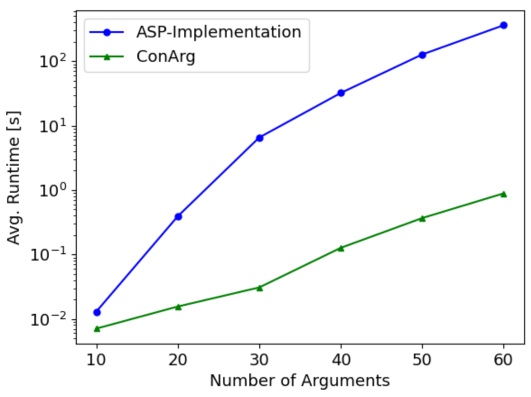
(b) Average runtime DC.

Figure 32: Runtime comparison between the ASP-implementation and ConArg for the problems EE and DC under α^{γ}-complete semantics with the parameters $\alpha = 0$ and $\gamma = 0$, with the combined instances B and W.

eters $\alpha = 0$ and $\gamma = 0$ with the combined instances of B and W. For the problem EE, the ASP-implementation shows a consistently fast runtime and demonstrates better runtime performance than ConArg starting from 40 arguments. For the execution of the experiment with ConArg, runtime limits for calculations with 80 and 90 arguments were increased to be able to calculate the average runtime. For the instances with 100 arguments, due to the high runtime of ConArg, we did not repeat this experiment. As shown in Figure 32, for the problem DC under α^{γ}-complete semantics, ConArg shows better runtime performance up to 60 arguments, after which the ASP-implementation performs better.



(a) Average runtime EE.



(b) Average runtime DC.

Figure 33: Runtime comparison between the ASP-implementation and ConArg for the problems EE and DC under α^{γ}-complete semantics with the parameters $\alpha = 8$ and $\alpha = 8$, with the instances E.

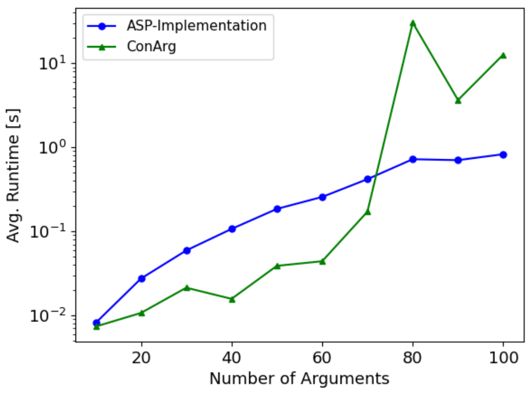
For the instances E, ConArg significantly outperforms the ASP-implementation in terms of runtime for the problems DC and EE as shown in Figure 33. This difference in runtime was also previously observed under α^γ-admissible semantics.

We have calculated the problem DC with the ASP-implementation in this experiment under α^γ-complete semantics. As already discussed in Section 7.3.1, this leads to an increased runtime, and therefore, for this problem, the α^γ-admissible semantics should be chosen, since the output is identical. As we can see by comparing the Figures 28 and 32, the runtime for the problem DC with ConArg is different for both experiments, indicating that this optimization has not yet been implemented in ConArg.

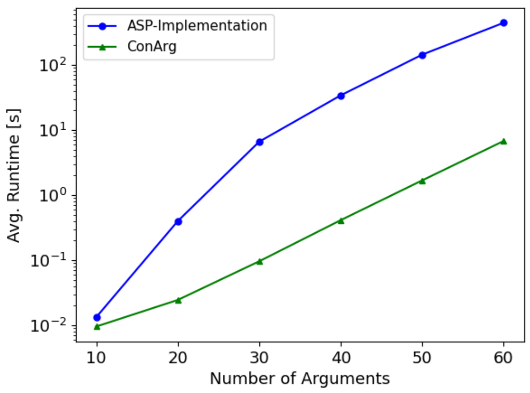
Stable Semantics

For the runtime comparison under α^γ-stable semantics, we only consider the problem EE, since ConArg encounters an error when processing the problems DC and DS under these semantics. For the experiments, we have chosen suitable relaxation parameters so that solutions can be found for α^γ-stable extensions.

In Figure 34, the runtime comparison between the ASP-implementation and ConArg for the problem EE with various instances and relaxations parameters is illustrated. The calculation of the α^γ-stable extensions displays a similar runtime behavior as previously observed in experiments with α^γ-complete semantics. For the combined instances B and W with the relaxation parameters $\alpha = 4$ and $\gamma = 4$, ConArg is faster for a smaller number of arguments, but from 80 arguments onwards, the ASP-implementation shows better runtime performance. For the instances E, ConArg consistently shows better runtime performance than the ASP-implementation, as we have also observed under α^γ-complete semantics.



(a) Instances B and W, $\alpha = 4$, $\gamma = 4$.



(b) Instances E, $\alpha = 8$, $\gamma = 8$.

Figure 34: Runtime comparison between the ASP-implementation and ConArg for the problem EE under α^γ-stable semantics, with various instances and relaxation parameters.

7.4 Discussion

In this section, we further discuss the results of the experiments evaluated in Section 7.3 to address the research questions described in Section 1.

Impact of the Model

We observed that the model used for generating WAFs has a significant impact on the runtime performance of the ASP-implementation. While the generated instances with the models Barabási-Albert and Watts–Strogatz exhibit similar runtime behavior, we observed a significantly higher runtime to solve the instances generated with the model Erdős–Rényi, with a high number of attacks. In terms of acceptability, the generated instances with the models Barabási-Albert and Watts–Strogatz behaved similarly and showed a high number of accepted arguments and extensions under α^γ-admissible and α^γ-complete semantics, whereas for α^γ-stable semantics, hardly any extensions were found without the use of relaxations. For the generated instances with the model Erdős–Rényi, only credulously accepted arguments for instances with a low number of arguments were found under all semantics.

Impact of the Problem Type

For the impact of the problem type, we observed that the ASP-implementation generally exhibited consistent runtime behavior across the different problem types. Only for the problem EE, we observed an increase in runtime under α^γ-complete semantics, due to the large number of computed extensions. It is particularly interesting that the problems EC and ES demonstrated similar runtime behavior compared to the problems DC and DS. This is a benefit of the ASP-implementation, particularly when examining all credulously or skeptically accepted arguments of a given WAF. To improve the runtime performance of the problems DC and DS, it is advisable to incorporate preprocessing techniques to focus specifically on the investigated argument and filter out arguments that are not relevant for the respective semantics.

Impact of the Relaxations

In our analysis of the impact of the relaxations, we observed that the relaxations have no significant influence on the runtime performance of the ASP-implementation. A difference in runtime for the problem EE is only observed when selecting relaxation parameters with high values. However, this difference is not specific to the ASP-implementation, but is due to the increased computed extensions. Regarding acceptability, we noted that under α^γ-stable semantics without the use of relaxations, hardly any credulously accepted arguments were found, and specifically for instances generated with the model Erdős–Rényi, high relaxation parameters had to be chosen to find acceptable arguments.

Impact of the Implementation

When comparing the runtime performance of the ASP-implementation with the runtime performance of ConArg, we observed that under admissible semantics for the problem DC, ConArg consistently exhibits better runtime performance for the investigated instances. Under both α^γ-complete and α^γ-stable semantics, the ASP-implementation demonstrated performance comparable to that of ConArg when investigating instances generated with the models Barabási-Albert and Watts-Strogatz, but was significantly slower for instances generated with the model Erdős–Rényi. The primary distinction between the implementations is that ConArg efficiently solves instances with a large number of attacks under various semantics, while the ASP-implementation shows slow runtime performance for these instances.

8 Conclusion

In this work, we have presented an initial approach to implement semiring-based weighted argumentation frameworks using ASP. We have introduced ASP-encodings for α-conflict-free sets, α^γ-admissible sets, α^γ-complete semantics and α^γ-stable semantics, which are compatible with the ASP system Clingo. We evaluated the ASP-implementation under various semantics and problem types, with generated instances of WAFs using different models and with instances from ICCMA 2019.

For α^γ-complete semantics, we have found that there are several definitions [7, 11] that differ slightly. Therefore, along with these definitions, we have presented an additional definition for α^γ-complete semantics, where the counterattack of an argument is also considered, but with the definition not being too general. This definition also matches with the functionality of ConArg. An alternative approach would be to adjust the notion of γ-defence to consider counterattacks, but in this work, we have adhered to the original definition of γ-defence by Bistarelli et al. [7].

For the development of the ASP-encodings, we adopted approaches from ASPARTIX. However, due to the underlying structure of the semiring and the resulting composite weight of attacks, the encodings differ significantly, especially noticeable in the ASP-encoding for α^γ-complete semantics. The ASP-encodings have not yet been optimized for performance, but are intended to demonstrate how an ASP-implementation of semiring-based weighted argumentation frameworks can be done.

For the evaluation of the ASP-implementation, we generated instances of WAFs using different models, and additionally, examined instances from ICCMA 2019. We investigated these instances for acceptability under various problems, semantics and relaxations parameters and also examined the runtime of the ASP-implementation. Furthermore, we compared the runtime performance of the ASP-implementation with that of ConArg. The ASP-implementation has the advantage of being able to compute the problem types EC and ES without runtime losses and shows good runtime performance under α^γ-complete and α^γ-stable semantics for various instances. However, for the problem DC under α^γ-admissible semantics and for instances with a high number of attacks, ConArg exhibits significantly better runtime performance compared to the ASP-implementation.

Future Work

For future work, several approaches can be pursued. The provided ASP-encodings can be further optimized by, for example, eliminating unnecessary computations, such as avoiding the computation of the internal conflict weight when dealing with WAFs where the relaxation of conflict-freeness is not required. Another possibility is the application of preprocessing techniques, as already described for AAFs in [20] for the problems DC and DS under α^γ-complete semantics.

Another promising approach involves creating ASP-encodings using labeling semantics. Definitions for labeling semantics are presented in [11], which also includes definitions for w-preferred and w-grounded semantics. An implementation with labeling semantics has the advantage that Clingo heuristics can be applied for the maximization of a set to investigate WAFs under α^γ-preferred semantics. However, for an implementation of α^γ-preferred and α^γ-grounded semantics the effects of relaxations must also be considered.

References

[1] Réka Albert and Albert-László Barabási. Topology of evolving networks: local events and universality. *Physical review letters*, 85(24):5234, 2000.

[2] Leila Amgoud and Cayrol Claudette. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34:197–215, 2002.

[3] Trevor J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.

[4] Stefano Bistarelli, Lars Kotthoff, Francesco Santini, and Carlo Taticchi. Summary report for the third international competition on computational models of argumentation. *AI magazine*, 42(3):70–73, 2021.

[5] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of ACM*, 44:201–236, 1997.

[6] Stefano Bistarelli, Fabio Rossi, and Francesco Santini. A conarg-based library for abstract argumentation. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 374–381, 2017.

[7] Stefano Bistarelli, Fabio Rossi, and Francesco Santini. A novel weighted defence and its relaxation in abstract argumentation. *International Journal of Approximate Reasoning*, 92:66–86, 2018.

[8] Stefano Bistarelli, Fabio Rossi, Francesco Santini, et al. Conarg: A tool for classical and weighted argumentation. In *COMMA*, pages 463–464, 2016.

[9] Stefano Bistarelli and Francesco Santini. A hasse diagram for weighted sceptical semantics with a unique-status grounded semantics. In *Logic Programming and Nonmonotonic Reasoning: 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings 14*, pages 49–56. Springer, 2017.

[10] Stefano Bistarelli and Francesco Santini. Weighted argumentation. *Journal of Applied Logics*, 8(6):1589–1622, 2021.

[11] Stefano Bistarelli and Carlo Taticchi. A labelling semantics for weighted argumentation frameworks. In *35th Edition of the Italian Conference on Computational Logic (CILC 2020)*, 2020.

[12] Álvaro Carrera and Carlos A. Iglesias. A systematic review of argumentation techniques for multi-agent systems research. *Artificial Intelligence Review*, 44:509–535, 2015.

[13] Claudette Cayrol, Caroline Devred, and Marie-Christine Lagasquie-Schiex. *Acceptability semantics accounting for strength of attacks in argumentation.* PhD thesis, IRIT-Institut de recherche en informatique de Toulouse, 2010.

[14] Claudette Cayrol and Lagasquie-Schiex Marie-Christine. On the acceptability of arguments in bipolar argumentation frameworks. *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 378–389, 2005.

[15] Federico Cerutti, Massimiliano Giacomin, Mauro Vallati, et al. Generating challenging benchmark afs. *COMMA*, 14:457–458, 2014.

[16] Sylvie Coste-Marquis, Sébastian Konieczny, Pierre Marquis, and Mohand Akli Ouali. Weighted attacks in arugmentation frameworks. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR*, pages 593–597. AAAI Press, 2012.

[17] Martin Diller, Wolfgang Dvořák, Jörg Pührer, Johannes Peter Wallner, and Stefan Woltran. Applications of ASP in formal argumentation. In *Proceedings of the Second Workshop on Theory and Applications of Answer Set Programming*, 2018.

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

[19] Paul E. Dunne, Anthony Hunter, Peter McBurney, Simon Parsons, and Michael Wooldridge. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175:457–486, 2011.

[20] Wolfgang Dvořák, Matthias König, Johannes P Wallner, and Stefan Woltran. Aspartix-v21. *arXiv preprint arXiv:2109.03166*, 2021.

[21] Wolfgang Dvořák, Anna Rapberger, Johannes Peter Wallner, and Stefan Woltran. Aspartix-v19 - an answer-set programming based system for abstract argumentation. In *Foundations of Information and Knowledge Systems - 11th International Symposium*, pages 79–89. Springer, 2020.

[22] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Aspartix: Implementing argumentation frameworks using answer-set programming. In *Logic Programming, 24th International Conference*, pages 734–738. Springer, 2008.

[23] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation 1*, 2:147–177, 2010.

[24] P ERDdS and A R&wi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

[25] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[26] Martin Gebser, Roland Kaminski, Arne König, and Thorsten Schaub. Advances in gringo series 3. In *Logic Programming, 11th International Conference*, pages 345–351. Springer, 2011.

[27] Martin Gebser, Benjamin Kaufmann, André Neumann, and Thorsten Schaub. clasp: A conflict-driven answer set solver. In *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

[28] Martin Gebser, Thorsten Schaub, and Sven Thiele. Gringo: A new grounder for answer set programming. In *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

[29] Souhila Kaci and Cristophe Labreuche. Arguing with valued preference relations. In *ECSQARU 11: Proceedings of the 11th European conference on Symbolic and quantitative approaches to reasoning with uncertainty*, pages 62–73. Springer, 2011.

[30] Benjamin Kaufmann, Nicola Leone, Simona Perri, and Thorsten Schaub. Grounding and solving in answer set programming. *AI Magazine*, 37(3):25–32, 2016.

[31] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7*, 3:499–562, 2006.

[32] Vladimir Lifschitz. *Answer Set Programming*, volume 3. Springer, 2019.

[33] Diego C. Martínez, Alejandro Javier García, and Guillermo Ricardo Simari. An abstract argumentation framework with varied-strength attacks. In *Priciples of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference*, pages 135–144. AAAI Press, 2008.

[34] Sanjay Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9):901–934, 2009.

[35] Martin Možina, Jure Žabkar, and Ivan Bratko. Argument based machine learning. *Artificial Intelligence*, 171:922–937, 2007.

[36] Henry Prakken and Giovanni Sartor. Law and logic: A review from an argumentation perspective. *Artificial Intelligence*, 227:214–245, 2015.

[37] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation.* Doctoral dissertation, Saarland University, 2009.

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

[39] Frans H. Van Eemeren, Bart Garssen, Erik C.W. Krabbe, A. Francisca Snoeck Henkemans, Bart Verheij, and Jean H.M. Wagemans. Classical backgrounds. In *Handbook of Argumentation Theory*, pages 51–139. Springer, 2014.

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